

# 1

## Overview of Large Language Models

In 2017, a team at Google Brain introduced an advanced artificial intelligence (AI) deep learning architecture called the Transformer. Since then, the Transformer has become the standard for tackling various natural language processing (NLP) tasks in academia and industry. It is likely that you have interacted with models built on top of the Transformer architecture in recent years without even realizing it. Google, for example, experimented with using the Bidirectional Encoder Representations from Transformer (BERT), an LLM the company created, to enhance its search engine by better understanding users' search queries. In more recent years, Google started to use Gemini, another LLM it created, to overhaul its search experience altogether. The Generative Pre-trained Transformer (GPT) family of models from OpenAI have also received attention for their ability to generate human-like text and images.

---

### Note

The code for this book is always free and up-to-date on our GitHub repository at [github.com/sinanuozdemir/quick-start-guide-to-llms](https://github.com/sinanuozdemir/quick-start-guide-to-llms).

---

These Transformers power applications such as GitHub's Copilot (developed by OpenAI in collaboration with Microsoft), which can convert comments and snippets of code into fully functioning source code that can even call upon other large language models (LLMs) (as in [Listing 1.1](#)) to perform NLP tasks.

**Listing 1.1 Using GitHub's Copilot LLM to get an output from Meta's BART LLM**

[Click here to view code image](#)

```
from transformers import pipeline

def classify_text(email):
    """
    Use Facebook's BART model to classify an email into "spam" or "not spam"

    Args:
    email (str): The email to classify
    Returns:
    str: The classification of the email
    """

    # COPILOT START. EVERYTHING BEFORE THIS COMMENT WAS INPUT TO COPILOT
    classifier = pipeline(
        'zero-shot-classification', model='facebook/bart-large-mnli')
    labels = ['spam', 'not spam']
    hypothesis_template = 'This email is {}.'

    results = classifier(
        email, labels, hypothesis_template=hypothesis_template)

    return results['labels'][0]
# COPILOT END
```

---

In [Listing 1.1](#), I asked Copilot to take in a Python function definition and some comments I wrote and complete that function to my specifications. There's no cherry-picking here, just a fully working Python function that I can call like this:

[Click here to view code image](#)

```
classify_text('hi I am spam') # spam
```

It appears we are surrounded by LLMs, but just what are they doing under the hood? Let's find out!

## What Are Large Language Models?

**Large language models (LLMs)** are AI models that are usually (but not necessarily) derived from the Transformer architecture and are designed to *parse* and *generate* language, code, and much more. These models are

trained on vast amounts of text data, allowing them to capture the complexities and nuances of human language. LLMs can perform a wide range of language-related tasks, from simple text classification to text generation, with high accuracy, fluency, and style.

In the healthcare industry, LLMs are being used for electronic medical record (EMR) processing, clinical trial matching, and drug discovery. In finance, they are being utilized for fraud detection, sentiment analysis of financial news, and even trading strategies. LLMs are also used for customer service automation via chatbots and virtual assistants. Owing to their versatility and highly performant natures, Transformer-based LLMs are becoming an increasingly valuable asset in a variety of industries and applications.

---

#### Note

I will use the term *understand* a fair amount in this text. In this context, I am usually referring to “natural language understanding” (NLU)—a research branch of NLP that focuses on developing algorithms and models that can accurately interpret human language. As we will see, NLU models excel at tasks such as classification, sentiment analysis, and named entity recognition. However, it is important to note that while these models can perform complex language tasks, they do not possess true understanding in the same way that humans do.

---

The success of LLMs and Transformers is due to the combination of several ideas. Most of these ideas had been around for years but were also being actively researched around the same time. Mechanisms such as attention, transfer learning, and scaling up neural networks, which provide the scaffolding for Transformers, were seeing breakthroughs right around the same time. **Figure 1.1** outlines some of the biggest advancements in NLP in the last few decades, all leading up to the invention of the Transformer.

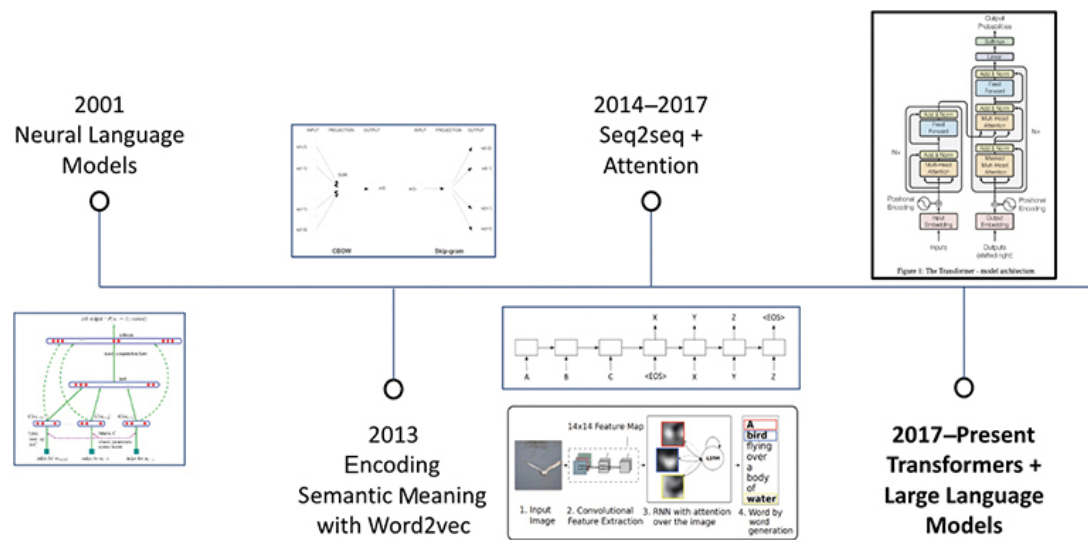


Figure 1.1 A brief history of modern NLP highlights the use of deep learning to tackle language modeling, advancements in large-scale semantic token embeddings (Word2vec), sequence-to-sequence models with attention (something we will see in more depth later in this chapter), and finally the Transformer in 2017.

The Transformer architecture itself is quite impressive. It can be highly parallelized and scaled in ways that previous state-of-the-art NLP models could not be, allowing it to scale to much larger datasets and training times than was possible with previous NLP models. The Transformer uses a special kind of attention calculation called **self-attention** to allow each word in a sequence to “attend to” (look to for context) all other words in the sequence, enabling it to capture long-range dependencies and contextual relationships between words. Of course, no architecture is perfect. Transformers are still limited to an input context window, which represents the maximum length of text they can process at any given moment.

Since the advent of the Transformer architecture in 2017, the ecosystem around using and deploying Transformers has exploded. The aptly named “Transformers” library and its supporting packages have enabled practitioners to use, train, and share models, greatly accelerating this model’s adoption, to the point that it is now being used by thousands of organizations (and counting). Popular LLM repositories such as Hugging Face have popped up, providing access to powerful open-source models to the masses. In short, using and productionizing a Transformer has never been easier.

That's where this book comes in.

My goal is to guide you on how to use, train, and optimize all kinds of LLMs for practical applications while giving you just enough insight into the inner workings of the model to know how to make optimal decisions about model choice, data format, fine-tuning parameters, and so much more.

My aim is to make use of Transformers accessible for software developers, data scientists, analysts, and hobbyists alike. To do that, we should start on a level playing field and learn a bit more about LLMs.

## Definition of LLMs

To back up only slightly, we should talk first about the specific NLP task that LLMs and Transformers are being used to solve, which provides the foundation layer for their ability to solve a multitude of tasks. **Language modeling** is a subfield of NLP that involves the creation of statistical/deep learning models for predicting the likelihood of a sequence of tokens in a specified **vocabulary** (a limited and known set of tokens). There are generally two kinds of language modeling tasks out there: autoencoding tasks and autoregressive tasks ([Figure 1.2](#)).

If you don't \_\_\_\_ at the sign, you will get a ticket.



**Autoencoding** language models ask a model to fill in missing words from any portion of a phrase from a known vocabulary



**Autoregressive** language models ask a model to generate the next most likely token of a given phrase from a known vocabulary

Figure 1.2 Both the autoencoding and autoregressive language modeling tasks involve filling in a missing token, but only the autoencoding task allows for context to be seen on both sides of the missing token.

---

### Note

A **token** is the smallest unit of semantic meaning, which is created by breaking down a sentence or piece of text into smaller units; it is the basic input for an LLM. Tokens can be words but also can be “sub-words,” as we will see in more depth throughout this book. Some readers may be familiar with the term “ $n$ -gram,” which refers to a sequence of  $n$  consecutive tokens.

---

**Autoregressive** language models are trained to predict the next token in a sentence, based on only the previous tokens in the phrase. These models correspond to the decoder part of the Transformer model, with a mask being applied to the full sentence so that the attention heads can see only the tokens that came before. Autoregressive models are ideal for text generation. A good example of this type of model is GPT.

**Autoencoding** language models are trained to reconstruct the original sentence from a corrupted version of the input. These models correspond to the encoder part of the Transformer model and have access to the full input without any mask. Autoencoding models create a bidirectional representation of the whole sentence. They can be fine-tuned for a variety of tasks such as text generation, but their main application is sentence classification or token classification. A typical example of this type of model is BERT.

To summarize, LLMs are language models that may be either autoregressive, autoencoding, or a combination of the two. Modern LLMs are usually based on the Transformer architecture (which we will use in this book), but can also be based on another architecture. The defining features of LLMs are their large size and large training datasets, which enable them to perform complex language tasks, such as text generation and classification, with high accuracy and with little to no fine-tuning.

For now, let's look at some of the popular LLMs we'll be using throughout this book.

## Popular Modern LLMs

BERT, GPT, T5, and Llama are four popular LLMs developed by Google, OpenAI, Google, and Meta, respectively. These models differ quite dramatically in terms of their architecture, even though they all share the Transformer as a common ancestor. Other widely used variants of LLMs in the Transformer family include RoBERTa, BART (which we saw earlier performing some text classification in [Listing 1.1](#)), and ELECTRA.

### BERT

BERT ([Figure 1.3](#)) is an autoencoding model that uses attention to build a bidirectional representation of a sentence. This approach makes it ideal for sentence classification and token classification tasks.

