Using Retrieval Augmented Generation (RAG) on a Custom PDF Dataset with Dell Technologies

October 20th, 2023 | Read Time: 21 minutes

DO David O'Dell

The Generative AI transformation

Artificial Intelligence is transforming the entire landscape of IT and our digital lives. We've witnessed several major disruptions that have changed the course of technology over the past few decades. The birth of the internet, virtual reality, 3D printing, containerization, and more have contributed to major shifts in efficiency as well as the democratization of the tools required to create in those spaces.

Generative AI (GenAI) is now a major disruptor, forcing us all to rethink what efficiency leaps can, should, and should not be made with this new technology.

On its current trajectory, the larger AI industry has the potential to change entire economies. AI isn't tremendously new. Within the past decade, the bottlenecks that once held back progress have been removed by massive gains in GPU technology, abundant data availability, and vast oceans of distributed storage.

Nowadays, we must differentiate between **traditional AI** – used to perform specific tasks and make predictions based on patterns – and **GenAI** – used to create new data that resembles human-like content.

With GenAI large-language models (LLM) leading the pack of the latest AI innovations, let's pause for just a moment and ask ourselves, "Why is it suddenly so popular?", "How does it work?", and, more importantly, "How can I make it work better?" There's no better way to answer these questions than by diving into the code that makes GenAI such a hot item.

Our mission today with GenAI

If I were to ask you a random question, chances are you could answer it in a sophisticated, accurate, and grammatically correct way – a "human-like" way. If I asked you about cars, chances are the topic of tires might come up since cars and tires have a strong relationship.   Your response probably wouldn't contain anything about zebras since zebras and cars are not strongly related.   What if I asked you about a skyscraper?   The word, “building” is strongly related to skyscrapers, but why not the words, “moon” or “bird” - they’re in the sky too, right?

To achieve a response that pleases us, we want an accurate answer presented to us in a human-like manner. Those two concepts – “accuracy” and “human-like” – are the common threads woven throughout the code for all generative AI development.

A traditional AI response of “yes” or “no” can maintain high accuracy, but is that what I want to give to my users or my customers? A dry, robotic, binary response? Absolutely not. I want a human-like response that provides context and additional help. I want the response to solve my problem either directly through automated actions or indirectly by enabling me to help myself.   If we can’t get all of this, then why bother building any of it?

Having a human-like response is of tremendous value and something the market desires. So how do we humanize a computer created response? It takes brains.

Human brains are massive pattern-matching machines that rely on millions of physical neural connections. AI essentially mirrors those physical connections in the form of numerical relationship strings called vectors. Accuracy comes from thousands of interlocking relationships of general knowledge about individual things. Each “thing” you feed an AI model, whether it’s a pixel or a word, is digitized and labeled as a vector that has unique value and location, either on an image or in a sentence.

Once we digitize our content into a form the computer can digest, we can start to analyze it for patterns of relationships and eventually build a model that is good at providing accurate, human-like responses based on the relationships it was given.

Defining the problem

There are dozens of major LLMs to choose from and thousands of homebrewed variants. Each model supports unique features and use cases, so choosing the right one is vital.   Let’s first define our problem and then determine model selection.

Our example company would like to improve their overall customer experience when chatting with support. Besides improving the response time and providing a better self-help pathway, they would like to integrate their legacy and future knowledge base articles into a help desk chatbot that can respond to questions with new information obtained from their pdf dataset. In this example, we'll use a collection of white papers and infographics from Dell Infohub.

