CHAPTER

7 Large Language Models

"How much do we know at any time? Much more, or so I believe, than we know we know."

Agatha Christie, *The Moving Finger*

The literature of the fantastic abounds in inanimate objects magically endowed with the gift of speech. From Ovid's statue of Pygmalion to Mary Shelley's story about Frankenstein, we continually reinvent stories about creating something and then having a chat with it. Legend has it that after finishing his sculpture *Moses*, Michelangelo thought it so lifelike that he tapped it on the knee and commanded it to speak. Perhaps this shouldn't be surprising. Language is the mark of humanity and sentience. conversation is the most fundamental arena of language, the first kind of language we learn as children, and the kind we engage in constantly, whether we are teaching or learning, ordering lunch, or talking with our families or friends.

This chapter introduces the **Large Language Model**, or **LLM**, a computational agent that can interact conversationally with people. The fact that LLMs are designed for interaction with people has strong implications for their design and use.

Many of these implications already became clear in a computational system from 60 years ago, ELIZA (Weizenbaum, 1966). ELIZA, designed to simulate a Rogerian psychologist, illustrates a number of important issues with chatbots. For example people became deeply emotionally involved and conducted very personal conversations, even to the extent of asking Weizenbaum to leave the room while they were typing. These issues of emotional engagement and privacy mean we need to think carefully about how we deploy language models and consider their effect on the people who are interacting with them.

In this chapter we begin by introducing the computational principles of LLMs; we'll discuss their implementation in the transformer architecture in the following chapter. The central new idea that makes LLMs possible is the idea of **pretraining**, so let's begin by thinking about the idea of learning from text, the basic way that LLMs are trained.

We know that fluent speakers of a language bring an enormous amount of knowledge to bear during comprehension and production. This knowledge is embodied in many forms, perhaps most obviously in the vocabulary, the rich representations we have of words and their meanings and usage. This makes the vocabulary a useful lens to explore the acquisition of knowledge from text, by both people and machines.

Estimates of the size of adult vocabularies vary widely both within and across languages. For example, estimates of the vocabulary size of young adult speakers of American English range from 30,000 to 100,000 depending on the resources used to make the estimate and the definition of what it means to know a word. A simple consequence of these facts is that children have to learn about 7 to 10 words a day, *every single day*, to arrive at observed vocabulary levels by the time they are 20 years of age. And indeed empirical estimates of vocabulary growth in late elementary through high school are consistent with this rate. How do children achieve this rate of vocabulary growth? Research suggests that the bulk of this knowledge acquisition happens as a by-product of reading. Reading is a process of rich contextual processing; we don't learn words one at a time in isolation. In fact, at some points during learning the rate of vocabulary growth exceeds the rate at which new words are appearing to the learner! That suggests that every time we read a word, we are also strengthening our understanding of other words that are associated with it.

Such facts are consistent with the *distributional hypothesis* of Chapter 5, which proposes that some aspects of meaning can be learned solely from the texts we encounter over our lives, based on the complex association of words with the words they co-occur with (and with the words that those words occur with). The distributional hypothesis suggests both that we can acquire remarkable amounts of knowledge from text, and that this knowledge can be brought to bear long after its initial acquisition. Of course, grounding from real-world interaction or other modalities can help build even more powerful models, but even text alone is remarkably useful.

What made the modern NLP revolution possible is that large language models can learn all this knowledge of language, context, and the world simply by being taught to predict the next word, again and again, based on context, in a (very) large corpus of text. In this chapter and the next we formalize this idea that we'll call

pretraining

pretraining—learning knowledge about language and the world from iteratively predicting tokens in vast amounts of text—and call the resulting pretrained models **large language models**. Large language models exhibit remarkable performance on natural language tasks because of the knowledge they learn in pretraining.

What can language models learn from word prediction? Consider the examples below. What kinds of knowledge do you think the model might pick up from learning to predict what word fills the underbar (the correct answer is shown in blue)? Think about this for each example before you read ahead to the next paragraph:.

With roses, dahlias, and peonies, I was surrounded by ______ flowers
The room wasn't just big it was ______ enormous
The square root of 4 is ______ 2
The author of "A Room of One's Own" is ______ Virginia Woolf
The professor said that ______ he

From the first sentence a model can learn ontological facts like that roses and dahlias and peonies are all kinds of flowers. From the second, a model could learn that "enormous" means something on the same scale as big but further along on the scale. From the third sentence, the system could learn math, while from the 4th sentence facts about the world and historical authors. Finally, the last sentence, if a model was exposed to such sentences repeatedly, it might learn to associate professors only with male pronouns, or other kinds of associations that might cause models to act unfairly to different people.

What is a large language model? As we saw back in Chapter 3, a language model is simply a computational system that can predict the next word from previous words. That is, given a context or prefix of words, a language model assigns a probability distribution over the possible next words. Fig. 7.1 sketches this idea.

Of course we've already seen language models! We saw n-gram language models in Chapter 3 and briefly touched on the feedforward network applied to language

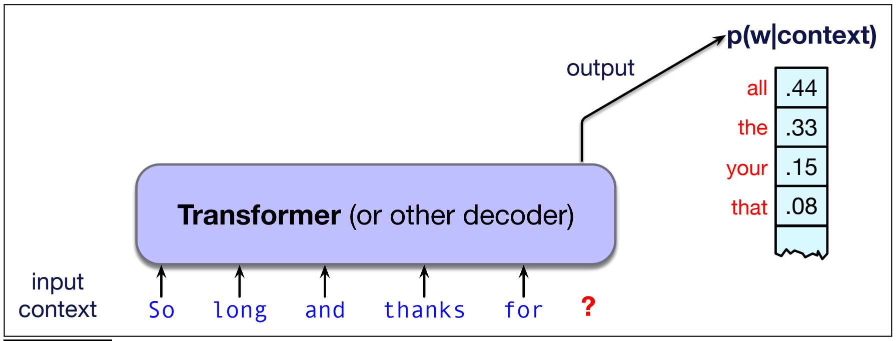

Figure 7.1 A large language model is a neural network that takes as input a context or prefix, and outputs a distribution over possible next words.

modeling in Chapter 6. A large language model is just a (much) larger version of these. For example, in Chapter 3 we introduced bigram and trigram language models that can predict words from the previous word or handful of words. By contrast, large language models can predict words given contexts of thousands or even tens of thousands of words!

The fundamental intuition of language models is that a model that can *predict* text (assigning a distribution over following words) can also be used to *generate* text by **sampling** from the distribution. Recall from Chapter 3 that sampling means to choose a word from a distribution.

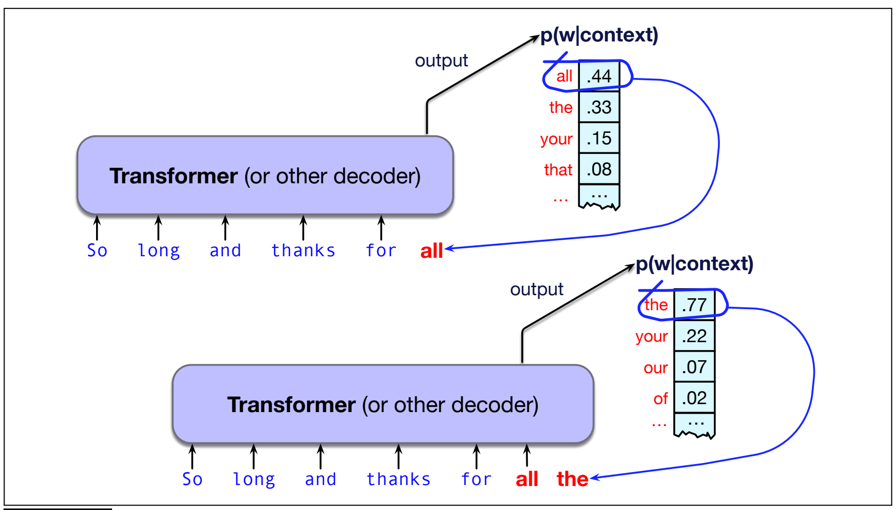

Figure 7.2 Turning a predictive model that gives a probability distribution over next words into a generative model by repeatedly sampling from the distribution. The result is a left-to-right (also called autoregressive) language models. As each token is generated, it gets added onto the context as a prefix for generating the next token.