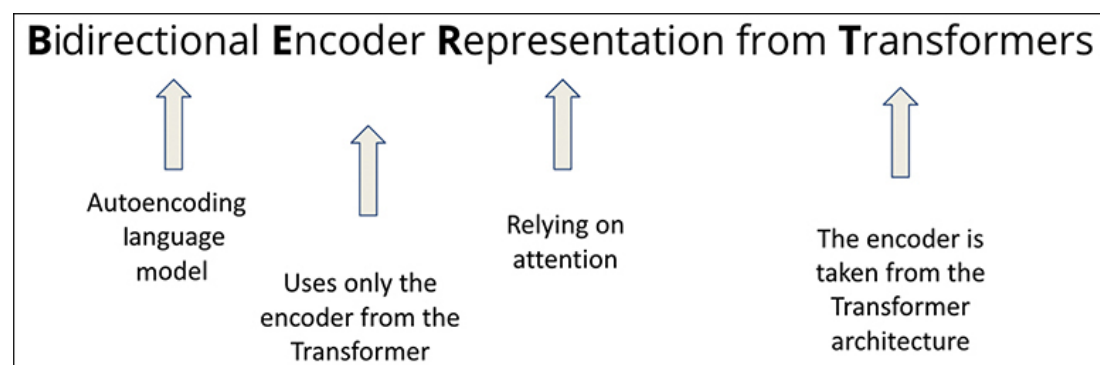


Figure 1.3 BERT was one of the first LLMs and continues to be popular for many NLP tasks that involve fast processing of large amounts of text.

BERT uses the encoder of the Transformer and ignores the decoder to become exceedingly good at processing/understanding massive amounts of text very quickly relative to other, slower LLMs that focus on generating text one token at a time. BERT-derived architectures, therefore, are best for working with and analyzing large corpora quickly when we don't need to write free-text.

BERT itself doesn't classify text or summarize documents, but it is often used as a pre-trained model for downstream NLP tasks. BERT has become a widely used and highly regarded LLM in the NLP community, paving the way for the development of even more advanced language models.

## The GPT Family and ChatGPT

GPT ([Figure 1.4](#)), in contrast to BERT, is an autoregressive model that uses attention to predict the next token in a sequence based on the previous tokens. The GPT family of models (which include ChatGPT and GPT-4) is primarily used for text generation and has been known for its ability to generate natural-sounding, human-like text.

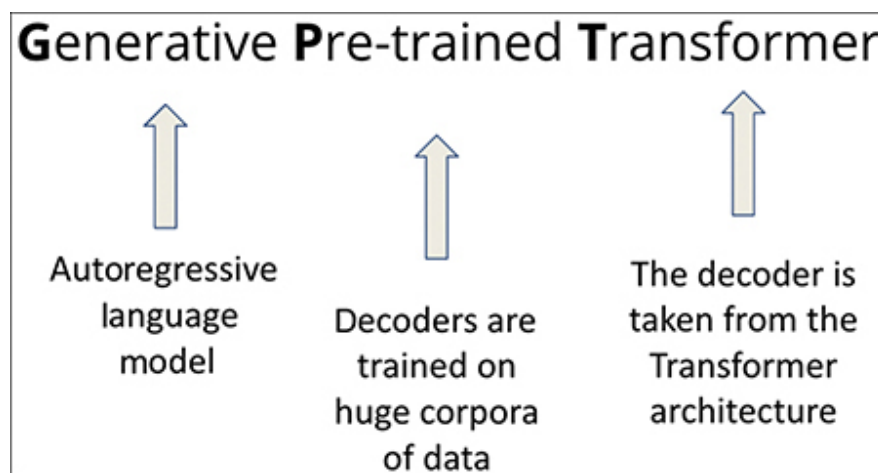


Figure 1.4 The GPT family of models excels at generating free-text aligned with the user's intent.

GPT relies on the decoder portion of the Transformer and ignores the encoder, so it is exceptionally good at generating text one token at a time. GPT-based models are best for generating text given a rather large context window. They can also be used to process and understand text, as we will see later in this book. GPT-derived architectures are ideal for applications that require the ability to freely write text.

## T5

T5 is a pure encoder/decoder Transformer model that was designed to perform several NLP tasks, ranging from text classification to text sum-



marization and generation, right off the shelf. It is one of the first popular models to be able to boast of such a feat, in fact. Before T5, LLMs like BERT and GPT-2 generally had to be fine-tuned using labeled data before they could be relied on to perform such specific tasks.

T5 uses both the encoder and the decoder of the Transformer, so it is highly versatile in both processing and generating text. T5-based models can perform a wide range of NLP tasks, from text classification to text generation, due to their ability to build representations of the input text using the encoder and generate text using the decoder (**Figure 1.5**). T5-derived architectures are ideal for applications that “require both the ability to process and understand text and the ability to generate text freely.”

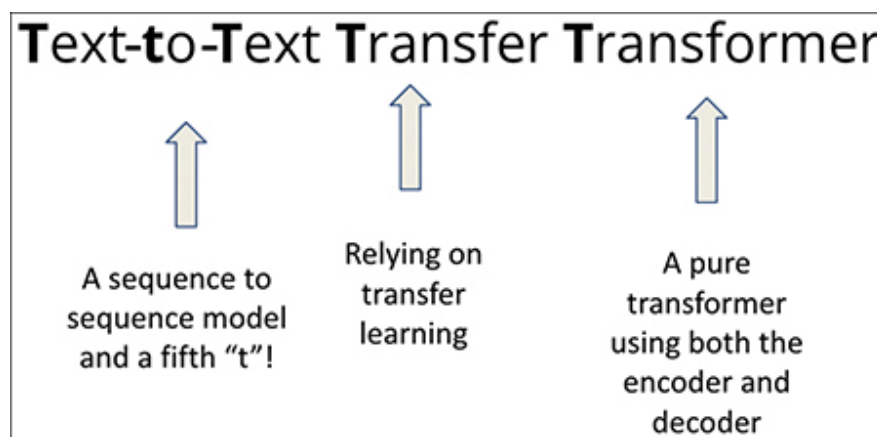


Figure 1.5 T5 was one of the first LLMs to show promise in solving multiple tasks at once without any fine-tuning.

T5’s ability to perform multiple tasks with no fine-tuning spurred the development of other versatile LLMs that can perform multiple tasks with efficiency and accuracy with little or no fine-tuning. GPT-3, released around the same time as T5, also boasted this ability but was closed source and under OpenAI’s control.

More modern open-source LLMs like Llama (seen in **Figure 1.6**) are popping up seemingly by the day and represent a wonderful and massive shift toward a more open and transparent AI community. This shift is not without speedbumps, however. Even Llama—considered one of the most powerful open-source family of autoregressive models—is not 100%

open. To download the parameter weights, you must agree to a relatively strict license, and we do not have access to either the training data or the code used to make the model.

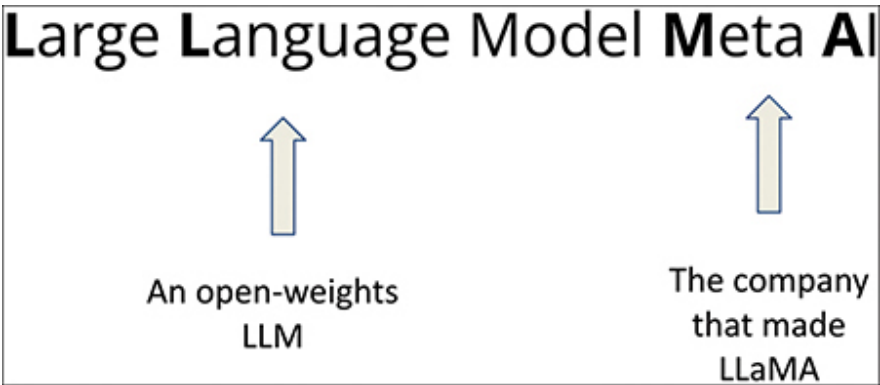


Figure 1.6 The Llama family of models is considered one of the more powerful (mostly) open-weights (fully open-source models would include the training data as well as the training code) families of LLMs, trained on trillions of tokens and ready to be fine-tuned for specific tasks.

Nearly all LLMs are highly versatile and are used for various NLP tasks, such as text classification, text generation, machine translation, and sentiment analysis, among others. These LLMs, along with flavors (variants) of them, will be the main focus of this book and our applications.

**Table 1.1** shows the disk size, memory usage, number of **parameters**—the internal numbers that make up the matrices of the deep learning architecture itself, and the approximate size of the pre-training data for several popular LLMs. Note that these sizes are approximate and may vary depending on the specific implementation and hardware used.

Table 1.1 Comparison of a Sample of Large Language Models

LLM	Disk Size (~GB)	Memory Usage (~GB)	Parameters (~millions)	Training Data Size (~GB)
-----	-----------------	--------------------	------------------------	--------------------------

BERT-Large	1.3	3.3	340	20
------------	-----	-----	-----	----

LLM	Disk Size (~GB)	Memory Usage (~GB)	Parameters (~millions)	Training Data Size (~GB)
GPT-2 117M	0.5	1.5	117	40
GPT-2 1.5B	6	16	1500	40
GPT-3 175B	700	2000	175,000	570
T5-11B	45	40	11,000	750
RoBERTa- Large	1.5	3.5	355	160
ELECTRA- Large	1.3	3.3	335	20

But size isn't everything. Let's look at some of the key characteristics of LLMs and then dive into how they learn to read and write.

## Key Characteristics of LLMs

The original Transformer architecture, as devised in 2017, was a **sequence-to-sequence model**, which means it had two main components:

- An **encoder**, which is tasked with taking in raw text, splitting it up into its core components (more on this later), converting those components into vectors (similar to the Word2vec process), and using attention to *understand* the context of the text
- A **decoder**, which excels at *generating* text by using a modified type of attention to predict the best next token that would fit best in context

As shown in [Figure 1.7](#), the Transformer has many other subcomponents (which we won't get into) that promote faster training, generalizability, and better performance. Today's LLMs are, for the most part, variants of the original Transformer. Models like BERT and GPT dissect the

Transformer into only an encoder and a decoder (respectively) so as to build models that excel in understanding and generating (also respectively).