Training a model from scratch vs. fine-tuning vs. RAG

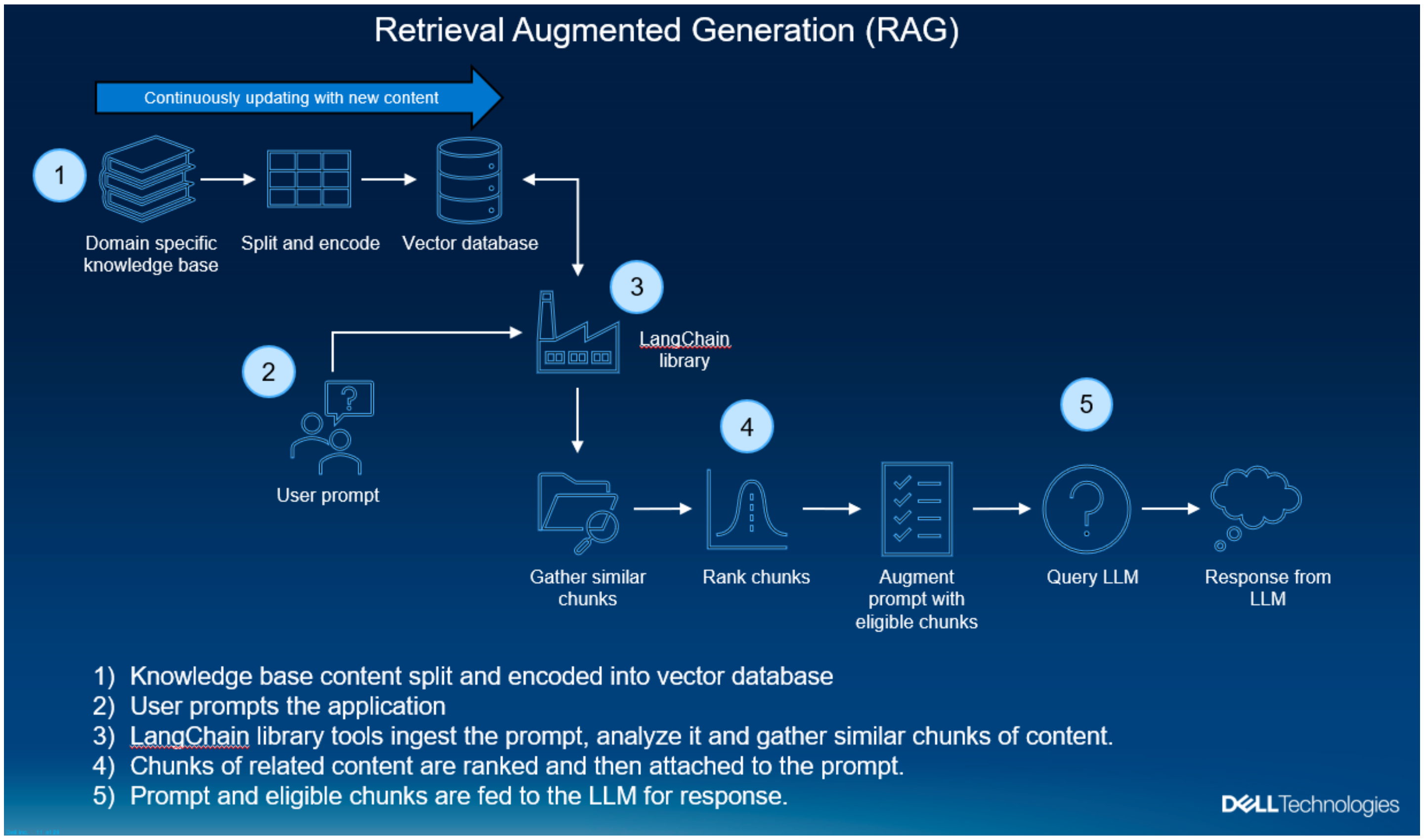
If your industry is highly specialized and has a considerably unique vocabulary - such as legal, medical, or scientific – or your business requires a high level of privacy where intermingling publicly and privately sourced data is forbidden, training a model from scratch might be the route to take.

In most cases, using an existing open-source model and then fine-tuning it to enable a new task is preferred since it requires much less compute and saves a lot of time. With the right balance of compute, storage, and software, fine-tuning an existing model can be extremely effective.

If your response model is good and performs the task you want but could use some specific learning based on content from a custom document dataset – a knowledge base for example - then Retrieval Augmented Generation (RAG) would be a great candidate for this type of workload.

Why use Retrieval Augmented Generation?

Retrieval Augmented Generation (RAG) is used in LLM applications to retrieve relevant knowledge base-style content, augment the user prompt with this domain-specific content, then feed both the prompt and content into the LLM to generate a more complete, useful response.