Fig. 7.2 shows the same example from Fig. 7.1, in which a language model is given a text prefix and generates a possible completion. The model selects the word `all`, adds that to the context, uses the updated context to get a new predictive distribution, and then selects `the` from that distribution and generates it, and so on. Notice that the model is conditioning on both the priming context and its own subsequently generated outputs.

This kind of setting in which we iteratively predict and generate words left-to-right from earlier words is often called **causal** or **autoregressive** language models. (We will introduce alternative non-autoregressive models, like BERT and other masked language models that predict words using information from both the left and the right, in Chapter 10.)

generative AI

This idea of using computational models to generate text, as well as code, speech, and images, constitutes the important new area called **generative AI**. Applying LLMs to generate text has vastly broadened the scope of NLP, which historically was focused more on algorithms for parsing or understanding text rather than generating it.

In the rest of the chapter, we'll see that almost any NLP task can be modeled as word prediction in a large language model, if we think about it in the right way, and we'll motivate and introduce the idea of **prompting** language models. We'll introduce specific algorithms for generating text from a language model, like **greedy decoding** and **sampling**. We'll introduce the details of **pretraining**, the way that language models are self-trained by iteratively being taught to guess the next word in the text from the prior words. We'll sketch out the other two stages of language model training: instruction tuning (also called supervised finetuning or SFT), and alignment, concepts that we'll return to in Chapter 9. And we'll see how to evaluate these models. Let's begin, though, by talking about different kinds of language models.

7.1 Three architectures for language models

The architecture we sketched above for a left-to-right or autoregressive language model, which is the language model architecture we will define in this chapter, is actually only one of three common LM architectures.

The three architectures are the **encoder**, the **decoder**, and the **encoder-decoder**. Fig. 7.3 gives a schematic picture of the three.

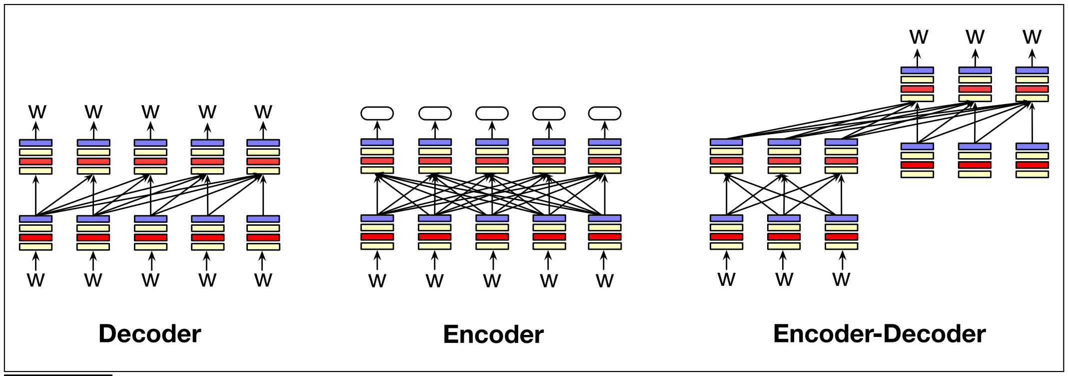

Figure 7.3 Three architectures for language models: decoders, encoders, and encoder-decoders. The arrows sketch out the information flow in the three architectures. Decoders take tokens as input and generate tokens as output. Encoders take tokens as input and produce an encoding (a vector representation of each token) as output. Encoder-decoders take tokens as input and generate a series of tokens as output.

decoder

The **decoder** is the architecture we've introduced above. It takes as input a series of tokens, and iteratively generates an output token one at a time. The decoder is the architecture used to create large language models like GPT, Claude, Llama, and Mistral. The information flow in decoders goes left-to-right, meaning that the model predicts the next word only from the prior words. Decoders are generative models, meaning that, given input tokens, they generate novel output tokens. We'll discuss decoders in the rest of this chapter and in Chapter 8.

encoder

The **encoder** takes as input a sequence of tokens and outputs a vector representation for each tokens. Encoders are usually masked language models, meaning they are trained by masking out a word, and learning to predict it by looking at surrounding words on both sides. Masked language models like BERT, RoBERTA, and others in the BERT family are encoder models. Encoder models are not generative models; they aren't used to generate text. Instead encoder models are often used to create classifiers, for example where the input is text and the output is a label, for example for sentiment or topic or other classes. This is done by finetuning them (training them on supervised data). We'll introduce encoder models in Chapter 10.

encoder-decoder

The **encoder-decoder** takes as input a sequence of tokens and outputs a series of tokens. What makes it different than the decoder-only models, is that an encoder-decoder has a much looser relationship between the input tokens and the output tokens, and they are used to map between different kinds of tokens. That is, in an encoder-decoder, the output tokens might be very different token-set or be much longer or shorter than the input tokens. For example encoder-decoder architectures are used for machine translation, where the input tokens are in one language and and the output tokens are in another language, and probably a different length than the input. Encoder-decoder architectures are also used for speech recognition, where the input is tokens representing speech, and the output is tokens representing text. We'll introduce the encoder-decoder architecture for machine translation in Chapter 12, and for speech recognition in Chapter 15.

These three architectures can be built out of many kinds of neural networks. The most widely used network type today is the **transformer** that we'll introduce in Chapter 8. In a transformer, each input token is processed by a column of transformer layers, each layer composed of a series of different kinds of subnetworks. In Chapter 13 we'll introduce an earlier architecture that is still relevant, the LSTM, a kind of recurrent neural network . And there are many more recent architectures such as the **state space models**.

We'll focus on transformers for much of this book, but for the purposes of this chapter, we'll be architecture-agnostic: we'll treat network that implements the decoder as a black box. The input to this black box is a sequence of tokens, and the output to the box is a distribution over tokens that we can sample from. We'll describe the mechanisms for learning and decoding in a network-agnostic manner.

7.2 Conditional Generation of Text: The Intuition

conditional generation

A fundamental intuition underlying language models is that almost anything we want to do with language can be modeled as **conditional generation** of text. (We mean *decoder* language models, which are what we will discuss in this chapter and the next).

Conditional generation is the task of generating text conditioned on an input piece of text. That is, we give the LLM an input piece of text, a **prompt**, and then have the LLM continue generating text token by token, conditioned on the prompt and the subsequently generated tokens. We generate from a model by first computing the probability of the next token w_i from the prior context: $P(w_i|w_{<i})$ and then sampling from that distribution to generate a token.

We'll talk in future sections about all the details, but in this section our goal is just to establish the intuition. How can simply computing the probability of the next token help an LLM do all sorts of different language-related tasks?

Imagine we want to do a classification tasks like sentiment analysis. We can treat this as conditional generation by giving a language model a context like:

`The sentiment of the sentence ‘‘I like Jackie Chan" is:`

and comparing the conditional probability of the following token "positive" and the following token "negative" to see which is higher. That is, as sketched in Fig. 7.4, we compare these two probabilities:

$$P(\text{“}\texttt{positive}\text{”}|\text{“}\texttt{The sentiment of the sentence ‘I like Jackie Chan’ is:}\text{”})$$
$$P(\text{“}\texttt{negative}\text{”}|\text{“}\texttt{The sentiment of the sentence ‘I like Jackie Chan’ is:}\text{”})$$

If the token "`positive`" is more probable, we could say the sentiment of the sen-

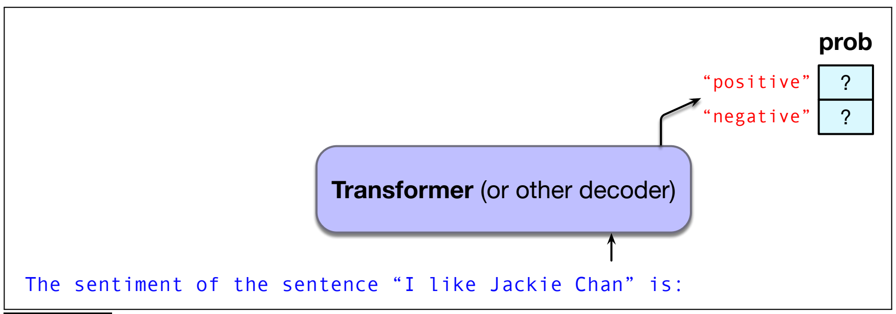

Figure 7.4 Computing the probabilities of the tokens `positive` and `negative` occurring after this prefix.

tence is positive, otherwise if the token "`negative`" is more probable we say the sentiment is negative.

