Member-only story

A Practical Introduction to LLMs

3 levels of using LLMs in practice

Shaw Talebi

Published in Towards Data Science · 7 min read · Jul 13, 2023

This is the first article in a series on using Large Language Models (LLMs) in practice. Here I will give an introduction to LLMs and present 3 levels of working with them. Future articles will explore practical aspects of LLMs, such as how to use OpenAI's public API, the Hugging Face Transformers Python library, how to fine-tune LLMs, and how to build an LLM from scratch.

Photo by Patrick Tomasso on Unsplash

What is an LLM?

LLM is short for **Large Language Model**, which is a recent innovation in AI and machine learning. This powerful new type of AI went viral in Dec 2022 with the release of ChatGPT.

For those enlightened enough to live outside the world of AI buzz and tech news cycles, **ChatGPT** is a chat interface that ran on an LLM called GPT-3 (now upgraded to either GPT-3.5 or GPT-4 at the time of writing this).

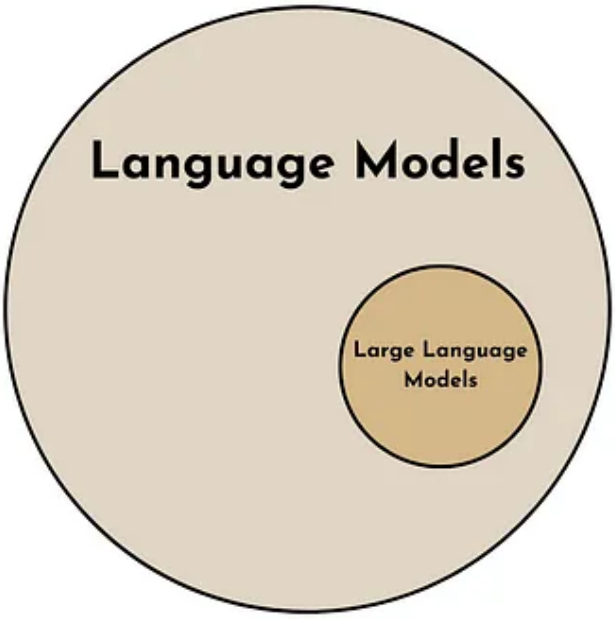
If you've used ChatGPT, it's obvious that this is not your traditional chatbot from AOL Instant Messenger or your credit card's customer care.

This one *feels* different.

What makes an LLM “large”?

When I heard the term “Large Language Model,” one of my first questions was, *how is this different from a “regular” language model?*

A language model is more generic than a large language model. Just like all squares are rectangles but not all rectangles are squares. **All LLMs are language models, but not all language models are LLMs.**



Large Language Models are a special type of Language Model. Image by author.

Okay, so LLMs are a special type of language model, **but what makes them special?**

There are **2 key properties** that distinguish LLMs from other language models. One is quantitative, and the other is qualitative.

1. **Quantitatively**, what distinguishes an LLM is the number of parameters used in the model. Current LLMs have on the order of **10–100 billion parameters** [1].
2. **Qualitatively**, something remarkable happens when a language model becomes "large." It exhibits so-called ***emergent properties*** e.g. zero-shot learning [1]. These are **properties that seem to suddenly appear** when a language model reaches a sufficiently large size.

Zero-shot Learning

The major innovation of GPT-3 (and other LLMs) is that it is capable of **zero-shot learning** in a wide variety of contexts [2]. This means ChatGPT can **perform a task even if it has not been explicitly trained to do it.**

While this might be no big deal to us highly evolved humans, this zero-shot learning ability starkly contrasts the prior machine learning paradigm.

