Exploring and comparing different LLMs

Introduction

This lesson will cover:

- Different types of LLMs in the current landscape.
- Testing, iterating, and comparing different models for your use case in Azure.
- How to deploy an LLM.

Learning Goals

After completing this lesson, you will be able to:

- Select the right model for your use case.
- Understand how to test, iterate, and improve performance of your model.
- Know how businesses deploy models.

Understand different types of LLMs

LLMs can have multiple categorizations based on their architecture, training data, and use case. Understanding these differences will help our startup select the right model for the scenario, and understand how to test, iterate, and improve performance.

There are many different types of LLM models, your choice of model depends on what you aim to use them for, your data, how much you're ready to pay and more.

Depending on if you aim to use the models for text, audio, video, image generation and so on, you might opt for a different type of model.

- **Audio and speech recognition**. For this purpose, Whisper-type models are a great choice as they're general-purpose and aimed at speech recognition. It's trained on diverse audio and can perform multilingual speech recognition. Learn more about Whisper type models here.
- **Image generation**. For image generation, DALL-E and Midjourney are two very known choices. DALL-E is offered by Azure OpenAI. Read more about DALL-E here and also in Chapter 9 of this curriculum.
- **Text generation**. Most models are trained on text generation and you have a large variety of choices from GPT-3.5 to GPT-4. They come at different costs with GPT-4 being the most expensive. It's worth looking into the Azure OpenAI playground to evaluate which models best fit your needs in terms of capability and cost.
- **Multi-modality**. If you're looking to handle multiple types of data in input and output, you might want to look into models like gpt-4 turbo with vision or gpt-4o -

Foundation Models versus LLMs

The term Foundation Model was coined by Stanford researchers and defined as an AI model that follows some criteria, such as:

- **They are trained using unsupervised learning or self-supervised learning**, meaning they are trained on unlabeled multi-modal data, and they do not require human annotation or labeling of data for their training process.
- **They are very large models**, based on very deep neural networks trained on billions of parameters.
- **They are normally intended to serve as a 'foundation' for other models**, meaning they can be used as a starting point for other models to be built on top of, which can be done by fine-tuning.