Retrieval Augmented Generation (RAG)
Continuously updating with new content
1
Domain specific knowledge base
Split and encode
Vector database
2
User prompt
3
LangChain library
4
5
Gather similar chunks
Rank chunks
Augment prompt with eligible chunks
Query LLM
Response from LLM
DELL Technologies

1) Knowledge base content split and encoded into vector database
2) User prompts the application
3) LangChain library tools ingest the prompt, analyze it and gather similar chunks of content.
4) Chunks of related content are ranked and then attached to the prompt.
5) Prompt and eligible chunks are fed to the LLM for response.

Figure 1. Understanding how RAG works

So how does RAG work? Imagine yourself at a restaurant, asking the waiter a question about wine pairings. In this analogy, you are the user who is inputting a question prompt, and the waiter is our LLM model. The waiter certainly has basic pairing suggestions – "Red wine pairs well with beef" – but this response speaks nothing to centuries of pairing history, recent wine trends, or this restaurant's decades of culture. This very dry response also doesn't account for the restaurant's current inventory. We wouldn't want the waiter to suggest a wine that isn't currently stocked.

With a RAG process, the waiter takes the question, retrieves relevant historical and up-to-date information specific to the restaurant (including inventory), and gathers it all together for the customer. That being said, it's not enough to just retrieve information. Our customer needs the finer touch of a well-informed suggestion rather than being inundated with a bunch of articles or snippets about wine pairings. That's where the LLM shines.

LLMs are excellent at taking large, disparate chunks of content, organizing them, and providing a human-like response. The original question, along with all those wine and food snippets about pairing, are fed into the LLM whereby a more complete, useful response is given, all without having to relearn the basics. That is to say, our waiter didn't have to become a sommelier to give this response. No retraining was required. The RAG process doesn't require time-consuming training runs.   The LLM is trained prior to engaging in the process. We simply made new domain-specific knowledge easier for the LLM to digest.

Peeking under the hood

This is all done by taking domain-specific knowledge bases (in this case pdf files), splitting them up intelligently, then encoding the textual content of these chunks into long numerical vectors.

Vectors representing the original text go into a vector database that can be queried extremely quickly. Vector databases come in a variety of types and use cases. In this example, we're using ChromaDB, a "pure" vector database that is designed to store and retrieve vectors from unstructured data, such as text, images, and files. This is perfect for our use case where we are taking random text from unknown documents in a variety of formats and converting them to vectors that are used to build relationships between the prompt and chunks of content.

In some ways, we can consider the vector database to be a form of long-term memory. As we continue to add new content to it and keep it maintained, our LLM can refer to the contents as the database expands with new information.

Using the original question, the vector database is queried to find which chunks are most related to the original question. The results are ranked for similarity whereby only the most relevant content is eligible to be fed into the LLM for response generation.

Choosing our LLM

We'll be using the Meta Llama2 model since it can be quantized and run locally on prem, comes in a variety of sizes, performs as well or better than ChatGPT, and is also available for free for commercial use.   Llama2 also has a moderately large context window that allows users to introduce text in the form of sentences or entire documents and then generate responses from the new information.

Using Llama2 or other open-source LLMs on prem allows full control over both your domain-specific content and any content that goes into the model, such as prompts or other proprietary information. On prem models are also not executable objects on their own since they lack the ability to send your private data back to the original authors.

Compute and Storage Environment

Our physical environment is on VMware vSphere as the hypervisor in a Dell APEX Private Cloud cluster with VxRail PowerEdge nodes, each with 3 Nvidia T4 GPUs. Storage is on the local vSAN. Our notebook server is running inside a Pytorch virtual environment on an Ubuntu 22.04 virtual machine with Miniconda. This can also be run on bare metal PowerEdge with any OS you choose as long as you can run Jupyter notebooks and have access to GPUs.

GenAI coding first steps

When running any sort of training or fine-tuning, you'll need access to models, datasets, and monitoring so that you can compare the performance of the chosen task. Luckily, there are free open-source versions of everything you need. Simply set up an account on the following sites and create an API access token to pull the leading models and datasets into your notebooks.

- **Hugging Face** – incredibly valuable and widely used open-source python libraries, datasets, models, notebook examples, and community support
- **Weights and Biases** – free SaaS-based monitoring dashboard for model performance analysis
- **Github** – open-source libraries, tools, and notebook examples

Along with those sites, at some point you will inevitably experiment with or use incarnations of the Meta (Facebook) Llama model. You'll need to fill out this simple permission form.