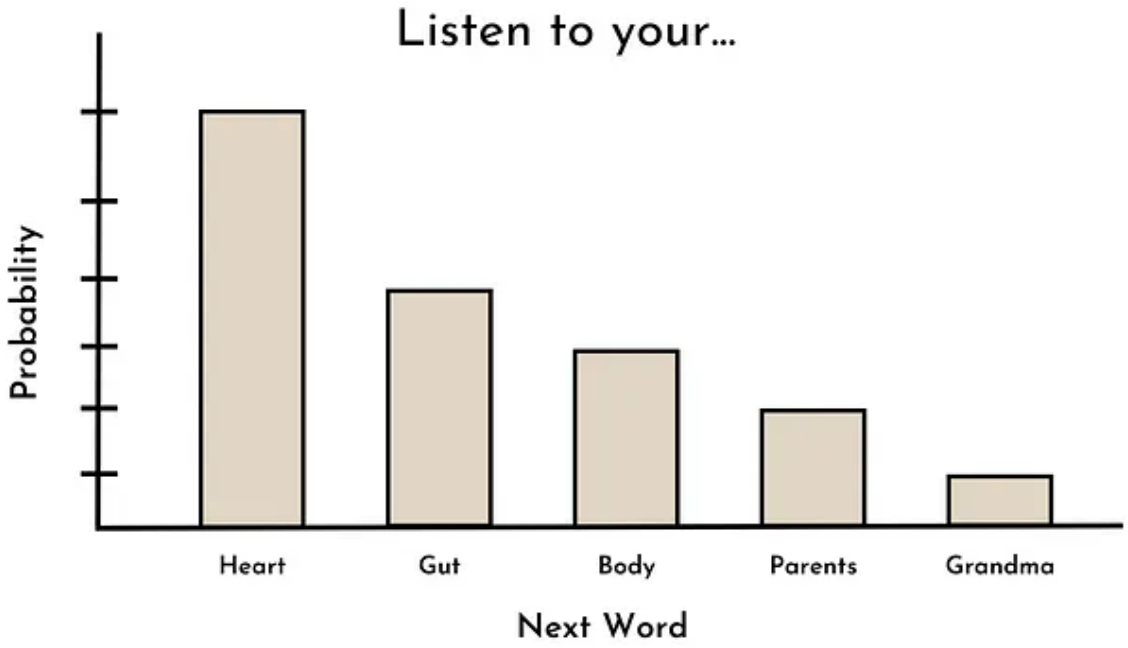
Previously, a model needed to be **explicitly trained on the task it aimed to do** in order to have good performance. This could require anywhere from 1k-1M pre-labeled training examples.

For instance, if you wanted a computer to do language translation, sentiment analysis, and identify grammatical errors. Each of these tasks would require a specialized model trained on a large set of labeled examples. Now, however, **LLMs can do all these things without explicit training.**

How do LLMs work?

The core task used to train most state-of-the-art LLMs is **word prediction.** In other words, given a sequence of words, **what is the probability distribution of the next word**?

For example, given the sequence "Listen to your ____," the most likely next words might be: heart, gut, body, parents, grandma, etc. This might look like the probability distribution shown below.



Toy probability distribution of next word in sequence "Listen to your ___." Image by author.

Interestingly, this is the same way many (non-large) language models have been trained in the past (e.g. GPT-1) [3]. However, for some reason, when language models get beyond a certain size (say ~10B parameters), these (emergent) abilities, such as zero-shot learning, can start to pop up [1].

Although there is no clear answer as to *why* this occurs (only speculations for now), it is clear that LLMs are a powerful technology with countless potential use cases.

3 Levels of Using LLMs

Now we turn to how to use this powerful technology in practice. While there are countless potential LLM use cases, here I categorize them into 3 levels **ordered by required technical knowledge and computational resources.** We start with the most accessible.

Level 1: Prompt Engineering

The first level of using LLMs in practice is **prompt engineering**, which I define as **any use of an LLM out-of-the-box** i.e. not changing any model parameters. While many technically-inclined individuals seem to scoff at the idea of prompt engineering, this is the most accessible way to use LLMs (both technically and economically) in practice.

There are 2 main ways to do prompt engineering: the **Easy Way** and the **Less Easy Way**.

The Easy Way: ChatGPT (or another convenient LLM UI) — The key benefit of this method is convenience. Tools like ChatGPT provide an intuitive, no-cost, and no-code way to use an LLM (it doesn't get much easier than that).

However, convenience often comes at a cost. In this case, there are **2 key drawbacks** to this approach. The **first** is a lack of functionality. For example, ChatGPT does not readily enable users to customize model input parameters (e.g. temperature or max response length), which are values that modulate LLM outputs. **Second**, interactions with the ChatGPT UI cannot be readily automated and thus applied to large-scale use cases.