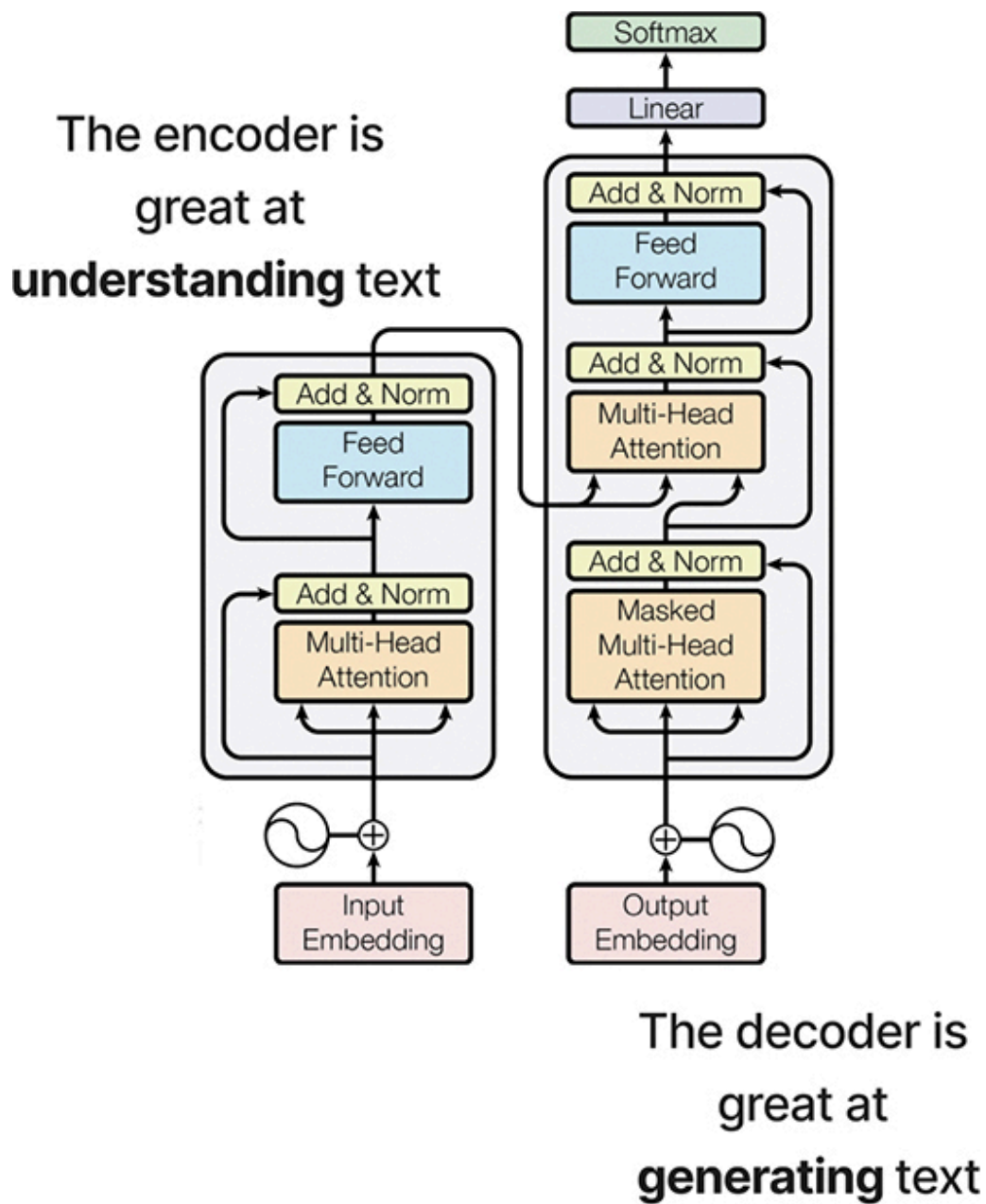


Figure 1.7 The original Transformer has two main components: an encoder (on the left), which is great at understanding text, and a decoder (on the right), which is great at generating text. Putting them together makes the entire model a “sequence-to-sequence” model.

As mentioned earlier, in general, LLMs can be categorized into three main buckets:

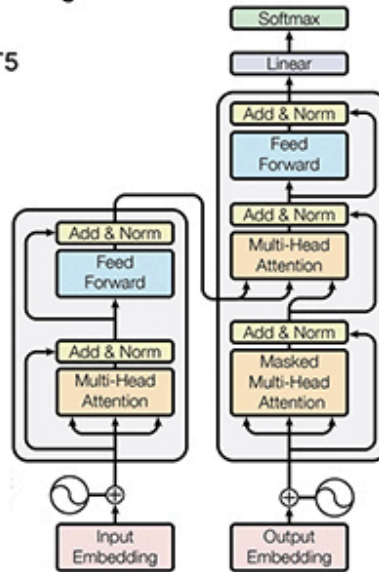
- **Autoregressive models**, such as GPT, which predict the next token in a sentence based on the previous tokens. These LLMs are effective at generating coherent free-text following a given context.
- **Autoencoding models**, such as BERT, which build a bidirectional representation of a sentence by masking some of the input tokens and trying to predict them from the remaining ones. These LLMs are adept at capturing contextual relationships between tokens quickly and at scale, which makes them great candidates for text classification tasks, for example.
- **Combinations** of autoregressive and autoencoding models, such as T5, which can use the encoder and decoder to be more versatile and flexible in generating text. Such combination models can generate more diverse and creative text in different contexts compared to pure decoder-based autoregressive models due to their ability to capture additional context using the encoder.

**Figure 1.8** shows the breakdown of the key characteristics of LLMs based on these three buckets.

## Original Sequence to Sequence Transformer

- Can be trained on and perform auto-encoding and auto-regressive language modeling tasks

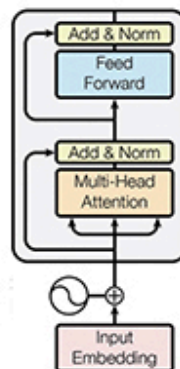
E.g., T5



## Encoder only Models

- Trained on and perform auto-encoding language modeling tasks
- These models excel at **understanding** tasks

E.g., BERT Family



## Decoder only Models

- Trained on and perform auto-regressive language modeling tasks
- These models excel at **generating** tasks

E.g., GPT Family

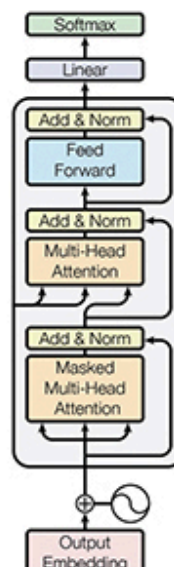


Figure 1.8 A breakdown of the key characteristics of LLM archetypes based on how they are derived from the original Transformer architecture.

## More Context, Please

No matter how the LLM is constructed and which parts of the Transformer it is using, they all care about context ([Figure 1.9](#)). The goal is to understand each token as it relates to the other tokens in the input text. Since the introduction of Word2vec around 2013, NLP practitioners and researchers have been curious about the best ways of combining semantic meaning (basically, word definitions) and context (with the surrounding tokens) to create the most meaningful token embeddings possible. The Transformer relies on the attention calculation to make this combination a reality.

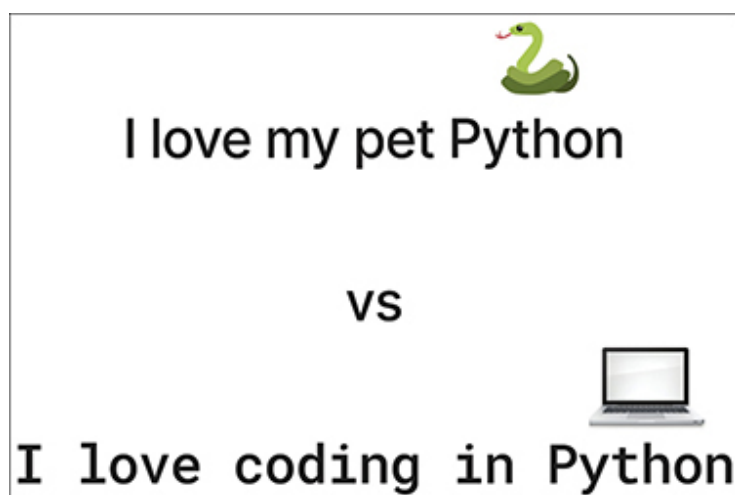


Figure 1.9 LLMs are great at understanding context. The word “Python” can have different meanings depending on the context. We could be talking about a snake or a pretty cool coding language. Images: Arizzona Design/Shutterstock (snake); RASudio/Shutterstock (laptop)

Choosing what kind of Transformer you want isn't enough. Just choosing the encoder doesn't mean your Transformer magically becomes good at understanding text. Let's look at how these LLMs actually learn to read and write.

## How LLMs Work

How an LLM is pre-trained and fine-tuned makes all the difference between an okay-performing model and a state-of-the-art, highly accurate LLM. We'll need to take a quick look into how LLMs are pre-trained to understand what they are good at, what they are bad at, and whether we would need to update them with our own custom data.

### Pre-training

Every LLM on the market has been **pre-trained** on a large corpus of text data and on specific language modeling-related tasks. During pre-training, the LLM tries to learn and understand general language and relationships between words. Every LLM is trained on different corpora and on different tasks.

BERT, for example, was originally pre-trained on two publicly available text corpora ([Figure 1.10](#)):

- **English Wikipedia:** a collection of articles from the English version of Wikipedia, a free online encyclopedia. It contains a range of topics and writing styles, making it a diverse and representative sample of English language text (at the time, 2.5 billion words).
- **The BookCorpus:** a large collection of fiction and nonfiction books. It was created by scraping book text from the web and includes a range of genres, from romance and mystery to science fiction and history. The books in the corpus were selected to have a minimum length of 2000 words and to be written in English by authors with verified identities (approximately 800 million words in total).