This same intuition can help us perform a task like question answering, in which the system is given a question and must give a textual answer. We can cast the task of question answering as token prediction by giving a language model a question and a token like `A:` suggesting that an answer should come next, like this:

`Q: Who wrote the book ‘‘The Origin of Species"? A:`

Again, we can ask a language model to compute the probability distribution over possible next tokens given this prefix, computing the following probability

$$P(w|\texttt{Q: Who wrote the book ‘‘The Origin of Species"? A:})$$

and look at which tokens w have high probabilities. As Fig. 7.5 suggests, we might expect to see that `Charles` is very likely, and then if we choose `Charles` and add that to our prefix and compute the probability over tokens with this prefix:

$$P(w|\texttt{Q: Who wrote the book ‘‘The Origin of Species"? A: Charles})$$

we might now see that `Darwin` is the most probable token, and select it.

7.3 Prompting

This simple idea of contextual generation is already very powerful, but becomes more powerful when language models are specially trained to answer questions and

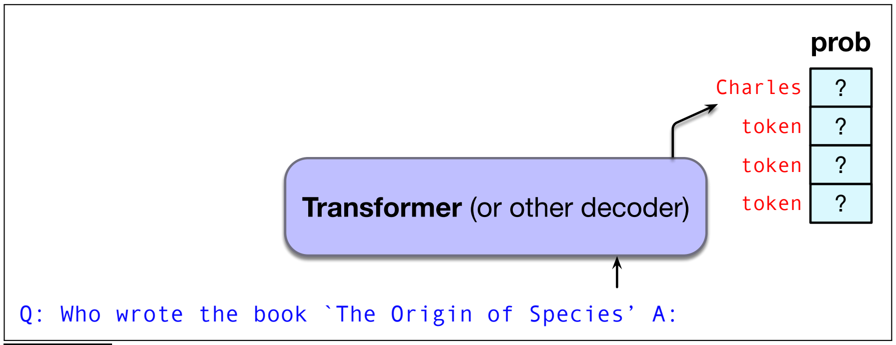

Figure 7.5 Answering a question by computing the probabilities of the tokens after a prefix stating the question; in this example the correct token `Charles` has the highest probability.

follow instructions. This extra training is called **instruction-tuning**. In instruction-tuning we take a base language model that has been trained to predict words, and continue training it on a special dataset of instructions together with the appropriate response to each. The data set has many examples of questions together with their answers, commands with their responses, and other examples of how to carry on a conversation. We'll discuss the details of instruction-tuning in Chapter 9.

prompt Language models that have been instruction-tuned are very good at following instructions and answering questions and carrying on a conversation and can be **prompted**. A **prompt** is a text string that a user issues to a language model to get the model to do something useful. In prompting, the user's prompt string is passed to the language model, which iteratively generates tokens conditioned on the prompt. **prompt engineering** The process of finding effective prompts for a task is known as **prompt engineering**.

As we suggested above when we introduced conditional generation, a prompt can be a question (like "`What is a transformer network?`"), possibly in a structured format (like "`Q: What is a transformer network? A:`"). A prompt can also be an instruction (like "`Translate the following sentence into Hindi: 'Chop the garlic finely'`").

More explicit prompts that specify the set of possible answers lead to better performance. For example here is a prompt template to do sentiment analysis that prespecifies the potential answers:

A prompt consisting of a review plus an incomplete statement

Human: Do you think that "input" has negative or positive sentiment?
Choices:
(P) Positive
(N) Negative

Assistant: I believe the best answer is: (

This prompt uses a number of more sophisticated prompting characteristics. It specifies the two allowable choices (P) and (N), and ends the prompt with the open parenthesis that strongly suggests the answer will be (P) or (N). Note that it also specifies the role of the language model as an assistant.

demonstrations **few-shot** **zero-shot** Including some labeled examples in the prompt can also improve performance. We call such examples **demonstrations**. The task of prompting with examples is sometimes called **few-shot prompting**, as contrasted with **zero-shot** prompting which means instructions that don't include labeled examples. For example Fig. 7.6

shows an example of a question using 2 demonstrations, hence 2-shot prompting. The example is drawn from a computer science question from the the MMLU dataset described in Section 7.6 that is often used to evaluate language models.

Example of demonstrations in a computer science question from the MMLU dataset described in Section 7.6

The following are multiple choice questions about high school computer science.

Let x = 1. What is x << 3 in Python 3?
(A) 1 (B) 3 (C) 8 (D) 16
Answer: C

Which is the largest asymptotically?
(A) O(1) (B) O(n) (C) O(n^2) (D) O(log(n))
Answer: C

What is the output of the statement "a" + "ab" in Python 3?
(A) Error (B) aab (C) ab (D) a ab
Answer:

Figure 7.6 Sample 2-shot prompt from MMLU testing high-school computer science. (The correct answer is (B)).

Demonstrations are generally drawn from a labeled training set. They can be selected by hand, or the choice of demonstrations can be optimized by using an optimizer like DSPy (Khattab et al., 2024) to automatically chose the set of demonstrations that most increases task performance of the prompt on a dev set. The number of demonstrations doesn't need to be large; more examples seem to give diminishing returns, and too many examples seems to cause the model to overfit to the exact examples. The primary benefit of demonstrations seems more to demonstrate the task and the format of the output rather than demonstrating the right answers for any particular question. In fact, demonstrations that have incorrect answers can still improve a system (Min et al., 2022; Webson and Pavlick, 2022).

Prompts are a way to get language models to generate text, but prompts can also can be viewed as a **learning** signal. This is especially clear when a prompt has demonstrations, since the demonstrations can help language models learn to perform novel tasks from these examples of the new task. This kind of learning is different than pretraining methods for setting language model weights via gradient descent methods that we will describe below. The weights of the model are not updated by prompting; what changes is just the context and the activations in the network.

in-context learning

We therefore call the kind of learning that takes place during prompting **in-context learning**—learning that improves model performance or reduces some loss but does not involve gradient-based updates to the model's underlying parameters.

system prompt

Large language models generally have a **system prompt**, a single text prompt that is the first instruction to the language model, and which defines the task or role for the LM, and sets overall tone and context. The system prompt is silently prepended to any user text. So for example a minimal system prompt that creates a multi-turn assistant conversation might be the following including some special metatokens:

```
<system>You are a helpful and knowledgeable assistant. Answer
concisely and correctly.
```

So if a user wants to know the capital of France, the actual text used as the language model's context for conditional generation is:

```
<system> You are a helpful and knowledgeable assistant.
Answer concisely and correctly. <user> What is the capital
of France?
```

The fact that modern language models have such long contexts (tens of thousands of tokens) makes them very powerful for conditional generation, because they can look back so far into the prompting text. That means system prompts, and prompts in general, can be very long.

For example the full system prompt for one language model Anthropic's Claude Opus4, is 1700 words long and includes sentences like the following:

```
Claude should give concise responses to very simple questions,
but provide thorough responses to complex and open-ended
questions.
Claude is able to explain difficult concepts or ideas clearly.
It can also illustrate its explanations with examples, thought
experiments, or metaphors.
Claude does not provide information that could be used to
make chemical or biological or nuclear weapons
For more casual, emotional, empathetic, or advice-driven
conversations, Claude keeps its tone natural, warm, and
empathetic
Claude cares about people's well-being and avoids encouraging
or facilitating self-destructive behavior
If Claude provides bullet points in its response, it should
use markdown, and each bullet point should be at least 1-2
sentences long unless the human requests otherwise
```

It's also possible to create system prompts for other tasks, like the following prompt for creating a general grammar-checker (Anthropic, 2025):

```
Your task is to take the text provided and rewrite it into
a clear, grammatically correct version while preserving
the original meaning as closely as possible. Correct any
spelling mistakes, punctuation errors, verb tense issues,
word choice problems, and other grammatical mistakes.
```

Each user can then make a prompt to have the system fix the grammar of a particular piece of text.

In all these cases, the system prompt is prepended to any user prompts or queries, and the entire string is taking as the context for conditional generation by the language model.

7.4 Generation and Sampling

Which tokens should a language model generate at each step?