- **Llama model permission form**: https://ai.meta.com/resources/models-and-libraries/llama-downloads/

Setting up RAG on the Llama2 model with a custom PDF dataset

First, let's log in to Huggingface so that we can access libraries, models, and datasets.

```
## code to auto login to hugging face, avoid the login prompt

!pip install -U huggingface-hub

# get your account token from https://huggingface.co/settings/tokens
token = '<insert your token here>'

from huggingface_hub import login
login(token=token, add_to_git_credential=True)
```

With each notebook you run, you'll end up installing, upgrading, and downgrading all sorts of libraries. The versions shown may very well change over time with added or deprecated features. If you run into version compatibility issues, try upgrading or downgrading the affected library.

```
!pip install torch
!pip install transformers
!pip install langchain
!pip install chromadb
!pip install pypdf
!pip install xformers
```

```
!pip install sentence_transformers
!pip install InstructorEmbedding
!pip install pdf2image
!pip install pycryptodome
!pip install auto-gptq
```

From our newly installed packages, let's import some of the libraries we'll need. Most of these will be related to LangChain, an amazing tool for chaining LLM components together as previously seen in figure 1. As you can see from the library titles, LangChain can connect our pdf loader and vector database and facilitate embeddings.

```
import torch
from auto_gptq import AutoGPTQForCausalLM
from langchain import HuggingFacePipeline, PromptTemplate
from langchain.chains import RetrievalQA
from langchain.document_loaders import PyPDFDirectoryLoader
from langchain.embeddings import HuggingFaceInstructEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import Chroma
from pdf2image import convert_from_path
from transformers import AutoTokenizer, TextStreamer, pipeline

DEVICE = "cuda:0" if torch.cuda.is_available() else "cpu"
```

Let's check our Nvidia GPU environment for availability, processes, and CUDA version. Here, we see 3 x T4 GPUs. The driver supports CUDA version up to 12.2 with around 16Gb per device, and no other running processes. Everything looks good to start our run.

```
!nvidia-smi

+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.113.01             Driver Version: 535.113.01    CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp    Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:0B:00.0 Off |                  Off |
| N/A   46C    P8              10W /  70W |      5MiB / 16384MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   1  Tesla T4                       Off | 00000000:14:00.0 Off |                  Off |
| N/A   30C    P8              10W /  70W |      5MiB / 16384MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   2  Tesla T4                       Off | 00000000:1D:00.0 Off |                  Off |
| N/A   32C    P8               9W /  70W |      5MiB / 16384MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU    GI   CI        PID    Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Let’s check to make sure we can reach some of the pdfs in our repo by placing a pdf file page thumbnail image into an array and calling it for a preview.

