Genius+ cloud architecture

This is a description of a production-ready Azure Cloud architecture for Question Answering using LLMs based on what Microsoft proposes [link]. This document will explain in detail how the different functions work together and why they are needed. Later we will add to this architecture for our specific use case, but this is a great base to use as a jumping off point.

Contents

Figure 1 shows a first proposal of an QnA LLM architecture built in Azure. In our case (5) will not be a direct interaction with the User through a chatbot environment but instead through an API call built within a clients own chatbot. We deliver the advances feature of UI guidance.

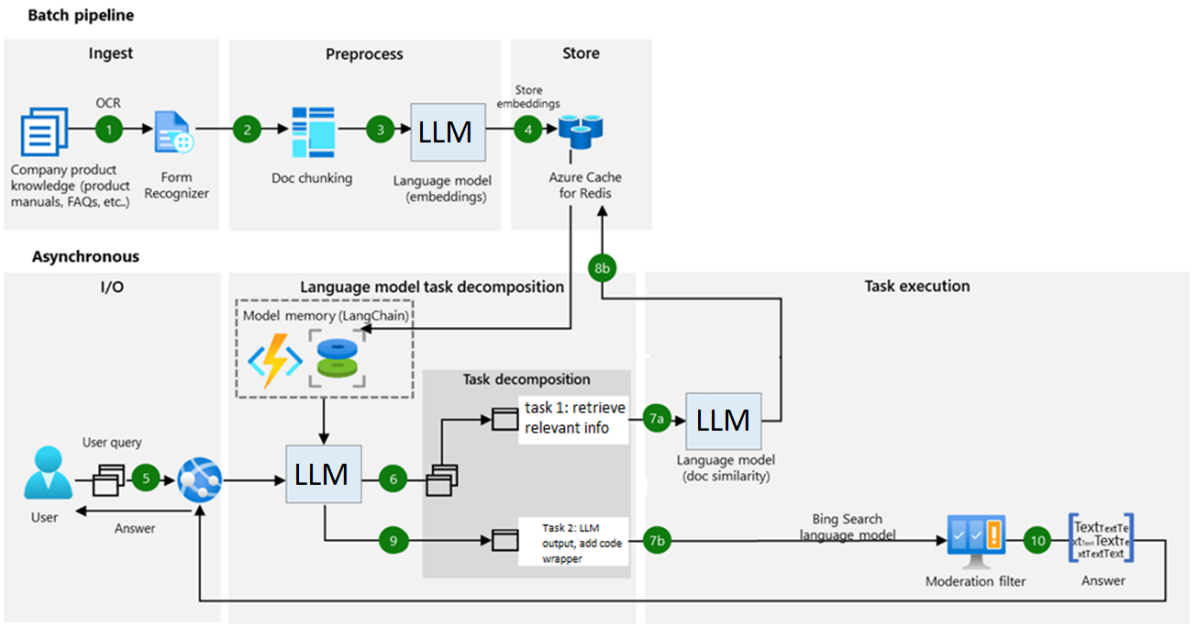


Figure 1: Azure architecture for QnA using LLMs

This workflow consists of 2 pipelines: Batch pipeline and (real-time) Asynchronous pipeline.

Batch pipeline

- The goal of this pipeline is to Batch process new information and turning the texts into a vector store. The vector store will then contain all knowledge that only our LLM has access to compared to other companies LLMs. The LLM will search through the vector store to find specific answers it wouldn't otherwise be able to find.

Services needed:

- **Blob storage**: for storing the documents in the cloud
- **Azure functions**: Hosts python code in the cloud. Can trigger when it finds new files in blob storage. It can then process the text and turn it into vectors (langchain (python library), just found Semantic Kernel from MS as an Azure alternative to langchain).

Storing embeddings options:

- **Azure Redis enterprice** [link]: one way to store and search embeddings in Azure
- **Pinecone** [link]: Alternative way of storing and searching embeddings. Becoming popular.
- **Azure Cosmos DB** [link]: Another way to store and search embeddings.

I don't yet know which the best embedding storing service for our problem is. Each one will likely do the job perfectly fine. Will talk with an expert from MS to gain insights. Not a blocker tho, since all are great options.

(real-time) Asynchronous pipeline

- This pipeline triggers (in this example) when a user asks a question. For our first implementation it would be the same, but eventually it is better to keep waiting for a particular behavioral pattern indicating that the user is lost. Then automatically run the flow to generate an answer to the problem even before the user has time to formulate the question themselves.