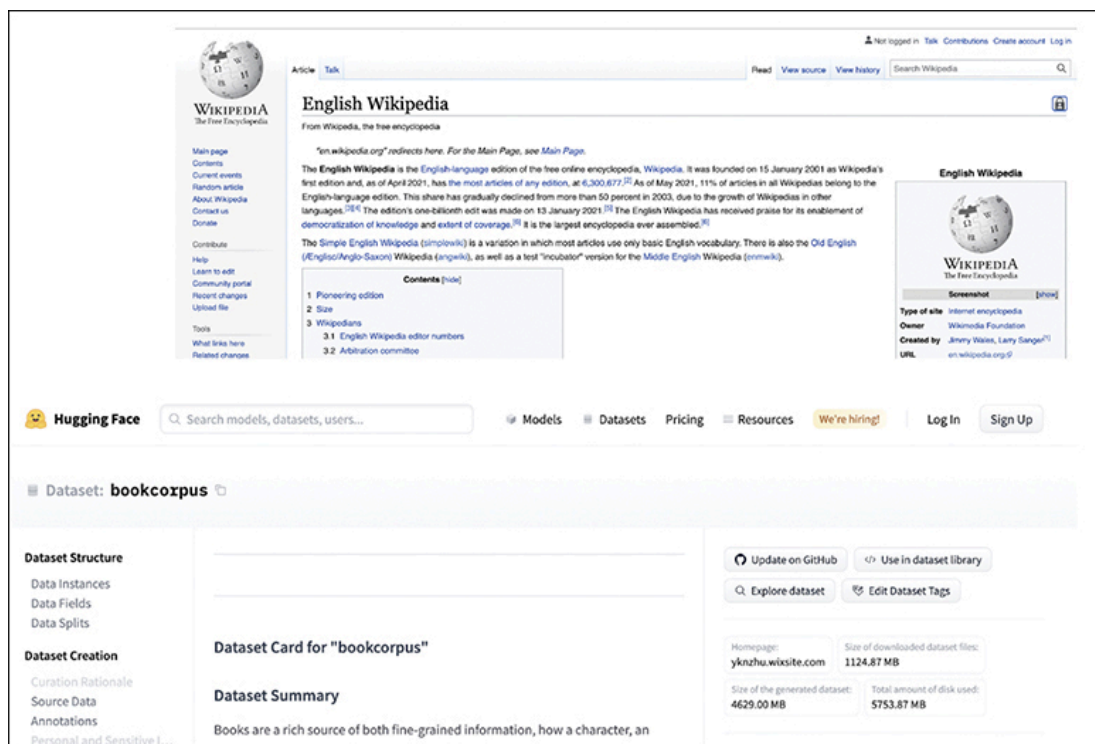


Figure 1.10 BERT was originally pre-trained on English Wikipedia and the BookCorpus. More modern LLMs are trained on datasets thousands of times larger.

BERT was also pre-trained on two specific language modeling tasks

(**Figure 1.11**):

- Masked Language Modeling (MLM) task (autoencoding task): helps BERT recognize token interactions within a single sentence.
- Next Sentence Prediction (NSP) task: helps BERT understand how tokens interact with each other between sentences.

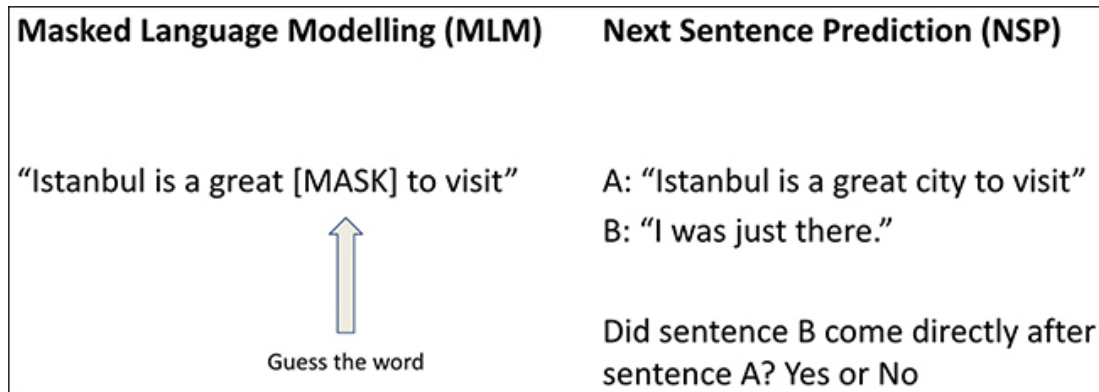


Figure 1.11 BERT was pre-trained on two tasks: the autoencoding language modeling task (referred to as the “masked language modeling” task) to teach it individual word embeddings and the “next sentence prediction” task to help it learn to embed entire sequences of text.

Pre-training on these corpora allowed BERT (mainly via the self-attention mechanism) to learn a rich set of language features and contextual relationships. The use of large, diverse corpora like these has become a common practice in NLP research, as it has been shown to improve the performance of models on downstream tasks.

---

#### Note

The pre-training process for an LLM can evolve over time as researchers find better ways of training LLMs and phase out methods that don’t help as much. For example, within a year of the original Google BERT release that used the NSP pre-training task, a BERT variant called RoBERTa (yes, most of these LLM names will be fun) by Facebook AI was shown to not require the NSP task to match and even beat the original BERT model’s performance in several areas.

---

BERT, as we now know, is an autoencoding model, so its pre-training will be different than how Llama-3, for example, is pre-trained. Instead of MLM and NSP, autoregressive models are pre-trained simply on the autoregressive language modeling task over a predefined corpus of data. Put another way, pre-training models like Llama-3 just means that they read

vast amounts of unstructured text mostly from the internet and are trained to emulate the language as closely as possible.

Depending on which LLM archetype you decide to use, it will likely be pre-trained differently from its counterparts. This is what sets LLMs apart from each other. For example, Google might decide to train the models it creates on data the company has easy access to (from Google searches), whereas Meta might use data from Facebook Messenger, WhatsApp, Instagram, or another of its own apps. Some LLMs are trained on proprietary data sources, including OpenAI's GPT family of models, to give their parent companies an edge over their competitors.

We won't revisit the idea of pre-training often in this book because it's not exactly the "quick" part of a "quick start guide." Nevertheless, it can be worth knowing how these models were pre-trained because this pre-training enables us to apply transfer learning, which lets us achieve the state-of-the-art results we want—which is a big deal!

## **Transfer Learning**

Transfer learning is a technique used in machine learning to leverage the knowledge gained from one task to improve performance on another related task. Transfer learning for LLMs involves taking an LLM that has been pre-trained on one corpus of text data and then fine-tuning it for a specific "downstream" task, such as text classification or text generation, by updating the model's parameters with task-specific data.

The idea behind transfer learning is that the pre-trained model has already learned a lot of information about the language and relationships between words, and this information can be used as a starting point to improve performance on a new task. Transfer learning allows LLMs to be fine-tuned for specific tasks with much smaller amounts of task-specific data than would be required if the model were trained from scratch. This greatly reduces the amount of time and resources needed to train LLMs.

**Figure 1.12** provides a visual representation of this relationship.

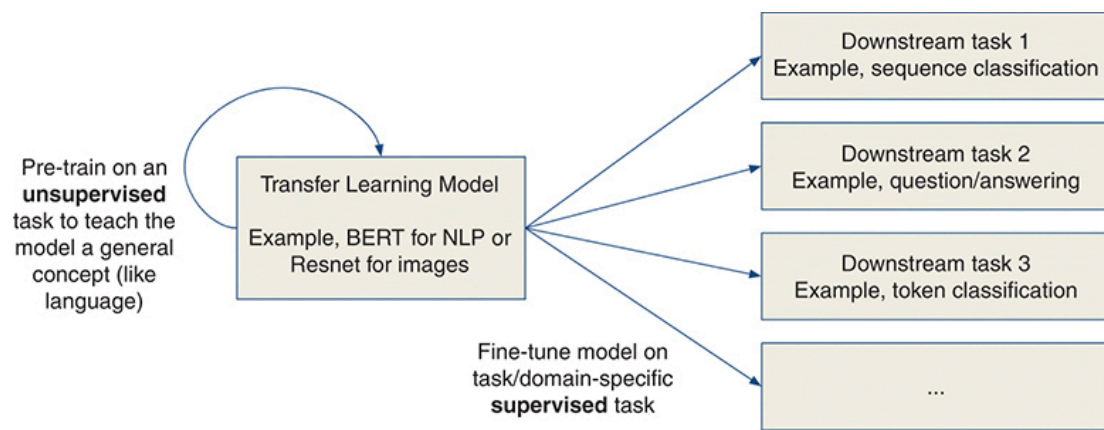


Figure 1.12 The general transfer learning loop involves pre-training a model on a generic dataset on some generic self-supervised task and then fine-tuning the model on a task-specific dataset.

## Fine-Tuning

Once an LLM has been pre-trained, it can be fine-tuned for specific tasks. Fine-tuning involves training the LLM on a smaller, task-specific dataset to adjust its parameters for the specific task at hand. This allows the LLM to leverage its pre-trained knowledge of the language to improve its accuracy for the specific task. Fine-tuning has been shown to drastically improve performance on domain-specific and task-specific tasks and lets LLMs adapt quickly to a wide variety of NLP applications.

**Figure 1.13** shows the basic fine-tuning loop that we will use for our models in later chapters. Whether they are open-source or closed-source, the loop is more or less the same:

1. We define the model we want to fine-tune as well as any fine-tuning parameters (e.g., learning rate).
2. We aggregate some training data (the format and other characteristics depend on the model we are updating).
3. We compute losses (a measure of error) and gradients (information about how to change the model to minimize error).
4. We update the model through backpropagation—a mechanism to update model parameters to minimize errors.

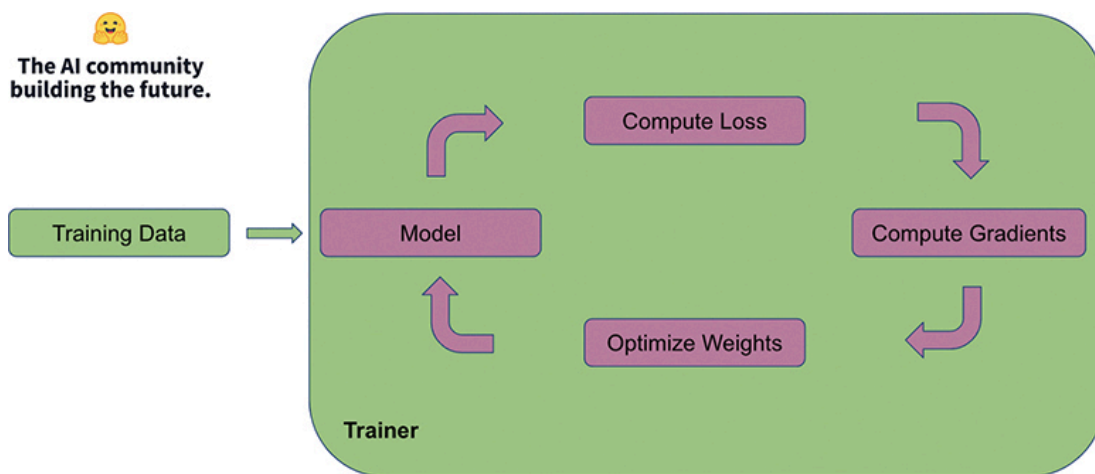


Figure 1.13 The Transformers package from Hugging Face provides a neat and clean interface for training and fine-tuning LLMs.