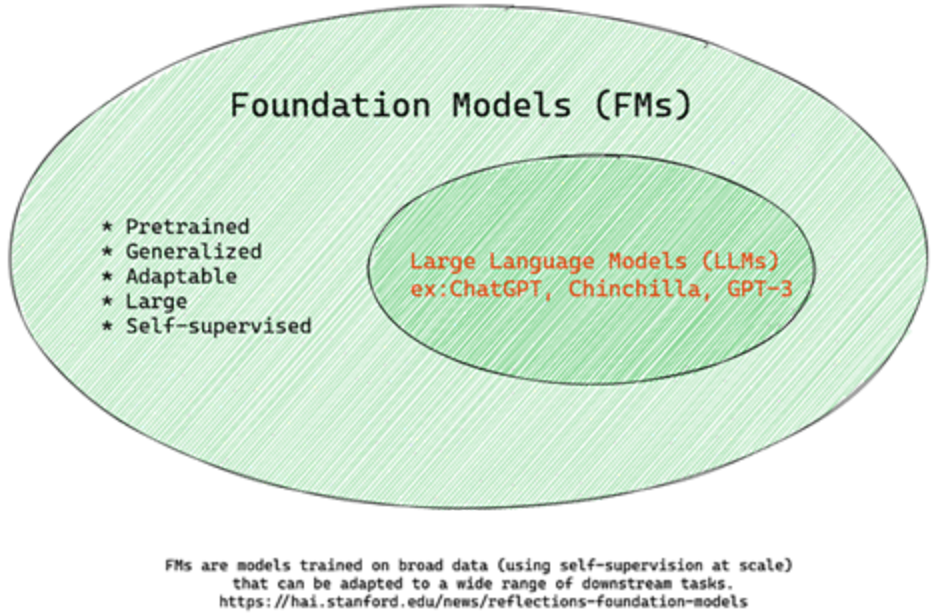
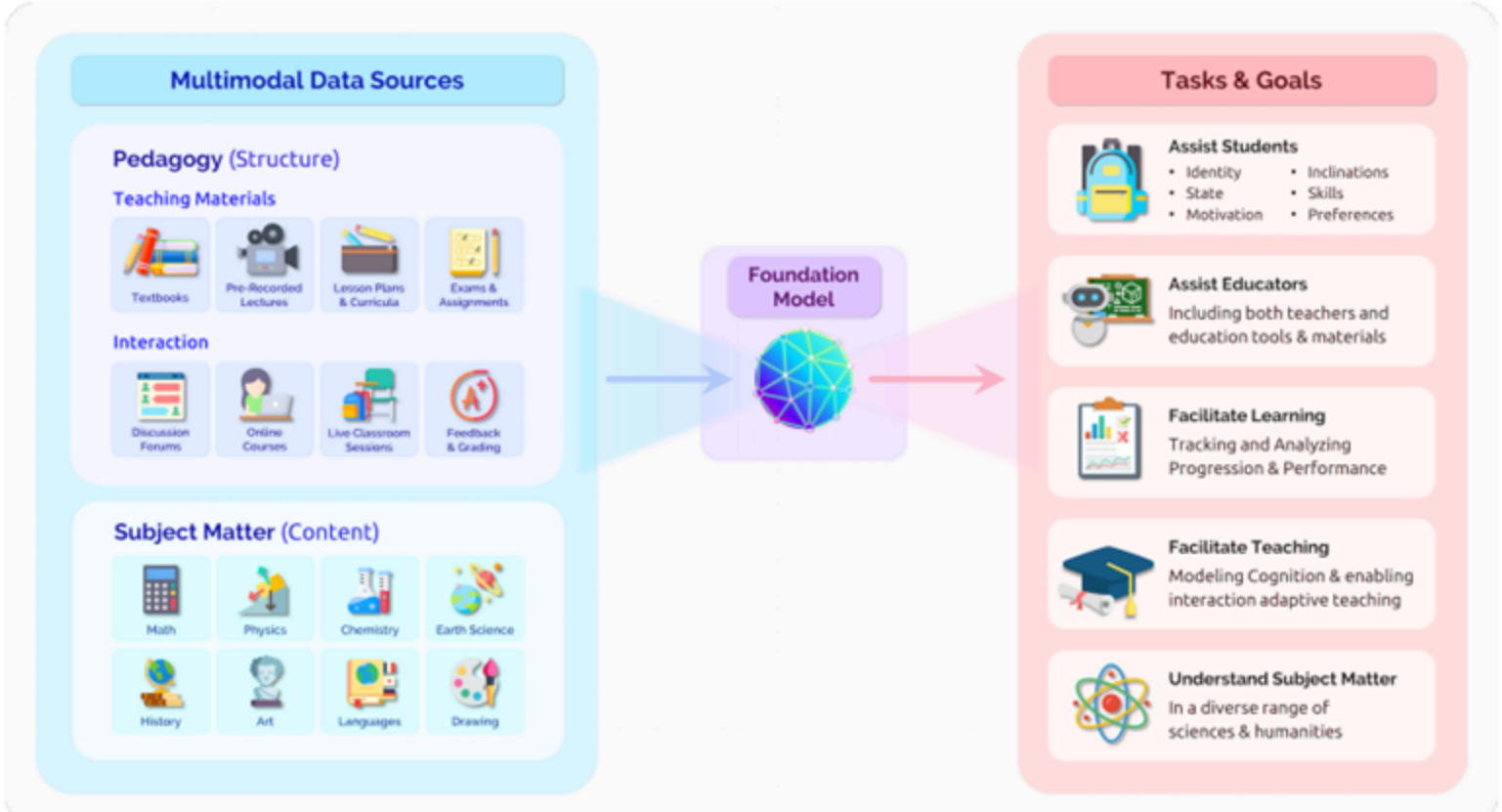


Image source: Essential Guide to Foundation Models and Large Language Models | by Babar M Bhatti | Medium

To further clarify this distinction, let's take ChatGPT as an example. To build the first version of ChatGPT, a model called GPT-3.5 served as the foundation model. This means that OpenAI used some chat-specific data to create a tuned version of GPT-3.5 that was specialized in performing well in conversational scenarios, such as chatbots.