This pipeline consists of three parts:

- **I/O**

  The Input/Output layer. This is the layer where users interact with the software. In the first version this would be the chatbot interface. There are various ways to set up a chatbot front-end. My experience here is slightly lacking, as I only know how it's done in Azure. Since we use Azure that might be good, but I mention some popular alternatives.

  Required services:

  - **Azure Bot Framework** [link]: as it says a framework with many features to enable a chat interface for end users. It does not contain any chat logic, that happens in the second part of this pipeline.
  - **Drift** [link]: There are various alternatives to Azure bot framework, this apparently is one of the best. What I really like about Drift is that it can integrate with many different services such as Salesforce out of the box [link].
  - **Botkit** [link]: Seems a great lightweight node.js alternative that integrates with Azure. Online there are people who used botkit and Azure bot framework who prefer the Azure alternative [link].

- **Language model task composition**
  This layer receives the question and aquires the intent of the question. What is the user expecting and what are the steps needed to get the info to satisfy this expectation. The example figure shows two tasks (7a, internal document search and 8a, internet search). It gets the relevant info into model memory and then extracts what is most relevant (9). The last layer will send the output back to the end user.
  Required services:
    - **Bot composer/skills** [link]: This is a feature from Azure bot framework for acquiring intents. There are two approaches here 'bot composer' or 'skills' [link]. A skill is a different chatbot underlying the original bot with its own logic. This way the front end can be shared with different companies, but the underlying logic is completely separate. 'bot composer' is similar but it enables sharing logic between different instances.
    - **Azure Functions** [link]: This is a method to run code in the cloud. We will use Python and specifically the langchain library to work with vectors and make api calls to the LLM of our choice.
- **Task execution**
  The last step is to send the output back to the end user. This can be done directly by connecting to the Azure bot Framework (or whichever framework is chosen). It is possible to add a moderation filter before showing the end user the results.
    - **Azure AI studio (content filtering)** [link]: If we are first using openAI then Azure has an easy way to filter output before sending it further down the line. Long term we want an alternative as we don't want to depend on openAI.
    - **Azure Functions (Python logic):** In the same or a different Azure function we can add python logic that filters the output it has generated. We can use langchain and other conventional NLP methods. This will give us full control, is flexible but takes more time to set up and we might miss an important thing to filter out.

Train an LLM [link]

Why train your own LLM:

- Customization: tailor to specific needs and capabilities.
- Reduced dependency: control over your own model.
- Cost efficiency: using LLMs is expensive.

Great resource: [link]
Use huggingface as the first place to find relevant datasets and pre-trained models. It can be considered to start with a open source pre-trained model and build on top of that. Databricks is often used to build the datapipeline (and also host the model).

1) **Get data**
   In our case we want website HTML to train the model on the functionality of these websites. Website code has a copyright so we cannot use a webscraper to get the data and train a model. We can access open source website code as a base to improve a pre-trained model as a start.

2) **Preprocess data**
   Databricks is often used to build the data preprocessing pipelines for LLMs. In combination with Apache Spark to parallelize the dataset builder process. Preprocessing consists of the conventional steps of removing duplicate data, fix encoding issues and remove any personal data or data with copyright.
3) **Tokenization and vocabulary training**
   A custom vocabulary is not required but if the use case of the LLM is significantly different from the foundational model it is based on it will improve the functionality. A custom vocabulary allows a model to better understand input and generate output. This is trained on a random sample of the same data as used for model training. Then we do tokenization. This is similar to the concept of encryption most people understand in that it secures the data before use for training.
4) **Model training (with MosaicML)** [link]
   Now that we have our dataset prepared and a base model, using MosaciML we can train the model. This in itself requires time to decide on the proper parameters of the model. There are multiple trade-offs that need to be considered. Using MosaicML we eventually get a properly trained new model. Note that the costs of training a new model can be significant, depending on the size of the dataset.
5) **Evaluation**
   Evaluating models is a process that is in the early stages of development. A common way is to use the HumanEval framework described here [link]. Keep up-to-date with resplit.com as they are developing a more automated process for this. Likely there are more startups building a similar process.
6) **Deployment to production**
   Deploying the model is not a very difficult process anymore using for instance Databricks or an environment using Kubernetes. For now I propose using databricks, but very likely any cloud provider will enable this functionality in the foreseeable future.
7) **Feedback and iteration**
   Replit.com mentions the following '*Our model training platform gives us the ability to go from raw data to a model deployed in production in less than a day. But more importantly, it allows us to train and deploy models, gather feedback, and then iterate rapidly based on that feedback.'* Retraining a model on newly gathered data will over time improve the model. Note that retraining a model is expensive and not done often.