If some of that went over your head, not to worry: We will rely on pre-built tools from Hugging Face’s Transformers package ([Figure 1.13](#)) and OpenAI’s Fine-Tuning API to abstract away a lot of this so we can really focus on our data and our models.

---

#### Note

You will not need a Hugging Face account or API key to follow along and use most of the code in this book, apart from the very specific advanced exercises where I will call it out.

---

#### Attention

The title of the original paper that introduced the Transformer was “Attention Is All You Need.” **Attention** is a mechanism used in deep learning models (not just Transformers) that assigns different weights to different parts of the input, allowing the model to prioritize and emphasize the most important information while performing tasks like translation or summarization. Essentially, attention allows a model to “focus” on different parts of the input dynamically, leading to improved performance and more accurate results. Before the popularization of attention, most neural networks processed all inputs equally and the models relied on a fixed

representation of the input to make predictions. Modern LLMs that rely on attention can dynamically focus on different parts of input sequences, allowing them to weigh the importance of each part in making predictions.

To recap, LLMs are pre-trained on large corpora and sometimes fine-tuned on smaller datasets for specific tasks. Recall that one of the factors behind the Transformer's effectiveness as a language model is that it is highly parallelizable, allowing for faster training and efficient processing of text. What really sets the Transformer apart from other deep learning architectures is its ability to capture long-range dependencies and relationships between tokens using attention. In other words, attention is a crucial component of Transformer-based LLMs, and it enables them to effectively retain information between training loops and tasks (i.e., transfer learning), while being able to process lengthy swatches of text with ease.

Attention is considered the aspect most responsible for helping LLMs learn (or at least recognize) internal world models and human-identifiable rules. A Stanford University study conducted in 2019 showed that certain attention calculations in BERT corresponded to linguistic notions of syntax and grammar rules. For example, the researchers noticed that BERT was able to notice direct objects of verbs, determiners of nouns, and objects of prepositions with remarkably high accuracy from only its pre-training. These relationships are presented visually in [\*\*Figure 1.14\*\*](#).



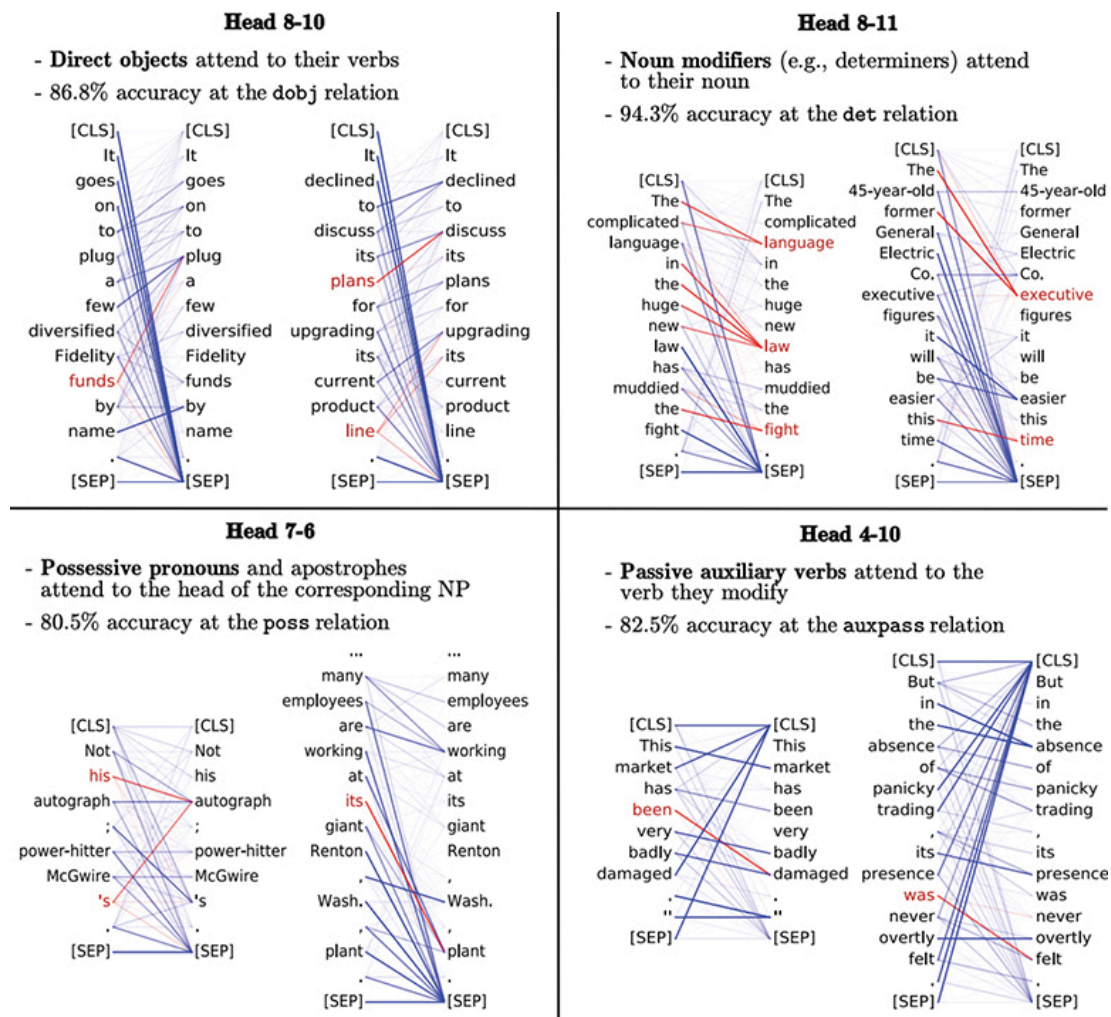


Figure 1.14 Research has probed into LLMs and revealed that they seem to be recognizing grammatical rules even when they were never explicitly told these rules.

Other research has explored which other kinds of “rules” LLMs are able to learn simply by pre-training and fine-tuning. One example is a series of experiments led by researchers at Harvard University that explored an LLM’s ability to learn a set of rules for a synthetic task like the game of Othello (**Figure 1.15**). They found evidence that an LLM was able to understand the rules of the game simply by training on historical move data.

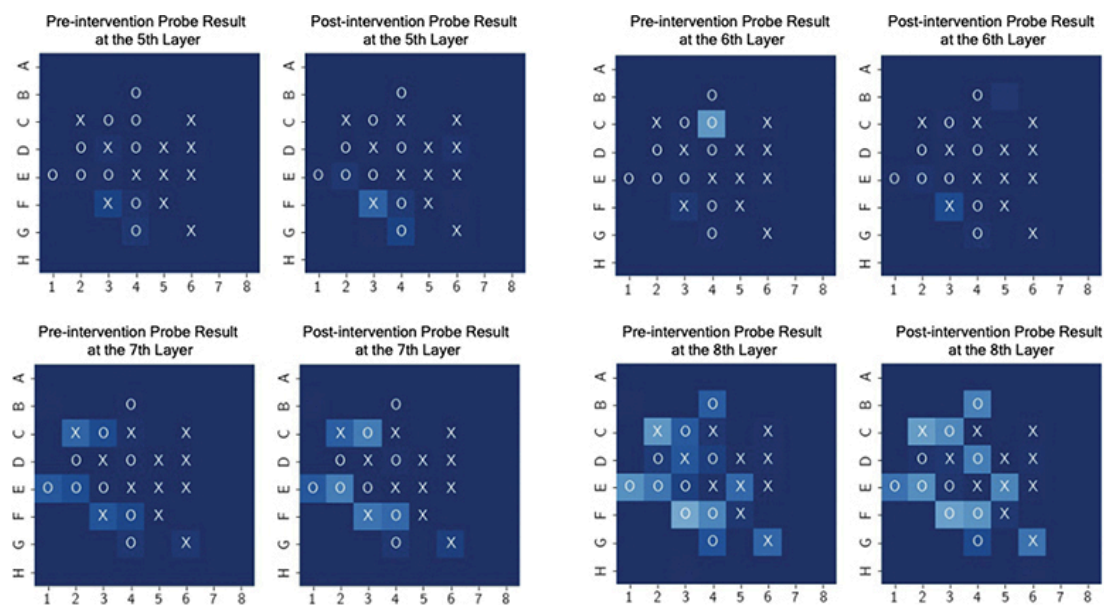


Figure 1.15 LLMs may be able to learn all kinds of things about the world, whether it be the rules and strategy of a game or the rules of human language.

For any LLM to learn any kind of rule, however, it has to convert what we perceive as text into something machine readable. This is done through the process of embedding.

## Embeddings

**Embeddings** are the mathematical representations of words, phrases, or tokens in a large-dimensional space. In NLP, embeddings are used to represent the words, phrases, or tokens in a way that captures their semantic meaning and relationships with other words. Several types of embeddings are possible, including position embeddings, which encode the position of a token in a sentence, and token embeddings, which encode the semantic meaning of a token ([Figure 1.16](#)).



