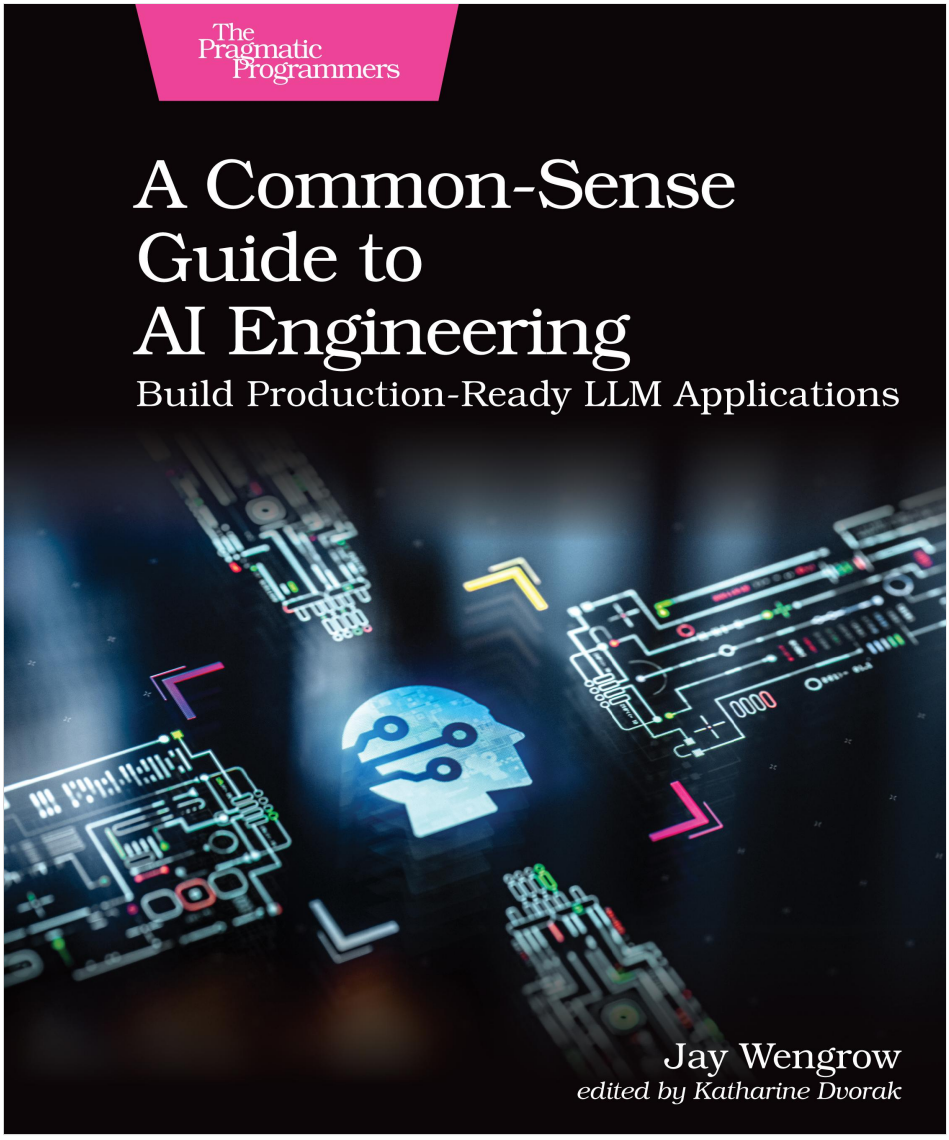


This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Copyright © The Pragmatic Programmers, LLC.

What Is a Large Language Model (LLM)?

To understand what LLMs are and how they work, we first need to zoom out a bit and explore the field of *machine learning*. If you're a software engineer who hasn't yet touched machine learning, that's perfectly fine, but to grasp the idea behind machine learning and thus, LLMs, you'll need to make an important mental shift.

As software developers, we're used to the concept of an *algorithm*—a set of instructions we give a computer to follow, usually in the form of code. The computer then performs each step of the algorithm, and by the time the computer executes the complete program, a meaningful task has been accomplished. Without this set of precise instructions, the computer cannot complete any task.

Now, if we were to write a program that can look at images of animals and identify whether the animal is a dog or a cat, we'd have to develop some pretty complex algorithms. In fact, it's difficult to even know where to start. Do we write code that analyzes the different pixels in the image and tries to discover a certain pattern? If yes, do we look out for pixels that seem like pointy ears and decide that this animal is a cat? Does an oval-shaped set of pink pixels represent the hanging tongue of a dog? This approach can get pretty complex pretty fast.