The generation depends on the probability of each token, so let's remind ourselves where this probability distribution comes from. The internal networks for language models (whether transformers or alternatives like LSTMs or state space models) generate scores called **logits** (real valued numbers) for each token in the vocabulary. This score vector **u** is then normalized by softmax to be a legal probability distribution, just as we saw for logistic regression in Chapter 4. So if we have a logit vector **u** of shape $[1 \times |V|]$ that gives a score for each possible next token, we can pass it through a softmax to get a vector **y**, also of shape $[1 \times |V|]$, which assigns a probability to each token in the vocabulary, as shown in the following equation:

$$\mathbf{y} = \text{softmax}(\mathbf{u}) \tag{7.1}$$

Fig. 7.7 shows an example in which the softmax is computed for pedagogical purposes on a simplified vocabulary of only 4 words.

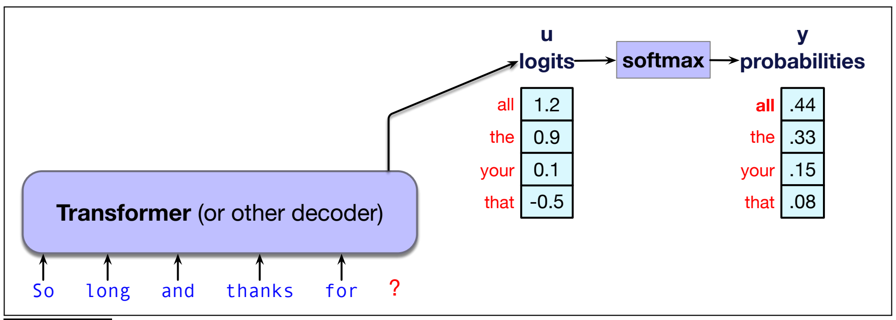

Figure 7.7 Taking the logit vector **u** and using the softmax to create a probability vector **y**.

Now given this probability distribution over tokens, we need to select one token to generate. The task of choosing a token to generate based on the model's probabilities is often called **decoding**. As we mentioned above, decoding from a language model in a left-to-right manner (or right-to-left for languages like Arabic in which we read from right to left), and thus repeatedly choosing the next token conditioned on our previous choices is called **autoregressive generation**.[1]

decoding

autoregressive generation

7.4.1 Greedy decoding

greedy decoding

The simplest way to generate tokens is to always generate the most likely token given the context, which is called **greedy decoding**. A **greedy algorithm** is one that makes a choice that is locally optimal, whether or not it will turn out to have been the best choice with hindsight. Thus in greedy decoding, at each time step in generation, we turn the logits into a probability distribution over tokens and then we choose as the output w_t the token in the vocabulary that has the highest probability (the argmax):

$$\hat{w}_t = \text{argmax}_{w \in V} P(w|\mathbf{w}_{<t}) \tag{7.2}$$

Fig. 7.8 shows that in our example, the model chooses to generate `all`.

[1] Technically an **autoregressive** model predicts a value at time t based on a linear function of the values at times $t-1$, $t-2$, and so on. Although language models are not linear (since, as we will see, they have many layers of non-linearities), we loosely refer to this generation technique as autoregressive since the token generated at each time step is conditioned on the token selected by the network from the previous step. As we'll see, alternatives like the masked language models of Chapter 10 are non-causal because they can predict tokens based on both past and future tokens).

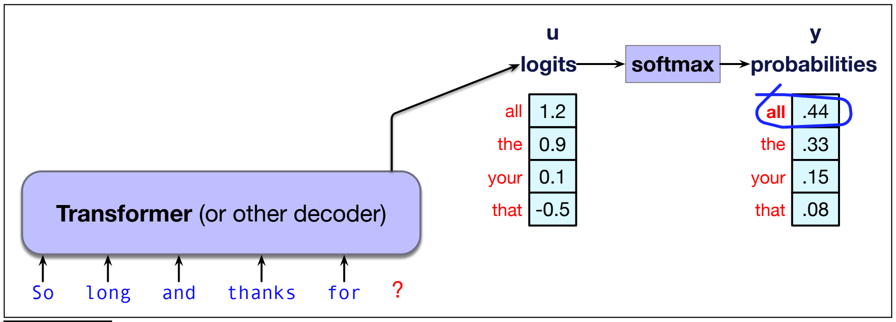

Figure 7.8 Greedy decoding: choose the highest probability word.

In practice, however, we don't use greedy decoding with large language models. A major problem with greedy decoding is that because the tokens it chooses are (by definition) extremely predictable, the resulting text is generic and often quite repetitive. Indeed, greedy decoding is so predictable that it is deterministic; if the context is identical, and the probabilistic model is the same, greedy decoding will always result in generating exactly the same string.

We'll see in Chapter 12 that an extension to greedy decoding called **beam search** works well in tasks like machine translation, which are very constrained in that we are always generating a text in one language conditioned on a very specific text in another language.

In most other tasks, however, people prefer text which has been generated by **sampling methods** that introduce a bit more diversity into the generations.

7.4.2 Random sampling

sampling

Thus the most common method for decoding in large language models involves **sampling**. Recall from Chapter 3 that **sampling** from a distribution means to choose random points according to their likelihood. Thus sampling from a language model—which represents a distribution over following tokens—means to choose the next token to generate according to its probability assigned by the model. Thus we are more likely to generate tokens that the model thinks have a high probability and less likely to generate tokens that the model thinks have a low probability.

That is, we randomly select a token to generate according to its probability in context as defined by the model, generate it, and iterate. We could think of this as rolling a die and choosing a token according to the resulting probability, as we saw in Chapter 3. Such a model is of course more likely to generate the highest probability token, just like the greedy algorithm, but it could also generate any token, just with smaller chances. But in general we are more likely to generate tokens that the model thinks have a high probability in the context and less likely to generate tokens that the model thinks have a low probability.

Sampling from language models was first suggested very early on by Shannon (1948) and Miller and Selfridge (1950), and we saw back in Chapter 3 on page **??** how to generate text from a unigram language model by repeatedly randomly sampling tokens according to their probability until we either reach a pre-determined length or select the end-of-sentence token.

To generate text from a large language model we'll just generalize this model a bit: at each step we'll sample tokens according to their probability *conditioned on our previous choices*, and we'll use the large language model as the probability model that tells us this probability.

random sampling

The algorithm is called **random sampling**, or **random multinomial sampling** (because we are sampling from a multinomial distribution across words). We can formalize random sampling as follows: we are generating a sequence of tokens $\{w_1, w_2, \ldots, w_N\}$ until we hit the end-of-sequence token, using $x \sim p(x)$ to mean 'choose x by sampling from the distribution $p(x)$':

```
i ← 1
w_i ~ p(w)
while w_i != EOS
   i ← i + 1
   w_i ~ p(w_i | w_<i)
```

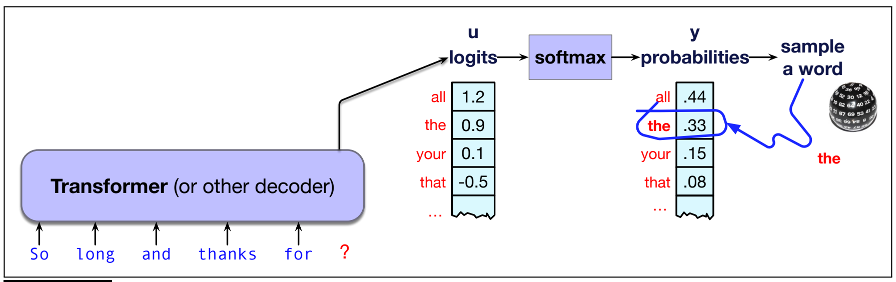

Figure 7.9 Random multinomial sampling: we randomly chose a word according to its probability.

Alas, it turns out random sampling doesn't work well either. The problem is that even though random sampling is mostly going to generate sensible, high-probable tokens, there are many odd, low-probability tokens in the tail of the distribution, and even though each one is low-probability, if you add up all the rare tokens, they constitute a large enough portion of the distribution that they get chosen often enough to result in generating weird sentences.

In other words, greedy decoding is too boring, and random sampling is too random. We need something that doesn't greedily choose the top choice every time, but doesn't stray down too far into the very low-probability events.