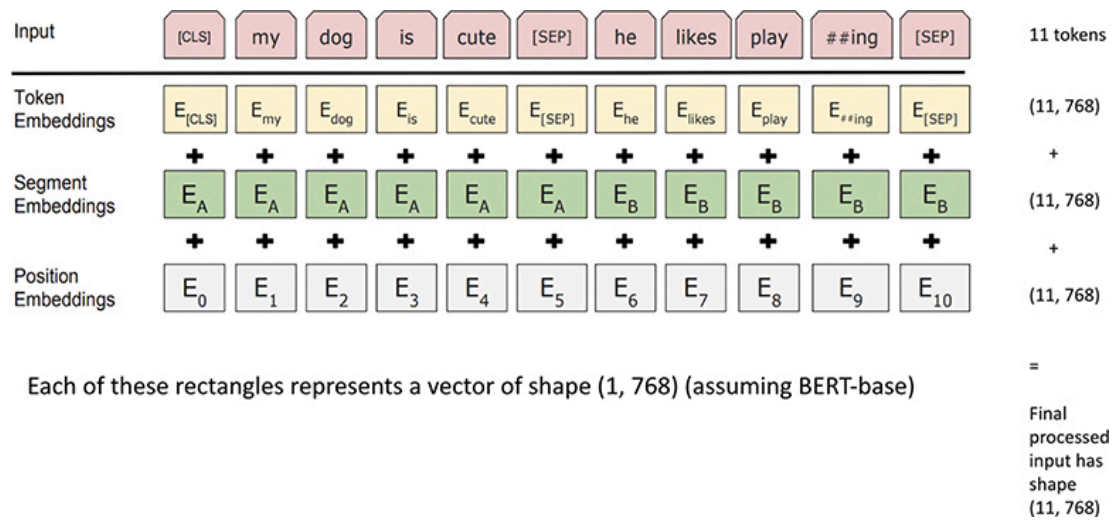


Figure 1.16 An example of how BERT uses three layers of embedding for a given piece of text. Once the text is tokenized, each token is given an embedding and then the values are added up, so each token ends up with an initial embedding before any attention is calculated. We won't focus too much on the individual layers of LLM embeddings in this text unless they serve a more practical purpose, but it is good to know about some of these parts and how they look under the hood.

LLMs learn different embeddings for tokens based on their pre-training and can further update these embeddings during fine-tuning.

## Tokenization

Tokenization, as mentioned previously, involves breaking text down into the smallest unit of understanding—tokens. These tokens are the pieces of information that are embedded into semantic meaning and act as inputs to the attention calculations, which leads to . . . well, the LLM actually learning and working. Tokens make up an LLM's static vocabulary and don't always represent entire words. For example, tokens can represent punctuation, individual characters, or even a sub-word if a word is not known to the LLM. Nearly all LLMs also have *special tokens* that have specific meaning to the model. For example, the BERT model has the special **[CLS]** token, which BERT automatically injects as the first token of every input and is meant to represent an encoded semantic meaning for the entire input sequence.

Readers may be familiar with techniques like stop-words removal, stemming, lemmatization, and truncation that are used in traditional NLP. These techniques are not used, nor are they necessary, for LLMs. LLMs are designed to handle the inherent complexity and variability of human language, including the usage of stop words like “the” and “an,” and variations in word forms like tenses and misspellings. Altering the input text to an LLM using these techniques could potentially harm the model’s performance by reducing the contextual information and altering the original meaning of the text.

Tokenization can also involve preprocessing steps like **casing**, which refers to the capitalization of the tokens. Two types of casing are distinguished: uncased and cased. In uncased tokenization, all the tokens are lowercase, and usually accents are stripped from letters. In cased tokenization, the capitalization of the tokens is preserved. The choice of casing can impact the model’s performance, as capitalization can provide important information about the meaning of a token. [Figure 1.17](#) provides an example.

Uncased Tokenization	Cased Tokenization
Removes accents and lowercases the input	Does nothing to the input
Café Dupont --> cafe dupont	Café Dupont --> Café Dupont

Figure 1.17 The choice of uncased versus cased tokenization depends on the task. Simple tasks like text classification usually prefer uncased tokenization, whereas tasks that derive meaning from case, such as named entity recognition, prefer a cased tokenization.

Note

Even the concept of casing carries some bias, depending on the model. To uncase a text—that is, to implement lowercasing and stripping of accents—is generally a Western-style

preprocessing step. I speak Turkish, so I know that the umlaut (e.g., the “Ö” in my last name) matters and can actually help the LLM understand the word being said in Turkish. Any language model that has not been sufficiently trained on diverse corpora may have trouble parsing and utilizing these bits of context.

**Figure 1.18** shows an example of tokenization—namely, how LLMs tend to handle out-of-vocabulary (OOV) phrases. OOV phrases are simply phrases/words that the LLM doesn’t recognize as a token and has to split up into smaller sub-words. For example, my name (Sinan) is not a token in most LLMs (the story of my life), so in BERT, the tokenization scheme will split my name up into two tokens (assuming uncased tokenization):

- Sin: the first part of my name
- ##an: a special sub-word token that is different from the word “an” and is used only to split up unknown words

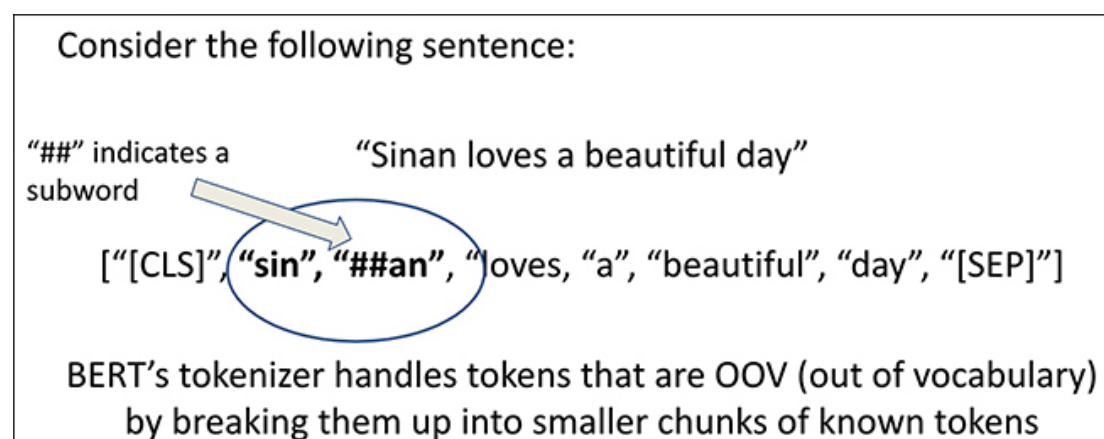


Figure 1.18 Every LLM has to deal with words it has never seen before. How an LLM tokenizes text can matter if we care about the token limit of an LLM. In the case of BERT, “sub-words” are denoted with a preceding “##”, indicating they are part of a single word and not the beginning of a new word. Here the token “##an” is an entirely different token than the word “an”.

Some LLMs limit the number of tokens we can input at any one time. How the LLM tokenizes text can matter if we are trying to be mindful about this limit.

So far, we have talked a lot about language modeling—predicting missing/next tokens in a phrase. However, modern LLMs can also borrow from other fields of AI to make their models more performant and, more importantly, more **aligned**—meaning that the AI is performing in accordance with a human’s expectation. Put another way, an aligned LLM has an objective that matches a human’s objective.

## **Beyond Language Modeling: Alignment + RLHF**

**Alignment** in language models refers to how well the model can respond to input prompts that match the user’s expectations. Standard language models predict the next word based on the preceding context, but this can limit their usefulness for specific instructions or prompts.

Researchers are coming up with scalable and performant ways of aligning language models to a user’s intent. One such broad method of aligning language models is through the incorporation of reinforcement learning (RL) into the training loop. Modern models are even being released in their pre-alignment and post-alignment forms. [Figure 1.19](#) shows Llama-2’s nonaligned and aligned versions answering the same question. The difference is quite stark.

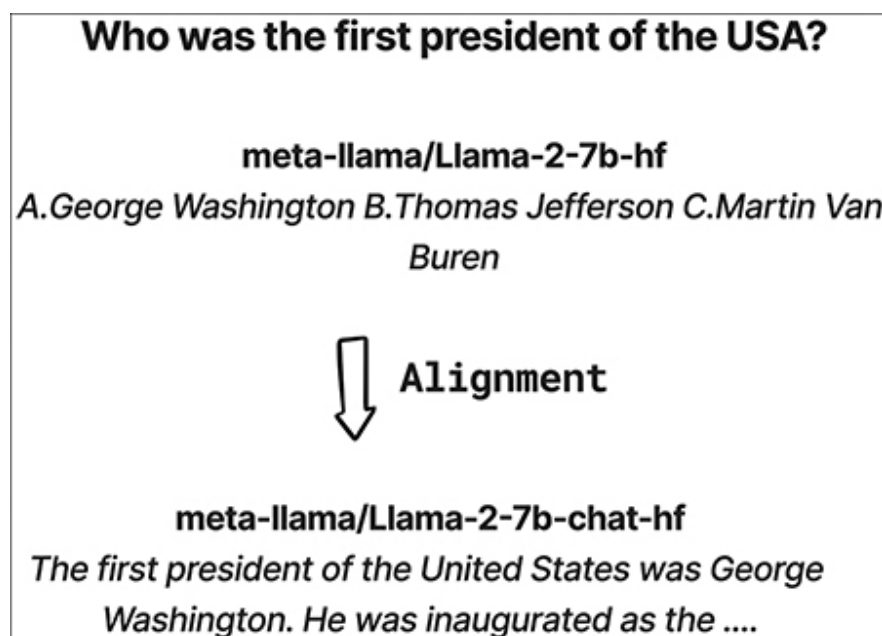


Figure 1.19 Asking the nonaligned (top) and aligned (bottom) versions of Llama-2 who America's first president was yields vastly different answers. The top model was trained only on the autoregressive language modeling task, whereas the bottom model had that plus additional fine-tuning to be able to hold a conversation

**Reinforcement learning from human feedback (RLHF)** is a popular method of aligning pre-trained LLMs that uses human feedback to enhance their performance. It allows the LLM to learn from a relatively small, high-quality batch of human feedback on its own outputs, thereby overcoming some of the limitations of traditional supervised learning. RLHF has shown significant improvements in modern LLMs like ChatGPT. It is one example of approaching alignment with RL, but other approaches are also emerging, such as RL with AI feedback (e.g., constitutional AI). We will explore alignment with reinforcement learning in detail in later chapters by aligning a Llama-3 model from scratch and much more.

## Domain-Specific LLMs

Domain-specific LLMs are LLMs that are trained in a particular subject area, such as biology or finance. Unlike general-purpose LLMs, these models are designed to understand the specific language and concepts used within the domain they were trained on.

One example of a domain-specific LLM is BioGPT (**Figure 1.20**), a domain-specific LLM that was pre-trained on large-scale biomedical literature. This model was developed by an AI healthcare company, Owkin, in collaboration with Hugging Face. The model was trained on a dataset of more than 2 million biomedical research articles, making it highly effective for a wide range of biomedical NLP tasks such as named entity recognition, relationship extraction, and question-answering. BioGPT, whose pre-training encoded biomedical knowledge and domain-specific jargon into the LLM, can be fine-tuned on smaller datasets, making it adaptable for specific biomedical tasks and reducing the need for large amounts of labeled data.