Fine-tune an LLM

1) Autotrain

It has recently become quite convenient to fine-tune a LLM on our own data through the use of AutoTrain [link].

AutoTrain supports the following types of LLM finetuning:

- Causal Language Modeling (CLM)
- Masked Language Modeling (MLM) [Coming Soon]

Note: For LLM finetuning, only Hugging Face Hub model choice is available.

2) Azure openAI studio

This is a feature in Azure to fine-tune openAi related models. This might be a good option for us too if we are planning to use Azure and quickly want to build functionality not possible with standard openAI LLMs [link].

The question is, why should you fine-tune an LLM instead of building one from the ground up?

Simply put, building and training a model from scratch can take multiple weeks and cost between 100k – 200k dollar. This is not realistic or worth it in most situations [link]. It is possible to not train a LLM but a transformer model such as bert. This is much cheaper and faster. The issue here is that transformer models on their own can perform one task well. If you want emergent abilities or a conversation with the model it needs to be a LLM at this point. Fine tuning a LLM is then often the right approach. In our case that will be true also.

Get website HTML for QnA

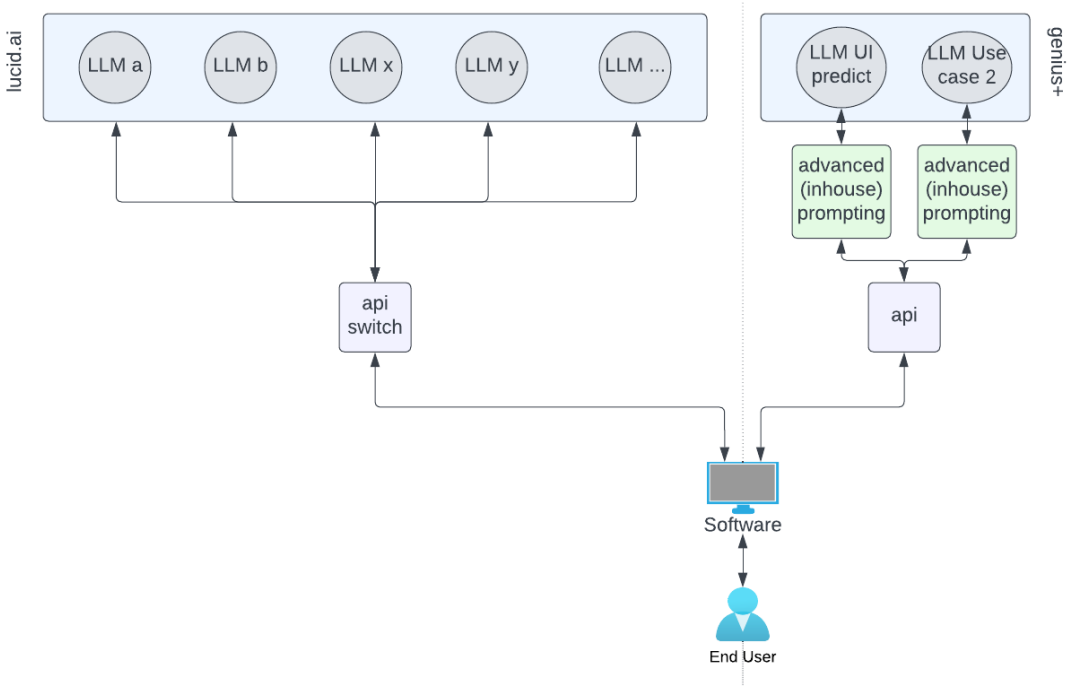
This can simply be shared within the API call. Developers have access to their own code within the application, but it is also possible to use a webscraper to aquire the html.

Get website HTML for fine-tuning an LLM

Setting up a webscraper to get the html from any website is not difficult to do [link]. The challenge is to make sure this code has no copyright so it can be used for fine-tuning the LLM. Here are a few website with open source website designs. This can be the starting point to build the required dataset:

- http://www.opendesigns.org/
- https://themewagon.com/theme-tag/open-source/

Alternative architecture



Genius+ is at 2 levels of the value chain
1) identify use cases (UI predict)
2) Build solutions (models & advanced prompting)

We foresee that the future value of LLMs lie not only in the platform to host and train LLMs (lucid.ai) but specifically in identifying use cases, becoming experts in specific use cases and deploying solutions for them. We have identified **UI guidance** as a key solution that most people have a desire for that LLMs are uniquely suited for to solve.