```
pdf_images = convert_from_path("pdfs-dell-infohub/apex-navigator-for-multicloud-storage-solution-overview.pdf", dpi=100)
pdf_images[0]
```

SOLUTION BRIEF

Dell APEX Navigator for Multicloud Storage

Simple SaaS-based management for Dell storage deployed across public clouds

Enterprise cloud storage

- Deploy and manage innovative Dell storage across public clouds
- Maintain control of data across on-premises and public cloud
- Accelerate Zero Trust adoption

Extreme flexibility

- Drive data mobility between on-premises and public clouds
- Take decisive action based on intelligent insights
- Consistent Dell storage management, no matter where it's deployed

Enhanced productivity

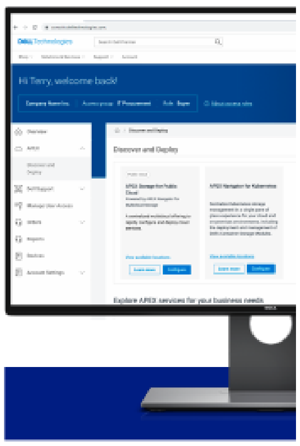
- Centralized multicloud storage management in Dell APEX console
- Rapidly deploy with automated provisioning and deployment
- Integrate with your automation platforms

Managing today's complex business and IT environment

Organizations in every industry are seeking new ways to optimize their IT strategy, including embracing the on-demand scaling, quick provisioning and access to advanced services offered by cloud providers, often utilizing multiple providers at once. However, according to the Dell Innovation Index, management complexity and siloed experiences are cited as one of the top five roadblocks preventing organizations from pursuing multicloud environments.

Solution: Dell APEX Navigator for Multicoud Storage

Dell APEX Navigator for Multicloud Storage simplifies multicloud storage management complexity by breaking down siloed experiences with centralized management of Dell storage endpoints across multiple public clouds, all from within Dell APEX Console. This management tool enables ITOps and storage admins to deploy, configure and manage Dell storage in public clouds, monitor Dell storage, and seamlessly move data across on-premises and public clouds.



Accelerate deployments across public clouds

Rapidly deploy Dell storage in public cloud, in four easy steps, with simple configuration and automated provisioning of underlying public cloud resources and automated deployment of Dell storage software.

Centralized, consistent management

Centralize day-to-day storage management activities with link and launch access to familiar Dell storage management tools for a consistent experience across public cloud and on-premises. This centralized access to consistent experiences eliminates the need for retraining and improves the efficiency of lifecycle management activities.

Embrace automation

Dell APEX Navigator for Multicloud Storage is API-first, allowing you to integrate your preferred automation tools such as Ansible and Terraform, so it's simple to leverage Dell storage software as part of your broader cloud and IT automation strategy. Utilize public APIs to automate deployment, management and decommissioning of Dell storage in public clouds and move data from on-premises into public cloud and back.

Let’s call the pdf directory loader from LangChain to get an idea of how many pages we are dealing with.

```
loader = PyPDFDirectoryLoader("pdfs-dell-infohub")
docs = loader.load()
len(docs)
```

```
791
```

Next, let’s split the pages into chunks of useful data. The downloaded `hkunlp/instructor-large` model helps us split this intelligently rather than via a brute force algorithm. We use the embeddings from this model to recognize our new content. Here we see that we’ve split this into over 1700 chunks.

```
embeddings = HuggingFaceInstructEmbeddings(
    model_name="hkunlp/instructor-large", model_kwargs={"device": DEVICE}
)


text_splitter = RecursiveCharacterTextSplitter(chunk_size=1024, chunk_overlap=64)
texts = text_splitter.split_documents(docs)
len(texts)
```

```
1731
```

Next, we prepare our LLM to receive both the prompt and the relevant chunks from our LangChain retrieval process. We’ll be using a variant of the Llama2 13 billion parameter model that provides memory optimized (quantized) revisions for us to download.

```
model_name_or_path = "TheBloke/Llama-2-13B-chat-GPTQ"
model_basename = "model"

tokenizer = AutoTokenizer.from_pretrained(model_name_or_path, use_fast=True)

model = AutoGPTQForCausalLM.from_quantized(
    model_name_or_path,
    revision="gptq-4bit-128g-actorder_True",
    model_basename=model_basename,
    use_safetensors=True,
    trust_remote_code=True,
    inject_fused_attention=False,
    device=DEVICE,
    quantize_config=None,
)
```

Since this is a chatbot, we need to interact with our model directly. We do this by installing Huggingface’s pipeline module that facilitates access to your model and creates a raw chat interactive session directly from your notebook code cells.

```
text_pipeline = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=1024,
    temperature=0,
    top_p=0.95,
    repetition_penalty=1.15,
    streamer=streamer,
)
```

Our prompt is vital in this entire effort. We need to tell the LLM how to behave with responses. This system prompt, based on the default prompt for the Llama2 model, will provide enough instruction to get us human-like results once our content and question are fed into it.

```
SYSTEM_PROMPT = "Use the following pieces of context to answer the question at the end. If y
ou don't know the answer, just say that you don't know, don't try to make up an answer."

template = generate_prompt(
    """
{context}

Question: {question}
""",
    system_prompt=SYSTEM_PROMPT,
)
```

Finally, our chain can be built. LangChain links the retriever and our LLM together, then stuffs the document chunks into the prompt and passes it along as a normal query to the LLM while also asking for the source document chunks.