Fortunately, there's a completely different approach for solving this cat/dog identification problem, and that is *machine learning*. Machine learning is a vast topic, but here's the gist of it as applied to our problem: Instead of writing an algorithm that tells the computer what to do, we instead show the computer many images (say, thousands) of cats and dogs and tell the computer which image contains which animal. This trove of images form what we call *training data*.

Based on this training data, the computer can then classify any *new* image as a cat or dog. It does this by essentially seeing if this new image is more similar to the cat images or dog images from its training data. Instead of telling the computer what to do, it can "do its own thing" by comparing a new image to the training data it's seen before.

Now, it would take a ridiculously long time for the computer to actually compare a new image to each and every image from its training data. So instead, what machine learning does is create a math-based function—called a *model*—based on the training data. That is, the model distills all the cat images into a set of numbers that kind of represent the "average" of all cats.

• Click HERE to purchase this book now. discuss

Similarly, the model distills all the *dog* images into numbers that represent the "average" dog.

Once we have our model set up, when we ask the computer to classify a new image, it converts the image into a set of numbers and sees whether they align more closely to the "cat numbers" or the "dog numbers."

I hope it's clear that I'm explaining machine learning *very* simplistically. There's a lot of complexity (and math) in getting it to work! However, this general perspective is all you need right now.

Dive Deeper: Machine Learning

If you'd like to learn more about machine learning in a hands-on way, I recommend Paolo Perrotta's book, *Programming Machine Learning*,[1] also published by the Pragmatic Bookshelf.

A Statistical Next-Word Predictor

Image classification is just one type of machine learning model. The *large language model* is another. An LLM is built by feeding it training data that consists of *text*, such as books, articles, webpages, and more. With this training data, the computer builds a math-based model that represents *the statistics of what word comes next in a sequence*. To demonstrate what that means, let's play a little game.

Guess the next word in the phrase: "Mary had a little…"

Assuming you're familiar with the classic nursery rhyme, you almost definitely answered "lamb."

Now, it could be that I actually had this sentence in mind: "Mary had a little too much to eat." However, since you may have seen or heard the phrase "Mary had a little" many times and it virtually always ends with "lamb," you predicted that "lamb" was the next word I had in mind.

An LLM does exactly this based on all the training data it's consumed. That is, after reading all the books and webpages, it develops a model to statistically predict the next word of *any* phrase.

Let's say that the LLM encountered the phrase "Mary had a little" in its training data 100 times. Let's also assume that for 97 of those times, the next word was "lamb." However, there were two instances where the word was "notebook" and one instance where the word was "too." Based on this, the

1. https://pragprog.com/titles/pplearn/programming-machine-learning

LLM creates the following table that sets up the odds for the next word in the phrase "Mary had a little":

Phrase	97%	2%	1%
Mary had a little	lamb	notebook	too

And so, if you type "Mary had a little" into an LLM app, the next word the LLM will spit out would be "lamb" since this is statistically the most likely word to come next in that sentence. In effect, an LLM acts like an "autocomplete engine"—very similar to the autocomplete features found on smartphone messaging apps. The way this works isn't based on any fancy algorithm; rather, it's simply based on the statistics derived from LLM's training data.

Based on this, I like to say that an LLM is a "Statistical Next-Word Predictor," or "SNWP" for short. This is a term of my own making, and I'll use and emphasize it countless times throughout this book, since understanding this idea is so pivotal to LLM engineering.

As another example, here's a row of statistics for the much shorter phrase "Hey":

Phrase	27%	18%	14%	11%	9%	8%	7%	6%
Hey	there	I	I'm	what	how	my	your	buddy

Now, although the phrases of "Hey" and "Mary had a little" are pretty short, an LLM can also predict the next word of a long article, or even an entire book!

The industry term for the text we send to the LLM (in order to receive the likely next word) is called the *prompt*. That is, the prompt is the phrase, article, book, or whatever we give to the LLM, and the LLM's job is to predict the next word.

What's incredible is that the LLM can do this for *any possible prompt*, and even for a prompt it's never seen before. After all, the number of possible prompts out there is infinite. You'll see soon how an LLM does this.

Multiple Words, One at a Time

Now, usually LLMs produce more than just one word. For example, if the prompt is "Mary had a little," the LLM may proceed to output the *entire* nursery rhyme, and not only the next word. In truth, the LLM does indeed predict just one new word at a time. However, the LLM will continue to do this *repeatedly* until it decides that it should stop. That is, it keeps generating one new word at a time over and over again.