Open Source versus Proprietary Models

Another way to categorize LLMs is whether they are open source or proprietary.

Open-source models are models that are made available to the public and can be used by anyone. They are often made available by the company that created them, or by the research community. These models are allowed to be inspected, modified, and customized for the various use cases in LLMs. However, they are not always optimized for production use, and may not be as performant as proprietary models. Plus, funding for open-source models can be limited, and they may not be maintained long term or may not be updated with the latest research. Examples of popular open source models include Alpaca, Bloom and LLaMA.

Proprietary models are models that are owned by a company and are not made available to the public. These models are often optimized for production use. However, they are not allowed to be inspected, modified, or customized for different use cases. Plus, they are not always available for free, and may require a subscription or payment to use. Also, users do not have control over the data that is used to train the model, which means they should entrust the model owner with ensuring commitment to data privacy and responsible use of AI. Examples of popular proprietary models include OpenAI models, Google Bard or Claude 2.

Embedding versus Image generation versus Text and Code generation

LLMs can also be categorized by the output they generate.

Embeddings are a set of models that can convert text into a numerical form, called embedding, which is a numerical representation of the input text. Embeddings make it easier for machines to understand the relationships between words or sentences and can be consumed as inputs by other models, such as classification models, or clustering models that have better performance on numerical data. Embedding models are often used for transfer learning, where a model is built for a surrogate task for which there's an abundance of data, and then the model weights (embeddings) are re-used for other downstream tasks. An example of this category is OpenAI embeddings.

Converter

Image generation models are models that generate images. These models are often used for image editing, image synthesis, and image translation. Image generation models are often trained on large datasets of images, such as LAION-5B, and can be used to generate new images or to edit existing images with inpainting, super-resolution, and colorization techniques. Examples include DALL-E-3 and Stable Diffusion models.

Text and code generation models are models that generate text or code. These models are often used for text summarization, translation, and question answering. Text generation models are often trained on large datasets of text, such as BookCorpus, and can be used to generate new text, or to answer questions. Code generation models, like CodeParrot, are often trained on large datasets of code, such as GitHub, and can be used to generate new code, or to fix bugs in existing code.

Encoder-Decoder versus Decoder-only

To talk about the different types of architectures of LLMs, let's use an analogy.

Imagine your manager gave you a task for writing a quiz for the students. You have two colleagues; one oversees creating the content and the other oversees reviewing them.

The content creator is like a Decoder only model, they can look at the topic and see what you already wrote and then he can write a course based on that. They are very good at writing engaging and informative content, but they are not very good at understanding the topic and the learning objectives. Some examples of Decoder models are GPT family models, such as GPT-3.

The reviewer is like an Encoder only model, they look at the course written and the answers, noticing the relationship between them and understanding context, but they are not good at generating content. An example of Encoder only model would be BERT.

Imagine that we can have someone as well who could create and review the quiz, this is

Service versus Model

Now, let's talk about the difference between a service and a model. A service is a product that is offered by a Cloud Service Provider, and is often a combination of models, data, and other components. A model is the core component of a service, and is often a foundation model, such as an LLM.

Services are often optimized for production use and are often easier to use than models, via a graphical user interface. However, services are not always available for free, and may require a subscription or payment to use, in exchange for leveraging the service owner’s equipment and resources, optimizing expenses and scaling easily.