```
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=vectordb.as_retriever(search_kwargs={"k": 2}),
    return_source_documents=True,
    chain_type_kwargs={"prompt": prompt},
)
```

Let’s ask it a question that could only be found in the new documents. In this example, we’ve chosen a very specific question the Llama2 model was never trained on: “Does APEX block storage support multi availability zones?” The response given is “yes”. Even though the answer is positive, the model goes into a lot more detail on what it did find, giving us a very useful, human-like response.

We can also prove the source of truth by using the `return_source_documents` feature of LangChain and returning that in the next cell. This way, there is no question of whether the response was part of a hallucination.

```
result = qa_chain("Does apex block storage support multi availability zones?")
```

```
 Based on the information provided in the text, the answer is yes. APEX Block Storage suppor
ts multi-availability zones (AZs). The text states that "data is distributed across three or
more availability zones" using fault sets, providing resilience without replication or unnec
```

```
essary copies of the data.


print(result["source_documents"][0].page_content)

of data or needing to use replication across AZs.  Leveraging the native fault sets feature,
data is distributed across three or more availability zones to offer additional protection a
gainst AZ failure.
Well Suited for Challenging Workloads
APEX Block Storage provides proven enterprise data services, such as thin provisioning, snap
shots, replication , volume migration,  and backup/restore to S3, which are needed to run mi
ssion -critical  workloads confidently on the public cloud . With it s extre me performance
and scalability, APEX Block Storage  is well suited to support very large databases, analyti
cs workloads, and multiple container  (Kubernetes) development and production deployments.
And with the enterprise -class services and resilien ce provided in the cloud , APEX Block S
torage  for AWS and Microsoft Azure  is the ideal solution to run your most challenging work
loads in the public cloud wi th confidence that you can meet your SLAs.

More Information



---------------------------------------------------
```

Let’s ask an even more specific question: “Provide a curl example of code to authenticate my PowerFlex.”

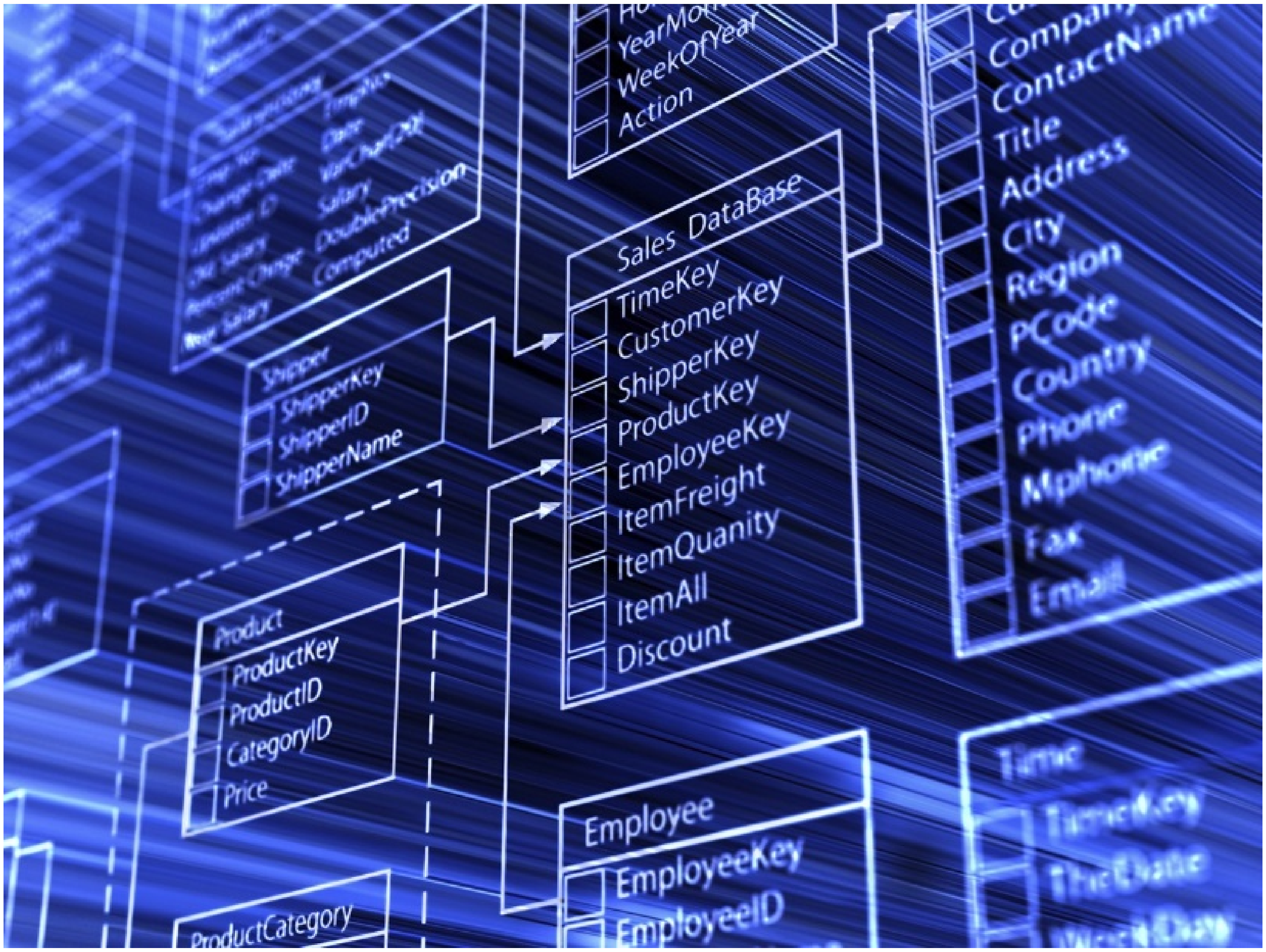
The LLM delivers a very pleasant response that is well-formatted, polite, and useful.