There are three standard sampling methods that modify random sampling to address these issues. We'll describe the most common, **temperature** sampling here, and talk about two others (**top-k** and **top-p**) in the next chapter.

7.4.3 Temperature sampling

temperature sampling

The idea of **temperature sampling** is to reshape the probability distribution to increase the probability of the high probability tokens and decrease the probability of the low probability tokens. The result is that we are less likely to generate very low-probability tokens, and more likely to generate tokens that are higher probability.

We implement this intuition by simply dividing the logit by a temperature parameter τ before passing it through the softmax. In low-temperature sampling, $\tau \in (0, 1]$. Thus instead of computing the probability distribution over the vocabulary directly from the logit as in the following (repeated from Eq. **??**):

$$\mathbf{y} = \text{softmax}(\mathbf{u}) \tag{7.3}$$

we instead first divide the logits by τ, computing the probability vector y as

$$\mathbf{y} = \text{softmax}(\mathbf{u}/\tau) \tag{7.4}$$

That is, normally we convert from logits to softmax as shown in Fig. 7.10(a). But when we use a temperature parameter we first scale the logit as in Fig. 7.10(b).

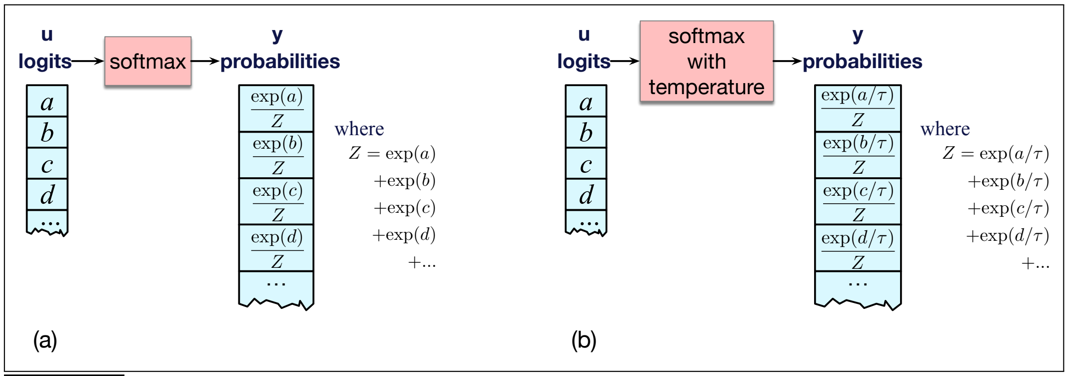

Figure 7.10 (a): Normal softmax without temperature scaling (b) Adding temperature scaling to the softmax by first dividing by the temperature parameter τ.

Why does dividing by τ increase the high probability elements and decrease the low probability elements in the vector over vocabulary items? When τ is 1, we are doing normal softmax, and so when τ is close to 1 the distribution doesn't change much. But the lower τ is, the larger the scores being passed to the softmax (because dividing by a smaller fraction $\tau \leq 1$ results in making each score larger).

Recall that one of the useful properties of a softmax is that it tends to push high values toward 1 and low values toward 0. Thus when larger numbers are passed to a softmax the result is a distribution with increased probabilities of the most high-probability tokens and decreased probabilities of the low probability tokens, making the distribution more greedy. By contrast, as as τ approaches 0 the probability of the most likely word approaches 1, resulting in greedy decoding..

The intuition for temperature sampling comes from thermodynamics, where a system at a high temperature is very flexible and can explore many possible states, while a system at a lower temperature is likely to explore a subset of lower energy (better) states. In low-temperature sampling, we smoothly increase the probability of the most probable tokens and decrease the probability of the rare tokens.

Fig. 7.11 shows a schematic example again simplified to have a vocabulary with only 4 tokens (*all*, *the*, *your*, *that*), and showing how different temperature values influence the probabilities computed from the initial logits. i $\tau = 1$ is the normal softmax, and we can see how setting $\tau = 0.5$ increases the probability of the top candidate from .55 to .59. Setting $\tau = 0.1$ increases the probability of the top candidate from .05, getting us close to greedy decoding.

We can also see in Fig. 7.11 some other options for situations where we may want to *flatten* the word probability distribution instead of making it greedy. Temperature sampling can help with this situation too, in this case **high-temperature** sampling, in which case we use $\tau > 1$.

7.5 Training Large Language Models

How do we learn a language model? What is the algorithm and what data do we train on?

Language models are trained in three stages, as shown in Fig. 7.12:

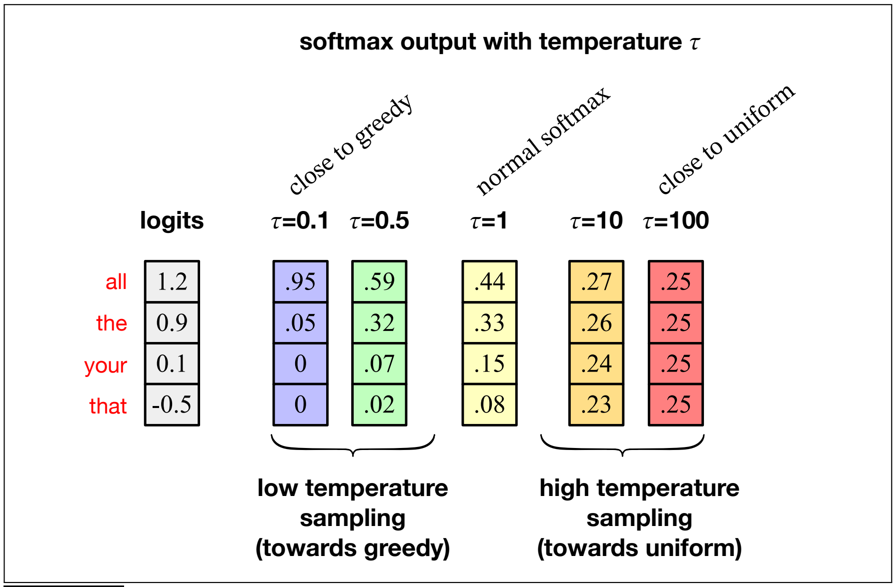

Figure 7.11 Seeing how different values of τ change the resulting probabilities from the initial logits in temperature sampling. In this simplified example, there are only 4 tokens in the vocabulary.

1. **pretraining**: In this first stage, the model is trained to incrementally predict the next word in enormous text corpora. The model uses the cross-entropy loss, sometimes called the **language modeling loss**, and that loss is backpropagated all the way through the network. The training data is usually based on cleaning up parts of the web. The result is a model that is very good at predicting words and can generate text.
2. **instruction tuning**, also called supervised finetuning or **SFT**: In the second stage, the model is trained, again by cross-entropy loss to follow instructions, for example to answer questions, give summaries, write code, translate sentences, and so on. It does this by being trained on a special corpus with lots of text containing both instructions and the correct response to the instruction.
3. **alignment**, also called **preference alignment**. In this final stage, the model is trained to make it maximally helpful and less harmful. Here the model is given preference data, which consists of a context followed by two potential continuations , which are labeled (usually by people) as an 'accepted' vs a 'rejected' continuation. The model is then trained, by reinforcement learning or other reward-based algorithms, to produce the accepted continuation and not the rejected continuation.

We'll introduce pretraining next, but we'll save instruction tuning and preference alignment for Chapter 9.