Models are just the Neural Network, with the parameters, weights, and others. Allowing companies to run locally, however, would need to buy equipment, build structure to scale and buy a license or use an open-source model. A model like LLaMA is available to be used, requiring computational power to run the model.

How to test and iterate with different models to understand performance on Azure

- Find the Foundation Model of interest in the catalog - either proprietary or open source, filtering by task, license, or name. To improve searchability, the models are organized into collections, like Azure OpenAI collection, Hugging Face collection, and more.
- Review the model card, including a detailed description of intended use and training data, code samples and evaluation results on internal evaluations library.
- Compare benchmarks across models and datasets available in the industry to assess which one meets the business scenario, through the Model Benchmarks pane.
- Fine-tune the model on custom training data to improve model performance in a specific workload, leveraging the experimentation and tracking capabilities of

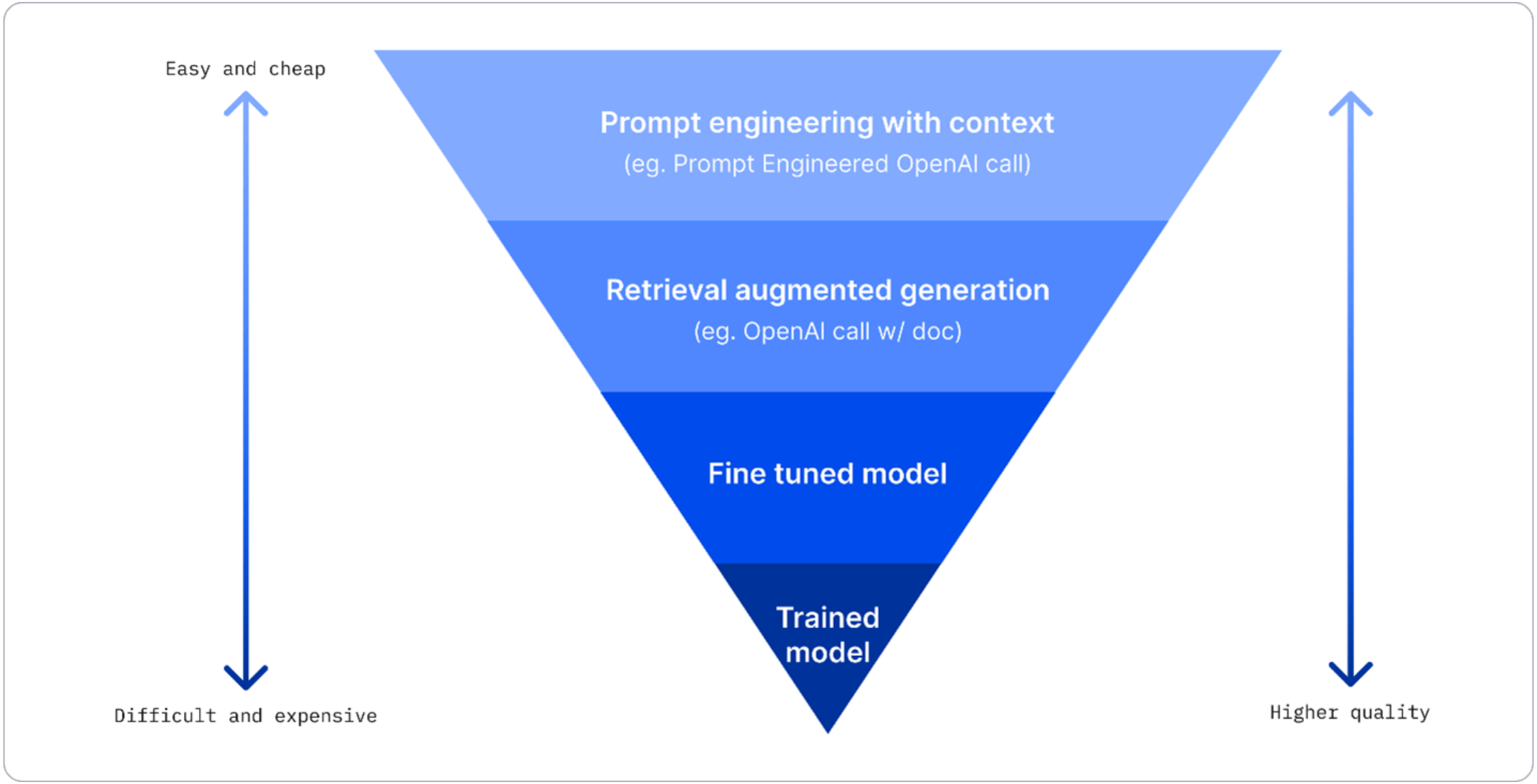
Improving LLM results

We've explored with our startup team different kinds of LLMs and a Cloud Platform (Azure Machine Learning) enabling us to compare different models, evaluate them on test data, improve performance and deploy them on inference endpoints.

But when shall they consider fine-tuning a model rather than using a pre-trained one? Are there other approaches to improve model performance on specific workloads?

There are several approaches a business can use to get the results they need from an LLM. You can select different types of models with different degrees of training when deploying an LLM in production, with different levels of complexity, cost, and quality. Here are some different approaches:

- **Prompt engineering with context**. The idea is to provide enough context when you prompt to ensure you get the responses you need.
- **Retrieval Augmented Generation, RAG**. Your data might exist in a database or web endpoint for example, to ensure this data, or a subset of it, is included at the time of prompting, you can fetch the relevant data and make that part of the user's prompt.
- **Fine-tuned model**. Here, you trained the model further on your own data which leads to the model being more exact and responsive to your needs but might be costly.



Img source: Four Ways that Enterprises Deploy LLMs | Fiddler AI Blog

Prompt Engineering with Context

Pre-trained LLMs work very well on generalized natural language tasks, even by calling them with a short prompt, like a sentence to complete or a question – the so-called "zero-shot" learning.