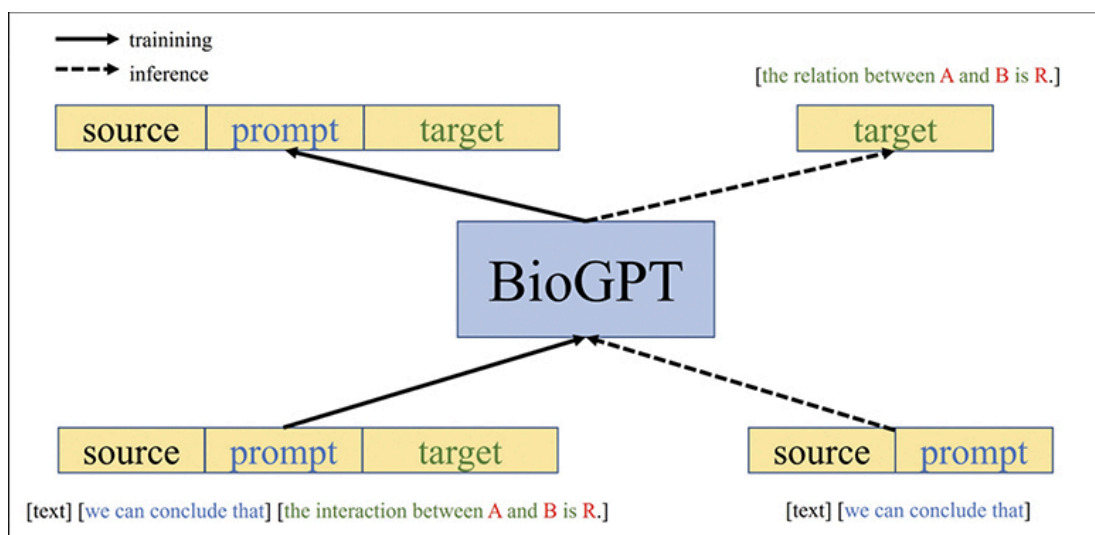


Figure 1.20 BioGPT is a domain-specific Transformer model that was pre-trained on large-scale biomedical literature. BioGPT's success in the biomedical domain has inspired other domain-specific LLMs such as SciBERT and BlueBERT.

The advantage of using domain-specific LLMs lies in their training on a specific set of texts. This relatively narrow, yet extensive pre-training allows them to better understand the language and concepts used within their specific domain, leading to improved accuracy and fluency for NLP tasks that are contained within that domain. By comparison, general-purpose LLMs may struggle to handle the language and concepts used in a specific domain as effectively.

## Applications of LLMs

As we've already seen, applications of LLMs vary widely and researchers continue to find novel applications of LLMs to this day. We will use LLMs in this book in generally three ways:

- Using a pre-trained LLM's underlying ability to process and generate text with no further fine-tuning to **encode** text as vectors as part of a larger architecture
  - Example: creating an information retrieval system using embeddings from a pre-trained BERT/GPT model

- **Fine-tuning** a pre-trained LLM to perform a very specific task using transfer learning and custom data
  - Example: fine-tuning GPT-3.5 (ChatGPT) to create summaries of documents in a specific domain/industry
- Asking a pre-trained LLM to solve a task it was pre-trained to solve or could reasonably intuit—we call this **prompting**
  - Example: prompting GPT-4 to write a blog post
  - Example: prompting Llama-3 to perform language translation

These methods—encoding, fine-tuning, and prompting—use LLMs in different ways. While all of them take advantage of an LLM’s pre-training, only the second option requires any fine-tuning. Let’s look at some specific applications of LLMs.

## **Classical NLP Tasks**

Most applications of LLMs are delivering state-of-the-art results in very common NLP tasks like classification and translation. It’s not that we weren’t solving these tasks before Transformers and LLMs came along; it’s just that now developers and practitioners can solve them with comparatively less labeled data (due to the efficient pre-training of the Transformer on huge corpora) and with a higher degree of accuracy.

### **Text Classification**

The text classification task assigns a label to a given piece of text. This task is commonly used in sentiment analysis, where the goal is to classify a piece of text as positive, negative, or neutral, or in topic classification, where the goal is to classify a piece of text into one or more predefined categories. Models like BERT can be fine-tuned to perform classification with relatively little labeled data, as seen in **Figure 1.21**.



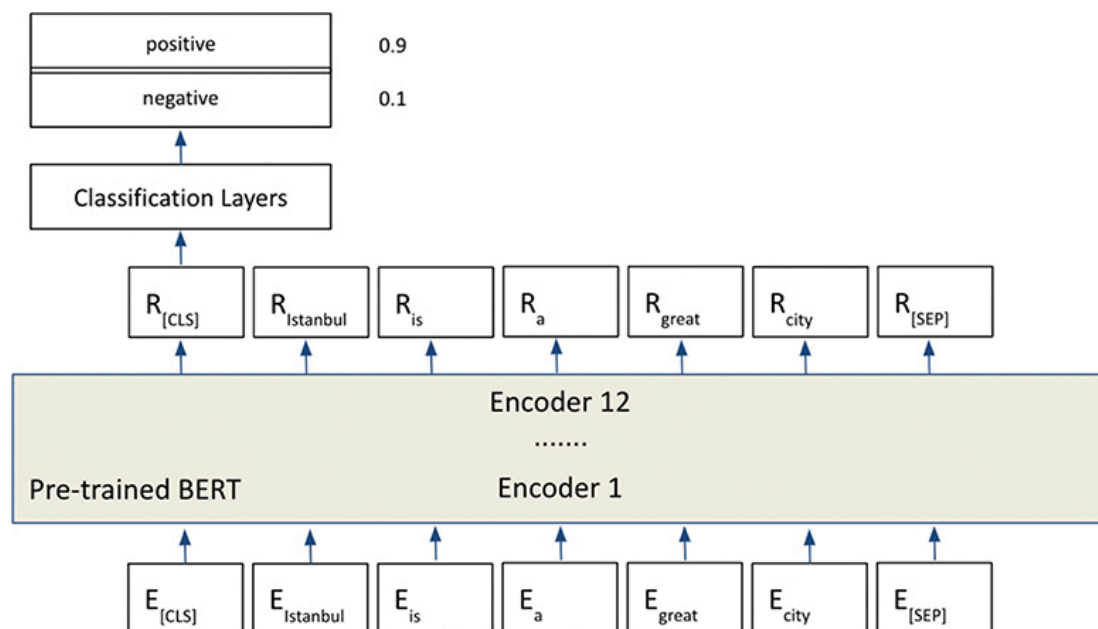


Figure 1.21 A peek at the architecture of using BERT to achieve fast and accurate text classification results. Classification layers usually act on the special [CLS] token that BERT uses to encode the semantic meaning of the entire input sequence.

Text classification remains one of the most globally recognizable and solvable NLP tasks. After all, sometimes we just need to know whether this email is “spam” or not and get on with our day!

## Translation Tasks

A harder, yet still classic NLP task is machine translation, where the goal is to automatically translate text from one language to another while preserving the meaning and context. Traditionally, this task is quite difficult because it involves having sufficient examples and domain knowledge of both languages to accurately gauge how well the model is doing. Modern LLMs seem to have an easier time with this task due to their pre-training and efficient attention calculations.

## Human Language <> Human Language

One of the first applications of attention (even before Transformers emerged) involved machine translation tasks, where AI models were ex-



pected to translate from one human language to another. T5 was one of the first LLMs to tout the ability to perform multiple tasks off the shelf (**Figure 1.22**). One of these tasks was the ability to translate English into a few languages and back.

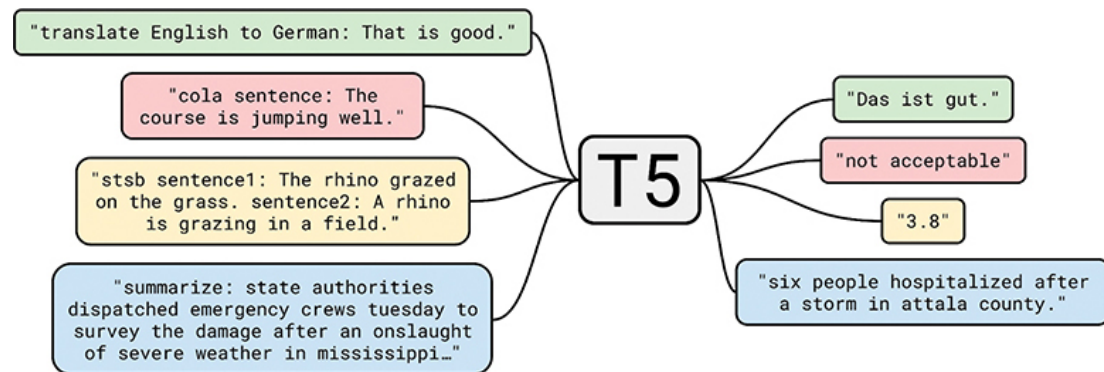


Figure 1.22 T5 could perform many NLP tasks off the shelf, including grammar correction, summarization, and translation.

Since the introduction of T5, language translation in LLMs has only gotten better and more diverse. Models like GPT-4 and the latest T5 models can translate between dozens of languages with relative ease. Of course, this bumps up against one major known limitation of LLMs: They are mostly trained from an English-speaking/usually U.S. point of view. As a result, most LLMs can handle English well and non-English languages, well, not quite so well.

### SQL Generation: Human Language → SQL

If we consider SQL as a language, then converting English to SQL is really not that different from converting English to French (**Figure 1.23**).

Modern LLMs can already do this at a basic level off the shelf, but more advanced SQL queries often require some fine-tuning.

