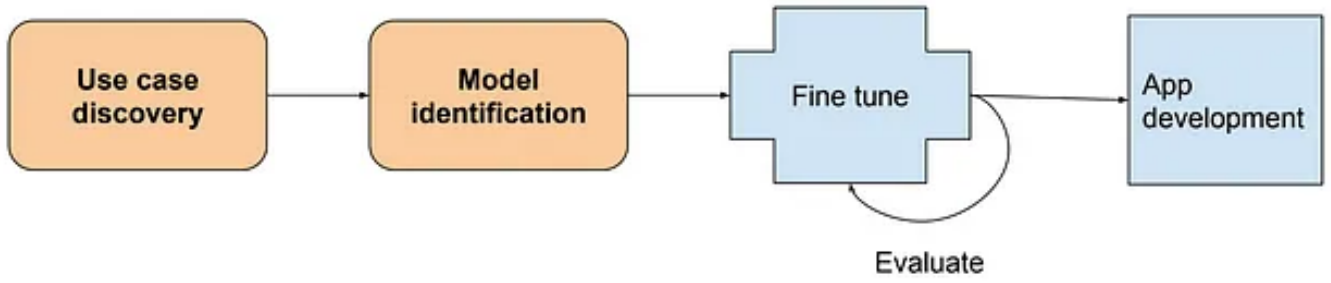
Generative AI project — Part 1

Muthu Arumugam

This article will help you understand how you can get involved and create products using Generative AI models. To get a quick intro to Generative AI, look at my previous articles — Quickies.

For an AI project, these are the following steps equivalent to SDLC.



AI project lifecycle

Use case discovery

You have the option to choose from a variety of tasks from LLMs. You can choose 1 or many for your project from below:

- Essay Writing
- Summarization
- Translation from language to language
- Translation from language to code
- Information retrieval
- Call external APIs

Model identification

The existing model may be sufficient or you have to pre-train with your

dataset. Each model fits for few use cases.

- **Autoencoding** models — BERT/ROBERTA — Sentiment analysis, Named entity recognition, word classification
- **Autoregressive** models — GPT/BLOOM — Text generation
- **Sequence-to-sequence** models — FLAN-T5/BART — Translation, Text summarization, Question/Answers

The model size is also a factor to consider. BERT-L is about 340 million parameters but GPT-3 is about 175 billion parameters. It affects the computing resources needed to train or use them. GPUs play a major role in helping to process them in parallel effectively. For example, you need to store 1 billion parameters for a model, you need 4GB memory (Each parameter is represented in a 32-bit float). We also need additional 20 bytes for each parameter to train. That comes to 80GB of memory for billion parameters.

You can use quantization techniques to save memory by trading off the precision. Instead of a 32-bit float, store the parameters as an integer of 8 bits which gives you 1/4 of the RAM needed. You definitely lose some precision on the output. Your choices are FP32, FP16, **BFLOAT16** (popular) and INT8. If the model size increases, you have the ability to split the tasks into multiple GPUs. There are some strategies available:

- DDP — **Distributed Data Parallel**
- FSDP — **Fully Sharded Data Parallel** (ZeRO)

NVIDIA A-100s are expensive GPUs and the above strategies help you to optimize and use less of those chips to train your models. Sometimes, it's better to choose a smaller model and train accordingly for your smaller use cases.

The measurement used to understand the training time is 1 “petaflop/s-day” = # floating point operations performed at the rate of 1 petaFLOP per second for one day.

There is a paper called “Chinchilla paper” that goes in deep to train LLMs optimally. Also, you have to consider the size of the training data model which is the ideal size of ~20x. If the model uses 70B parameters, you need to feed ~1.4T tokens of a dataset.

You can elect to choose a smaller model and then can train it for a specific field of yours. For example, Bloomberg published a model which is trained for finance-related LLM. See here: [Introducing BloombergGPT, Bloomberg's 50-billion parameter large language model, purpose-built from scratch for finance | Press | Bloomberg LP]

There are so many models developed & trained already. Look at **Hugging Face** to understand with a model card of their own.

- Hugging Face Model hub — https://huggingface.co/models
- Hugging Face Tasks — https://huggingface.co/tasks

We now can see how to choose a model and think about computing resources for training purposes. There are lots of new models that are evolving on a daily basis. Find the one that fits closely what you need rather than plan on training it if possible.

Disclaimer: This is not generated by an AI bot. Also, a lot of these were learned through the DeepLearning.ai course at Coursera. It's a great course. The title image was generated by DALL-E through the Bing chatbot.