7.5.1 Self-supervised training algorithm for pretraining

self-training

The intuition of pretraining large language models, is the same idea of **self-training** or **self-supervision** that we saw in Chapter 5 for learning word representations like word2vec. In self-training for language modeling, we take a corpus of text as training material and at each time step t ask the model to predict the next word. At first it will do poorly at this task, but since in each case we know the correct answer (it's

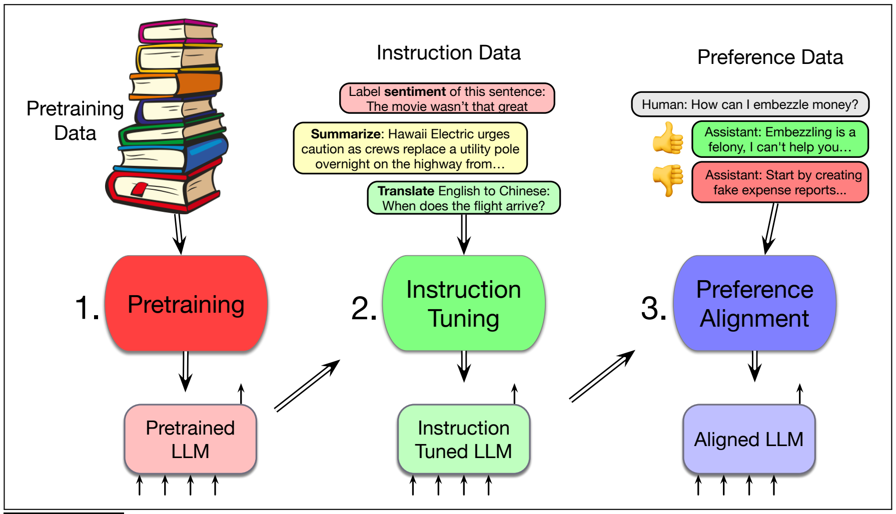

Figure 7.12 Three stages of training large language models: pretraining, instruction tuning, and preference alignment.

the next word in the corpus!) over time it well get better and better at predicting the correct next word. We call such a model self-supervised because we don't have to add any special gold labels to the data; the natural sequence of words is its own supervision! We simply train the model to minimize the error in predicting the true next word in the training sequence.

In practice, training the language model means setting the parameters of the underlying architecture. The transformer that we will introduce in the next chapter has various weight matrices for its feedforward and attention components. Like any other neural architecture, they will be trained by error backpropagation with gradient descent. So all we need is a loss function to minimize and pass back through the network. The loss function we use for language modeling is the cross-entropy loss function we've now seen twice, in Chapter 4 and Chapter 6.

Recall that the cross-entropy loss measures the difference between a predicted probability distribution and the correct distribution. The probability distribution is over the token vocabulary, making the loss be:

$$L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w] \tag{7.5}$$

In the case of language modeling, the correct distribution $\mathbf{y}_t$ comes from knowing the next word. This is represented as a one-hot vector corresponding to the vocabulary where the entry for the actual next word is 1, and all the other entries are 0. Thus, the cross-entropy loss for language modeling is determined by the probability the model assigns to the correct next token (all other tokens get multiplied by zero by the first term in Eq. 7.5).

So without loss of generality we can say that at time t the cross-entropy loss in Eq. 7.5 can be simplified as the negative log probability the model assigns to the next word in the training sequence, $-\log p(w_{t+1})$, or more formally, using $\hat{\mathbf{y}}$ to mean the the vector of estimated token probabilities from the language model:

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}] \tag{7.6}$$

Thus at each word position t of the input, the model takes as input the correct sequence of tokens $w_{1:t}$, and uses them to compute a probability distribution over possible next tokens so as to compute the model's loss for the next token w_{t+1}. Then we move to the next word, we ignore what the model predicted for the next word and instead use the correct sequence of tokens $w_{1:t+1}$ to get the model to estimate the probability of token w_{t+2}. This idea that we always give the model the correct history sequence to predict the next word (rather than feeding the model its best guess from the previous time step) is called **teacher forcing**.

teacher forcing

Fig. 7.13 illustrates the general training approach. At each step, given all the preceding tokens, the language model produces an output distribution over the entire vocabulary. During training, the probability assigned to the correct word is used to calculate the cross-entropy loss for each item in the sequence. The loss for each batch is the average cross-entropy loss over the entire sequence of negative log probabilities, or more formally:

$$L_{CE}(\text{batch of length T}) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\mathbf{y}}_t[w_t] \quad (7.7)$$

The weights in the network are then adjusted to minimize this average cross-entropy loss over the batch via gradient descent (Fig. **??**), using error backpropagation on the computation graph to compute the gradient. Training adjusts all the weights of the network. For the transformer model we will introduce in the next chapter, these weights include the embedding matrix **E** that contains the embeddings for each word. Thus embeddings will be learned that are most successful at predicting upcoming words.

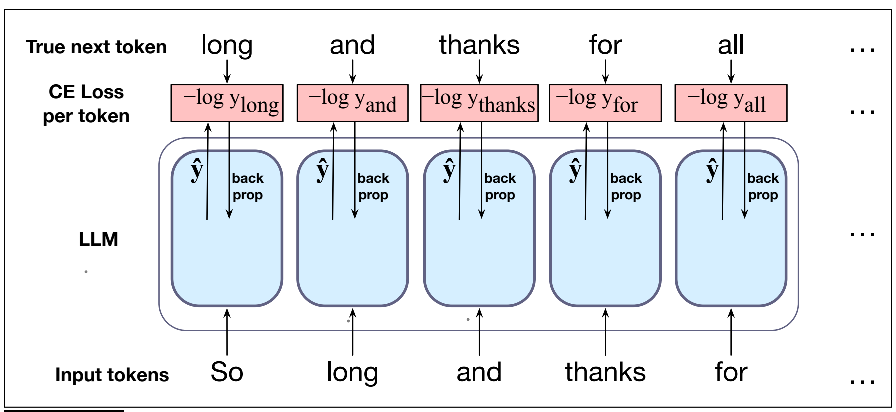

Figure 7.13 Training an LLM. At each token position, the model passes up $\hat{\mathbf{y}}$, its probability estimate for all possible next words. The negative log of the model's probability estimate for the correct token is used as the loss, which is then backpropagated through the model to train all the weights, including the embeddings. Losses are averaged over all the tokens in a batch.

More details of training of course depend on the specific network architecture used to implement the model; we'll see more details specifically for the transformer model in the next chapter.

7.5.2 Pretraining corpora for large language models

Large language models are mainly trained on text scraped from the web, augmented by more carefully curated data. Because these training corpora are so large, they are likely to contain many natural examples that can be helpful for NLP tasks, such as question and answer pairs (for example from FAQ lists), translations of sentences between various languages, documents together with their summaries, and so on.

Web text is usually taken from corpora of automatically-crawled web pages like the **common crawl**, a series of snapshots of the entire web produced by the non-profit Common Crawl (https://commoncrawl.org/) that each have billions of webpages. Various versions of common crawl data exist, such as the Colossal Clean Crawled Corpus (C4; Raffel et al. 2020), a corpus of 156 billion tokens of English that is filtered in various ways (deduplicated, removing non-natural language like code, sentences with offensive words from a blocklist). This C4 corpus seems to consist in large part of patent text documents, Wikipedia, and news sites (Dodge et al., 2021).

common crawl

Wikipedia plays a role in lots of language model training, as do corpora of books. **The Pile** (Gao et al., 2020) is an 825 GB English text corpus that is constructed by publicly released code, containing again a large amount of text scraped from the web as well as books and Wikipedia; Fig. 7.14 shows its composition. Dolma is a larger open corpus of English, created with public tools, containing three trillion tokens, which similarly consists of web text, academic papers, code, books, encyclopedic materials, and social media (Soldaini et al., 2024).

The Pile

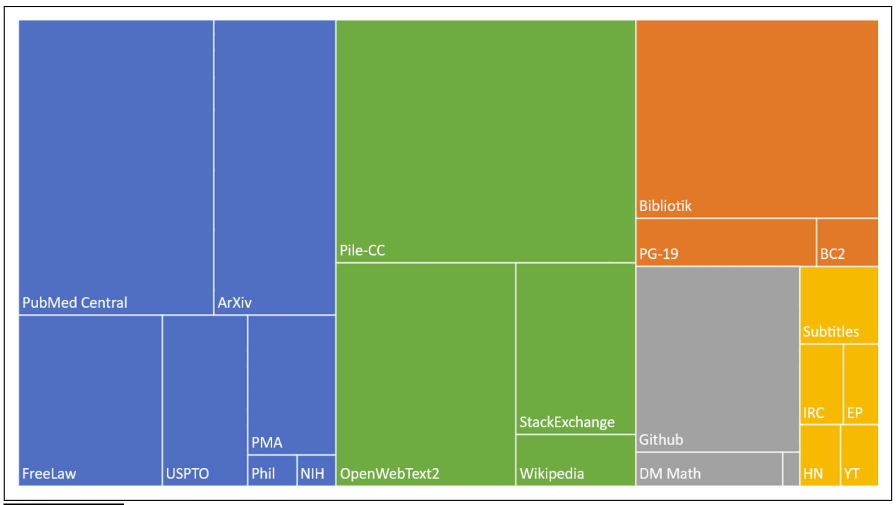

Figure 7.14 The Pile corpus, showing the size of different components, color coded as academic (articles from PubMed and ArXiv, patents from the USPTA; internet (webtext including a subset of the common crawl as well as Wikipedia), prose (a large corpus of books), dialogue (including movie subtitles and chat data), and misc.. Figure from Gao et al. (2020).