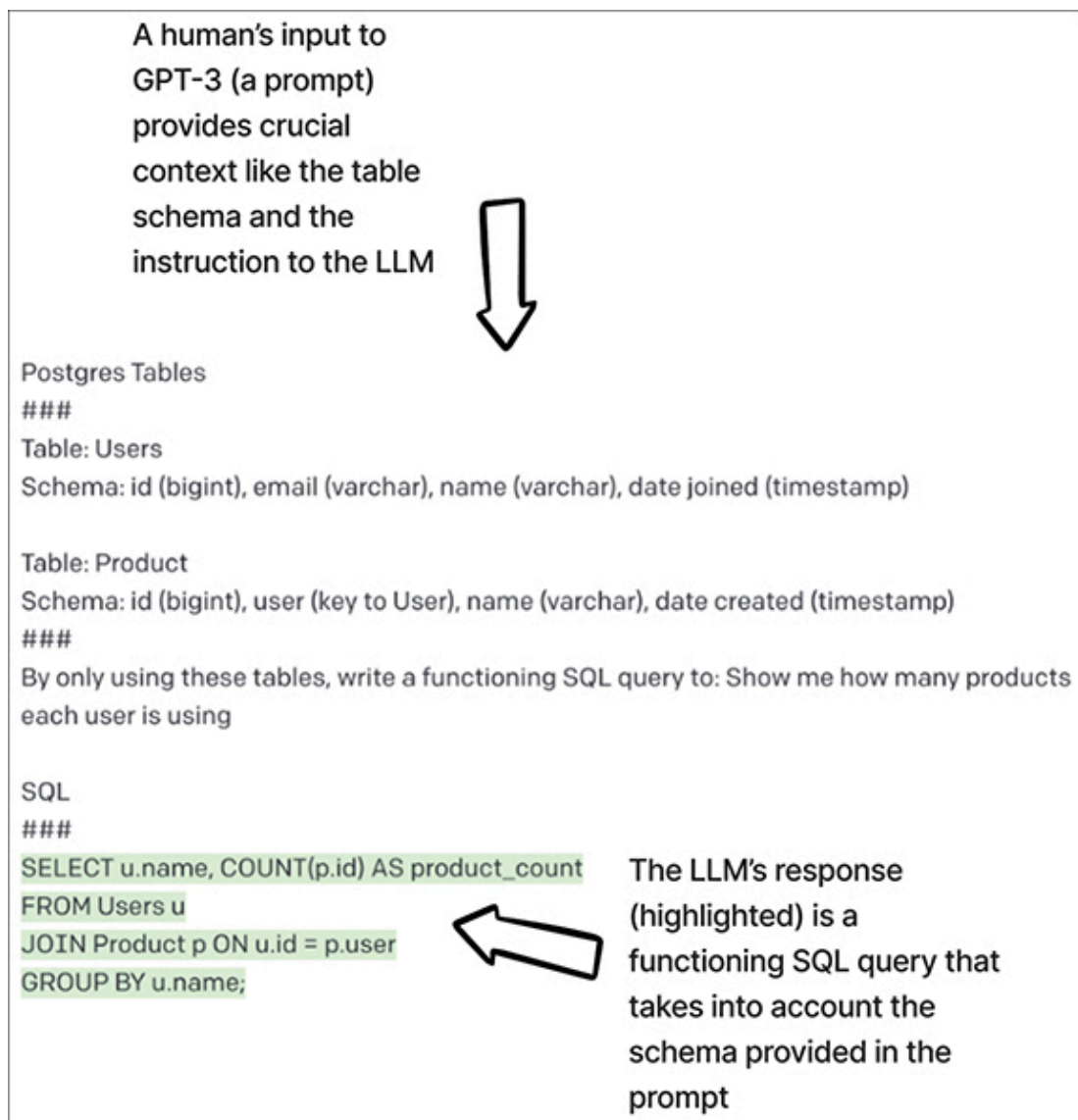



Figure 1.23 Using OpenAI's gpt-3.5-turbo-instruct to generate functioning SQL code from a (simple) Postgres schema.

If we expand our thinking about what can be considered a “translation,” then a lot of new opportunities lie ahead of us. For example, what if we wanted to “translate” between English and a series of wavelengths that a brain might interpret and execute as motor functions? I’m not a neuroscientist, but that seems like a fascinating area of research!


## Free-Text Generation

What first caught the world's eye in terms of modern LLMs like ChatGPT was their ability to freely write blogs, emails, and even academic papers. This notion of text generation is why many LLMs are affectionately referred to as “generative AI,” although that term is a bit reductive and imprecise. I will not often use the term “generative AI,” as the word “generative” has its own meaning in machine learning as the analogous way of learning to a “discriminative” model. (For more on that, check out my other book, *The Principles of Data Science*, Third Edition, published by Packt Publishing.)

We could, for example, prompt (ask) ChatGPT to help plan out a blog post, as shown in **Figure 1.24**. Even if you don't agree with the results, this can help humans with the “tabula rasa” problem and give us something to at least edit and start from rather than staring at a blank page for too long.



Can you write 3 major bullet points for a blog post I want to write on best places to go on a first date in San Francisco?



Sure! Here are three major bullet points you could cover in your blog post on the best places to go on a first date in San Francisco:

1. Romantic Walks and Views: San Francisco is known for its stunning views and romantic locations. Take your date on a stroll through the iconic Golden Gate Park, explore the picturesque Palace of Fine Arts, or enjoy the view of the city from Coit Tower. These locations offer the perfect backdrop for a romantic first date.
2. Food and Drinks: San Francisco is a foodie's paradise, and there are countless amazing restaurants and bars to choose from. Head to the trendy Mission district for some delicious tacos, burritos, and margaritas, or try some fresh seafood at Fisherman's Wharf. And don't forget about the city's famous sourdough bread and Ghirardelli chocolate!
3. Fun Activities: If you're looking for something more active and adventurous, San Francisco has plenty of options for a fun first date. Rent a tandem bike and ride across the Golden Gate Bridge, take a ferry to Alcatraz Island, or explore the famous Haight-Ashbury neighborhood. There's always something exciting to do in the City by the Bay!

Figure 1.24 ChatGPT can help ideate, scaffold, and even write entire blog posts.

---

### Note

I would be remiss if I didn't mention the controversy that LLMs' free-text generation ability can cause at the academic level. Just because an LLM can write entire blogs or even essays, that doesn't mean we should let them do so. Just as the expansion of the internet caused some to believe that we'd never need books again, some argue that ChatGPT means that we'll never need to write anything again. As long as institutions are aware of how to use this technology and proper regulations and rules are put in place, students and teachers alike can use ChatGPT and other text-generation-focused AIs safely and ethically.

---

We will use ChatGPT to solve several tasks in this book. In particular, we will rely on its ability to contextualize information in its context window and freely write back (usually) accurate responses. We will mostly be interacting with ChatGPT through the Playground and the API provided by OpenAI, as this model is not open source.

## Information Retrieval/Neural Semantic Search

LLMs encode information directly into their parameters via pre-training and fine-tuning, but keeping them up to date with new information is tricky. We either have to further fine-tune the model on new data or run the pre-training steps again from scratch. To dynamically keep information fresh, we will architect our own information retrieval system with a vector database (don't worry—we'll go into more details on all of this in [Chapter 2](#)). [Figure 1.25](#) shows an outline of the architecture we will build.

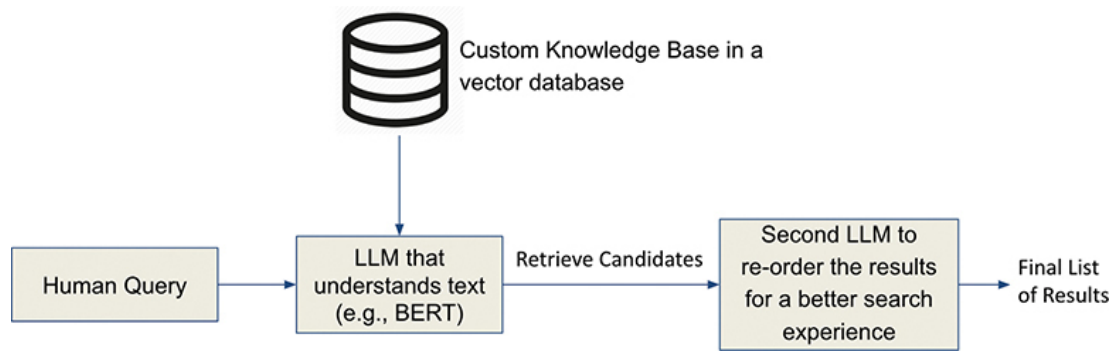


Figure 1.25 Our neural semantic search system will be able to take in new information dynamically and to retrieve relevant documents quickly and accurately given a user’s query using LLMs.

We will then add onto this system by building a ChatGPT-based chatbot to conversationally answer questions from our users.

## Chatbots

Everyone loves a good chatbot, right? Well, whether you love them or hate them, LLMs’ capacity for holding a conversation is evident through systems like ChatGPT and even older models like gpt-3.5-turbo-instruct (as seen in [Figure 1.26](#)). The way we architect chatbots using LLMs will be quite different from the traditional way of designing chatbots through intents, entities, and tree-based conversation flows. These concepts will be replaced by system prompts, context, and personas—all of which we will dive into in the coming chapters.

I am a chatbot. My ultimate goal is to respond with a proper functioning SQL query to pull the data that the human asked for. Only use the following tables:

Table: Users

Schema: id (bigint), email (varchar), name (varchar), date joined (timestamp)

Table: Product

Schema: id (bigint), user (key to User), name (varchar), date created (timestamp)

--- BEGIN CHAT ---

Human: begins chat

Bot: How can I help?

Human: I need to pull some data

Bot: What kind of data do you need?

Human: Can you show me how many users are in the DB?

Bot: Sure, I can help with that. The following SQL query should do the trick:

SELECT COUNT(\*) FROM Users;



GPT-3's multiple responses  
are highlighted. Anything not  
highlighted was human written

Figure 1.26 All text highlighted in green was written by the AI. ChatGPT isn't the only LLM that can hold a conversation. We can use gpt-3.5-turbo-instruct to construct a simple conversational chatbot. The text highlighted in green represents gpt-3.5-turbo-instruct's output. Note that before the chat even begins, I inject context into the prompt that would not be shown to the end user but that the LLM needs to provide accurate responses.

We have our work cut out for us. I'm excited to be on this journey with you, and I'm excited to get started!

## Summary

LLMs are advanced AI models that have revolutionized the field of NLP. LLMs are highly versatile and are used for a variety of NLP tasks, including text classification, text generation, and machine translation. They are pre-trained on large corpora of text data and can then be fine-tuned for specific tasks.



Using LLMs to solve NLP tasks has become a standard step in the development of AI models. In our first case study, we will explore the process of launching an application with both closed-source proprietary models like ChatGPT as well as open-source models. The differences between them begin with their architecture, pre-training, and recommended use-cases, and extend into ethical and philosophical alignment issues. We will get a hands-on look at the practical aspects of using LLMs for real-world NLP tasks, from model selection and fine-tuning to deployment and maintenance.