~~~
result = qa_chain("provide a curl example of code to authenticate my powerflex")


 Based on the provided context, here is an example of how to authenticate with PowerFlex usi
ng curl:
```bash
curl --location --request POST 'https://<PFXM>/rest/auth/login' \
 --header 'Content-Type: application/json' \
 --data-raw '{"username": "<username>", "password": "<password>"}'
```
This example uses the `POST` method to send a login request to the PowerFlex Manager, specif
ying the username and password in the request body. The `--header` option sets the `Content-
Type` header to `application/json`, and the `--data-raw` option sends the login credentials
in JSON format.

Note that you will need to replace `<PFXM>` with the actual hostname or IP address of your P
owerFlex Manager, and `<username>` and `<password>` with your PowerFlex username and passwor
d, respectively.
~~~

Bringing AI to your data

We've shown how the mission of GenAI is to provide accurate, human-like responses and that RAG is a low-impact method to augment those responses with your own custom content. We went through some of the tools that help facilitate this process, such as LangChain, vector databases, and the LLM model itself.

To really make these models shine, you need to apply your own data, which means having data sovereignty as well as secure access to your data. You simply can't afford to have private data leak, potentially being captured and uncontrollably exposed globally.



Your unique data has immense value. Dell is here to help bring AI to your data and achieve the best possible results with preconfigured or custom solutions catered to your business needs regardless of trajectory, whether it's using RAG, fine-tuning, or training from scratch.

With Dell Validated Design for Generative AI with Nvidia, customers can optimize the deployment speed of a modular, secure, and scalable AI platform. Dell PowerEdge servers deliver high performance and extreme reliability and can be purchased in a variety of ways, including bare metal, preconfigured with popular cloud stacks like our APEX Cloud Platforms, and as a subscription through Dell APEX. Simplify your structured or unstructured data expansion for GenAI with PowerFlex, PowerScale or ObjectScale, deployed on prem or as a subscription in the major cloud providers. Dell doesn't just stop at the data center. With Dell Precision AI workstations in the workplace, data scientists can speed innovation on the most intensive workloads.

If you have any questions or need expert assistance, Dell Professional Services can help craft an enterprise GenAI strategy for high value uses cases and the roadmap to achieve them.

Dell enables you to maintain data sovereignty and control while simplifying GenAI processes, providing the outcomes you demand with the flexible financing options you deserve.

Resources

- Code example for this notebook and others at: https://github.com/dell-examples
- Dell Technologies Infohub: https://infohub.delltechnologies.com/
- Original inspiration repo: https://github.com/curiousily/Get-Things-Done-with-Prompt-Engineering-and-LangChain

Author: David O'Dell, Technical Marketing Engineer, AI and Solutions