Filtering for quality and safety Pretraining data drawn from the web is filtered for both **quality** and **safety**. Quality filters are classifiers that assign a score to each document. Quality is of course subjective, so different quality filters are trained in different ways, but often to value high-quality reference corpora like Wikipedia, books, and particular websites and to avoid websites with lots of **PII** (Personal Identifiable Information) or adult content. Filters also remove boilerplate text which is very frequent on the web. Another kind of quality filtering is deduplication, which can be done at various levels, so as to remove duplicate documents, duplicate web pages, or duplicate text. Quality filtering generally improves language model performance (Longpre et al., 2024b; Llama Team, 2024).

PII

Safety filtering is again a subjective decision, and often includes **toxicity** detection based on running off-the-shelf toxicity classifiers. This can have mixed results. One problem is that current toxicity classifiers mistakenly flag non-toxic data if it is generated by speakers of minority dialects like African American English (Xu et al., 2021). Another problem is that models trained on toxicity-filtered data, while somewhat less toxic, are also worse at detecting toxicity themselves (Longpre et al., 2024b). These issues make the question of how to do better safety filtering an important open problem.

Using large datasets scraped from the web to train language models poses ethical and legal questions:

Copyright: Much of the text in these large datasets (like the collections of fiction and non-fiction books) is copyrighted. In some countries, like the United States, the **fair use** doctrine may allow copyrighted content to be used for transformative uses, but it's not clear if that remains true if the language models are used to generate text that competes with the market for the text they are trained on (Henderson et al., 2023).

Data consent: Owners of websites can indicate that they don't want their sites to be crawled by web crawlers (either via a robots.txt file, or via Terms of Service). Recently there has been a sharp increase in the number of websites that have indicated that they don't want large language model builders crawling their sites for training data (Longpre et al., 2024a). Because it's not clear what legal status these indications have in different countries, or whether these restrictions are retroactive, what effect this will have on large pretraining datasets is unclear.

Privacy: Large web datasets also have **privacy** issues since they contain private information like phone numbers and email addresses. While filters are used to try to remove websites likely to contain large amounts of personal information, such filtering isn't sufficient. We'll return to the privacy question in Section 7.7.

Skew: Training data is also disproportionately generated by authors from the US and from developed countries, which likely skews the resulting generation toward the perspectives or topics of this group alone.

7.5.3 Finetuning

Although the vast pretraining data for large language models includes text from many domains, we might want to apply it in a new domain or task that didn't appear sufficiently in the pretraining data. For example, we might want a language model that's specialized to legal or medical text. Or we might have a multilingual language model that knows many languages but might benefit from some more data in our particular language of interest.

In such cases, we can simply continue training the model on relevant data from the new domain or language (Gururangan et al., 2020). This process of taking a fully pretrained model and running additional training passes using the cross-entropy loss on some new data is called **finetuning**. The word "finetuning" means the process of taking a pretrained model and further adapting some or all of its parameters to some new data. Over the next few chapters we'll see a number of different ways that the word 'finetuning' is used, based on exactly which parameters get updated. The method we describe here, in which we just continue to train, as if the new data was at the end of our pretraining data, can also be called **continued pretraining**. Fig. 7.15 sketches the paradigm.

finetuning

continued pretraining

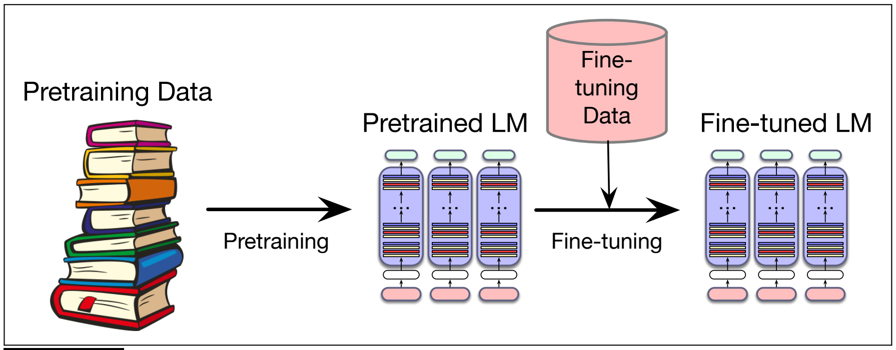

Figure 7.15 Pretraining and finetuning. A pre-trained model can be finetuned to a particular domain or dataset. There are many different ways to finetune, depending on exactly which parameters are updated from the finetuning data: all the parameters, some of the parameters, or only the parameters of specific extra circuitry, as we'll see in future chapters.

7.6 Evaluating Large Language Models

We can evaluate language models by accuracy (how well they predict unseen text, by how well they perform tasks like answering questions or translating text), or by other factors like how fast they can be run, how much energy they use, or how fair they are. We'll explore all of these in the next three sections.

7.6.1 Perplexity

As we first saw in Chapter 3, one way to evaluate language models is to measure how well they predict unseen text. A better language model is better at predicting upcoming words, and so it will be less surprised by (i.e., assign a higher probability to) each word when it occurs in the test set.

If we want to know which of two language models is a better model of some text, we can just see which assigns it a higher probability, or in practice, since we mostly deal with probabilities in log space, we see which assigns a higher log likelihood.

We've been talking about predicting one word at a time, computing the probability of the next token w_i from the prior context: $P(w_i|w_{<i})$. But of course as we saw in Chapter 3 the chain rule allows us to move between computing the probability of the next token and computing the probability of a whole text:

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_{<i}) \end{aligned} \tag{7.8}$$

We can compute the probability of text just by multiplying the conditional probabilities for each token in the text. The resulting (log) likelihood of a text is a useful metric for comparing how good two language models are on that text:

$$\text{log likelihood}(w_{1:n}) = \log \prod_{i=1}^{n} P(w_i|w_{<i}) \tag{7.9}$$

However, we often use another metric other than log likelihood to evaluate language models. The reason is that the probability of a test set (or any sequence) depends on the number of words or tokens in it. In fact, the probability of a test set gets smaller the longer the text is; this is clear from the chain rule, since if we are multiplying more probabilities, and each probability by definition is less than zero, the product will get smaller and smaller. So it's useful to have a metric that is per-token, normalized by length, so we could compare across texts of different lengths.

perplexity

A function of probability called **perplexity** is such a length-normalized metric. Recall from page **??** that the perplexity of a model θ on an unseen test set is the inverse probability that θ assigns to the test set (one over the probability of the test set), normalized by the test set length in tokens. For a test set of n tokens $w_{1:n}$, the perplexity is

$$\begin{aligned} \text{Perplexity}_\theta(w_{1:n}) &= P_\theta(w_{1:n})^{-\frac{1}{n}} \\ &= \sqrt[n]{\frac{1}{P_\theta(w_{1:n})}} \end{aligned} \tag{7.10}$$

To visualize how perplexity can be computed as a function of the probabilities the LM computes for each new word, we can use the chain rule to expand the computation of probability of the test set:

$$\text{Perplexity}_\theta(w_{1:n}) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P_\theta(w_i|w_{<i})}} \tag{7.11}$$

Note that because of the inverse in Eq. 7.10, the higher the probability of the word sequence, the lower the perplexity. Thus the **the lower the perplexity of a model on the data, the better the model**. Minimizing perplexity is equivalent to maximizing the test set probability according to the language model. Why does perplexity use the inverse probability? The inverse arises from the original definition of perplexity from cross-entropy rate in information theory; for those interested, the explanation is in Section **??**. Meanwhile, we just have to remember that perplexity has an inverse relationship with probability.

One caveat: because perplexity depends on the number of tokens n in a text, it is very sensitive to differences in the tokenization algorithm. That means that it's hard to exactly compare perplexities produced by two language models if they have very different tokenizers. For this reason perplexity is best used when comparing language models that use the same tokenizer.