However, the more the user can frame their query, with a detailed request and examples – the Context – the more accurate and closest to user's expectations the answer will be. In this case, we talk about "one-shot" learning if the prompt includes only one example and "few shot learning" if it includes multiple examples.
Prompt engineering with context is the most cost-effective approach to kick-off with.

Retrieval Augmented Generation (RAG)

LLMs have the limitation that they can use only the data that has been used during their training to generate an answer. This means that they don't know anything about the facts that happened after their training process, and they cannot access non-public information (like company data).
This can be overcome through RAG, a technique that augments prompt with external data in the form of chunks of documents, considering prompt length limits. This is supported by Vector database tools (like Azure Vector Search) that retrieve the useful chunks from varied pre-defined data sources and add them to the prompt Context.

This technique is very helpful when a business doesn't have enough data, enough time, or resources to fine-tune an LLM, but still wishes to improve performance on a specific workload and reduce risks of fabrications, i.e., mystification of reality or harmful content.

Fine-tuned model

Fine-tuning is a process that leverages transfer learning to ‘adapt’ the model to a downstream task or to solve a specific problem. Differently from few-shot learning and RAG, it results in a new model being generated, with updated weights and biases. It requires a set of training examples consisting of a single input (the prompt) and its associated output (the completion).

This would be the preferred approach if:

- **Using fine-tuned models**. A business would like to use fine-tuned less capable models (like embedding models) rather than high performance models, resulting in a more cost effective and fast solution.
- **Considering latency**. Latency is important for a specific use-case, so it's not possible to use very long prompts or the number of examples that should be learned from the model doesn't fit with the prompt length limit.
- **Staying up to date**. A business has a lot of high-quality data and ground truth labels and the resources required to maintain this data up to date over time.

Trained model

Training an LLM from scratch is without a doubt the most difficult and the most complex approach to adopt, requiring massive amounts of data, skilled resources, and appropriate computational power. This option should be considered only in a scenario where a business has a domain-specific use case and a large amount of domain-centric data.