While these drawbacks may be dealbreakers for some use cases, both can be ameliorated if we take prompt engineering one step further.

The Less Easy Way: Interact with LLM directly — We can overcome some of the drawbacks of ChatGPT by interacting directly with an LLM via programmatic interfaces. This could be via public APIs (e.g. OpenAI's API) or running an LLM locally (using libraries like Transformers).

While this way of doing prompt engineering is less convenient (since it requires programming knowledge and potential API costs), it provides a customizable, flexible, and scalable way to use LLMs in practice. Future articles in this series will discuss paid and cost-free ways to do this type of prompt engineering.

Although prompt engineering (as defined here) can handle most potential LLM applications, relying on a generic model, out-of-the-box may result in sub-optimal performance for specific use cases. For these situations, we can go to the next level of using LLMs.

Level 2: Model Fine-tuning

The second level of using an LLM is **model fine-tuning**, which I'll define as taking an existing LLM and tweaking it for a particular use case by **training at least 1 (internal) model parameter** i.e. weights and biases. For the aficionados out there, this is an example of *transfer learning* i.e. using some part of an existing model to develop another model.

Fine-tuning typically consists of 2 steps. **Step 1**: Obtain a pre-trained LLM. **Step 2**: Update model parameters for a specific task given (typically 1000s of) high-quality labeled examples.

The model parameters define the LLM's internal representation of the input text. Thus, by tweaking these parameters for a particular task, the internal

representations become optimized for the fine-tuning task (or at least that's the idea).

This is a powerful approach to model development because a relatively **small number of examples** and computational resources **can produce exceptional model performance**.

The downside, however, is it requires significantly more technical expertise and computational resources than prompt engineering. In a future article, I will attempt to curb this downside by reviewing fine-tuning techniques and sharing example Python code.

Fine-Tuning Large Language Models (LLMs)

A conceptual overview with example Python code

towardsdatascience.com

While prompt engineering and model fine-tuning can likely handle 99% of LLM applications, there are cases where one must go even further.

Level 3: Build your own LLM

The third and final way to use an LLM in practice is to **build your own**. In terms of model parameters, this is where you **come up with all the model parameters** from scratch.

An LLM is primarily a product of its training data. Thus, for some applications, it may be necessary to curate custom, high-quality text corpora for model training—for example, a medical research corpus for the development of a clinical application.

The biggest upside to this approach is you can **fully customize the LLM for your particular use case**. This is the ultimate flexibility. However, as is often the case, flexibility comes at the cost of convenience.

Since the **key to LLM performance is scale**, building an LLM from scratch requires tremendous computational resources and technical expertise. In other words, this isn't going to be a solo weekend project but a full team working for months, if not years, with a 7–8F budget.

Nevertheless, in a future article in this series, we will explore popular techniques for developing LLMs from scratch.

How to Build an LLM from Scratch

Data Curation, Transformers, Training at Scale, and Model Evaluation

towardsdatascience.com

Conclusion

While there is more than enough hype about LLMs, they are a powerful innovation in AI. Here, I provided a primer on what LLMs are and framed how they can be used in practice. The next article in this series will give a beginner's guide to OpenAI's Python API to help jumpstart your next LLM use case.

👉 **More on LLMs:** OpenAI API | Hugging Face Transformers | Prompt Engineering | Fine-tuning | Build an LLM

Cracking Open the OpenAI (Python) API

A complete beginner-friendly introduction with example code

towardsdatascience.com

Resources

Connect: My website | Book a call | Ask me anything

Socials: YouTube 🎥 | LinkedIn | Twitter

Support: Buy me a coffee ☕

The Data Entrepreneurs

A community for entrepreneurs in the data space. 👉 Join the Discord!

medium.com

[1] Survey of Large Language Models. arXiv:2303.18223 **[cs.CL]**

[2] GPT-3 Paper. arXiv:2005.14165 **[cs.CL]**

[3] Radford, A., & Narasimhan, K. (2018). Improving Language Understanding by Generative Pre-Training. (GPT-1 Paper)