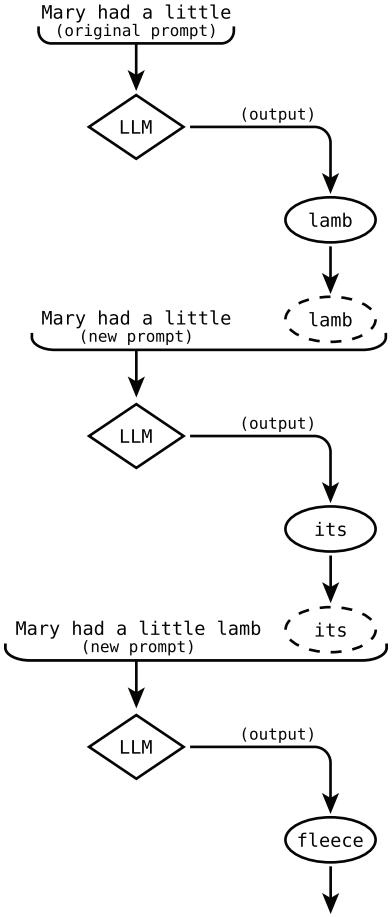
• Click HERE to purchase this book now. discuss

But for this to work, the prompt, too, grows one word at a time. Here's what I mean.

For example, say that our initial prompt to the LLM was `Mary had a little...` Initially, the LLM will output the word `lamb`. However, because the classic nursery rhyme from the training data usually doesn't just stop there, the LLM will go on to generate yet another word.

However, to generate the very next word, the LLM will now work with a new prompt, which is: `Mary had a little lamb`—with the word `lamb` now included at the end. That is, the word which the LLM previously predicted (`lamb`) now *becomes part of the next prompt*.

When the LLM considers this longer prompt of `Mary had a little lamb`, the next word that the LLM will infer will be `its`. And so, the prompts that follow will each be one word longer than the previous prompt, like this:



This system of each prompt including the result of the previous LLM output is known as *autoregressive decoding*. This autoregressive approach is critical, since the alternative would be for the LLM to predict the next word based only on the previous *single word*.

Given `Mary had a little`, if the LLM only looked back at the word `little` without the entire context, it might produce any random word, like `bit`, `boy`, or `girl`. The model needs to see the entire sentence to discern that the next word should specifically be `lamb`.

The LLM will continue to output the rest of the nursery rhyme until it's complete. As to how an LLM knows when to stop chatting, this also has to do with the training data. If the training data concludes the nursery rhyme at a certain point, the LLM will also predict that there are simply no more words that come next. Just as an LLM can predict a next word, the LLM can also predict *that there will be no more words*.

Let's now turn to chatbot apps like ChatGPT. Although ChatGPT is powered by an LLM, it is itself a distinct entity.

Where Chatbots Fit In

I've explained that LLMs are really just SNWPs; they predict the next word of a given prompt. However, this may seem to contradict your own experience using LLM apps. Let's take ChatGPT (or Claude or Gemini, or whatever chatbot you prefer). ChatGPT is itself not an LLM, but an app built *on top* of an LLM, just like the apps you'll learn to build with this book.

If you've used a product like ChatGPT, you'll know that you do *not* typically use it as an autocomplete engine. That is, you aren't typing in prompts like "Mary had a little" and expect the LLM to complete the sentence. Rather, it's a *chat*bot, meaning you typically carry on a *dialogue* with it.

For example, when you start a new chat, you'll see a screen that says something like "How can I help you today?" You'll enter something like "How do I give my car an oil change?" The LLM will then proceed to give you instructions on how to change your car's oil. This doesn't really seem like autocomplete, does it?

But here's the thing: a chatbot's conversation with you is actually *powered by autocomplete*, and here's how. The prompt being autocompleted is not simply your query "How do I give my car an oil change?" Rather, the prompt being completed is *the entire conversation* starting with the LLM's question, "How can I help you today?"

• Click HERE to purchase this book now. discuss

In other words, the prompt is something along the lines of this:

LLM Assistant: *How can I help you today?*

User: *How do I give my car an oil change?*

Now, if I sent you this *entire dialogue* and asked you to predict the next word, wouldn't you predict that it would be the beginning of an explanation of how to change a car's oil?

And so, that's exactly what ChatGPT does too. The fact that ChatGPT spits out a tutorial on how to do an oil change isn't because the LLM understands that you're asking it a question. Rather, ChatGPT sends the entire dialogue as a prompt to the LLM, and the LLM predicts what words would likely come next. In the case of a dialogue where a user asks an AI assistant how to do an oil change, the words to likely come next would indeed be an oil change how-to.

Now, let me explain why having a high-level understanding of how LLMs and chatbots work is so important.

• Click HERE to purchase this book now. discuss