7.6.2 Downstream tasks: Reasoning and world knowledge

Perplexity measures one kind of accuracy: accuracy at predicting words. But there are other kinds of accuracy. For each of the downstream tasks we want to apply our language model, like question answering, machine translation, or reasoning, we could measure the accuracy at those tasks. We'll have further discussion of these task-specific evaluations in future chapters; machine translation in Chapter 12, information retrieval in Chapter 11, and speech recognition in Chapter 15.

Here we briefly introduce one such metric: a mechanism for measuring accuracy in answering questions, focusing on multiple-choice questions. This dataset is

MMLU

MMLU (Massive Multitask Language Understanding), a commonly-used dataset of 15,908 knowledge and reasoning questions in 57 areas including medicine, mathematics, computer science, law, and others. Accuracy at answering these multiple-choice questions can be a useful proxy for the model's ability to reason, and its factual knowledge.

For example, here is an MMLU question from the microeconomics domain:[2]

MMLU microeconomics example

One of the reasons that the government discourages and regulates monopolies is that
(A) producer surplus is lost and consumer surplus is gained.
(B) monopoly prices ensure productive efficiency but cost society allocative efficiency.
(C) monopoly firms do not engage in significant research and development.
(D) consumer surplus is lost with higher prices and lower levels of output.

Fig. 7.16 shows the way MMLU turns these questions into prompted tests of a language model, in this case showing an example prompt with 2 demonstrations.

MMLU mathematics prompt

The following are multiple choice questions about high school mathematics.
How many numbers are in the list 25, 26, ..., 100?
(A) 75 (B) 76 (C) 22 (D) 23
Answer: B

Compute $i + i^2 + i^3 + \cdots + i^{258} + i^{259}$.
(A) -1 (B) 1 (C) i (D) -i
Answer: A

If 4 daps = 7 yaps, and 5 yaps = 3 baps, how many daps equal 42 baps?
(A) 28 (B) 21 (C) 40 (D) 30
Answer:

Figure 7.16 Sample 2-shot prompt from MMLU testing high-school mathematics. (The correct answer is (C)).

data contamination

Taking performance on MMLU as a metric for language model quality has a problem, though, one that is true of all evaluations based on public datasets. The problem is **data contamination**. Data contamination is when some part of a dataset that we are testing on (a test set of any kind) makes its way into our training set. For example, since large language models train on the web, and MMLU is on the web, models may well incorporate some MMLU questions into their training. If those questions are used for evaluation, the metric will overstate the performance of the language model. One way to mitigate data contamination is to make available the exact training data used to train a model, or at least to report training overlap with specific test sets (Zhang et al., 2025).

7.6.3 Other factors for evaluating language models

Accuracy isn't the only thing we care about in evaluating models (Dodge et al., 2019; Ethayarajh and Jurafsky, 2020, inter alia). For example, we often care about how big a model is, and how long it takes to train or do inference. We often have limited time, or limited memory, since the GPUs we run our models on have fixed memory

[2] For those of you whose economics is a bit rusty, the correct answer is (D).

sizes. Big models also use more energy, and we prefer models that use less energy, both to reduce the environmental impact of the model and to reduce the financial cost of building or deploying it. We can target our evaluation to these factors by measuring performance normalized to a given compute or memory budget. We can also directly measure the energy usage of our model in kWh or in kilograms of CO_2 emitted (Strubell et al., 2019; Henderson et al., 2020; Liang et al., 2023).

Another feature that a language model evaluation can measure is fairness. We know that language models are biased, exhibiting gendered and racial stereotypes, or decreased performance for language from or about certain demographics groups. There are language model evaluation benchmarks that measure the strength of these biases, such as StereoSet (Nadeem et al., 2021), RealToxicityPrompts (Gehman et al., 2020), and BBQ (Parrish et al., 2022) among many others. We also want language models whose performance is equally fair to different groups. For example, we could choose an evaluation that is fair in a Rawlsian sense by maximizing the welfare of the worst-off group (Rawls, 2001; Hashimoto et al., 2018; Sagawa et al., 2020).

Finally, there are many kinds of leaderboards like Dynabench (Kiela et al., 2021) and general evaluation protocols like HELM (Liang et al., 2023); we will return to these in later chapters when we introduce evaluation metrics for specific tasks like question answering and information retrieval.

7.7 Ethical and Safety Issues with Language Models

Ethical and safety issues have been key to how we think about designing artificial agents since well before we had large language models. Mary Shelley (depicted below) centered her novel *Frankenstein* around the problem of creating artificial agents without considering ethical and humanistic concerns.

hallucination

Large language models can be unsafe in many ways. For example, LLMs are prone to saying things that are false, a problem called **hallucination**. Language models are trained to generate text that is predictable and coherent, but the training algorithms we have seen so far don't have any way to enforce that the text that is generated is correct or true. This causes enormous problems for any application where the facts matter! A related symptom is that language models can **suggest unsafe actions**, for example directly suggesting that users do dangerous or illegal things like harming themselves or others. If users seek information from language models in safety-critical situations like asking medical advice, or in emergency situations, or when indicating the intentions of self-harm, incorrect advice can be dangerous and even life-threatening. Again, this problem predates large language models For example (Bickmore et al., 2018) gave participants medical problems to pose to three pre-LLM commercial dialogue systems (Siri, Alexa, Google Assistant) and asked them to determine an action to take based on the system responses; many of the proposed actions, if actually taken, would have led to harm or death. We'll return to the issue of hallucination and factuality in Chapter 11 where we introduce proposed mitigation methods like **retrieval augmented generation**, and Chapter 9 where we discussed safety tuning and alignment.

A system can also harm users by verbally **attacking** them, or creating **representational harms** (Blodgett et al., 2020) for example by generating abusive or harmful stereotypes (Cheng et al., 2023) and negative attitudes (Brown et al., 2020; Sheng et al., 2019) that demean particular groups of people; both abuse and stereotypes can cause psychological harm to users. Gehman et al. (2020) show that even completely non-toxic prompts can lead large language models to output hate speech and abuse their users. Liu et al. (2020) testing how systems responded to pairs of simulated user turns that were identical except for mentioning different genders or race. They found, for example, that simple changes like using the word 'she' instead of 'he' in a sentence caused systems to respond more offensively and with more negative sentiment. Hofmann et al. (2024) found that LLMs were likely to discriminate against people just because they used particular dialects like African-American English. Again, these problems predate large language models. Microsoft's 2016 **Tay** chatbot, for example, was taken offline 16 hours after it went live, when it began posting messages with racial slurs, conspiracy theories, and personal attacks on its users. Tay had learned these biases and actions from its training data, including from users who seemed to be purposely teaching the system to repeat this kind of language (Neff and Nagy 2016).

Tay

Another important ethical and safety issue is **privacy**. Privacy has been a concern from the very beginning of computing when Weizenbaum designed the chatbot ELIZA as an experiment in computational therapy (Weizenbaum, 1966). First, people became deeply emotionally involved and conducted very personal conversations with the ELIZA chatbot, even to the extent of asking Weizenbaum to leave the room while they were typing. When Weizenbaum suggested that he might want to store the ELIZA conversations, people immediately pointed out that this would violate people's privacy.

Users are likely to give quite personal information to large language models as well, and indeed the most common current LLM use case is for personal advice and support (Zao-Sanders, 2025). And the more human-like a system, the more users are likely to disclose private information, and yet less likely to worry about the harm of this disclosure (Ischen et al., 2019). We discussed above that pretraining data also is likely to have private information like phone numbers and addresses. This is problematic because large language models can **leak** information from their training data. That is, an adversary can extract training-data text from a language model such as a person's name, phone number, and address (Henderson et al. 2017, Carlini et al. 2021). This becomes even more problematic when large language models are trained on extremely sensitive private datasets such as electronic health records.

A related safety issue is **emotional dependence**. Reeves and Nass (1996) show that people tend to assign human characteristics to computers and interact with them in ways that are typical of human-human interactions. They interpret an utterance in the way they would if it had spoken by a human, (even though they are aware they are talking to a computer). Thus LLMs have had significant influences on people's cognitive and emotional state, leading to problems like emotional dependence on LLMs. These issues (emotional engagement and privacy) mean we need to think carefully about the impact of LLMs on the people who are interacting with them.

In addition to their ability to harm their users in these ways, LLMs may carry out additional harmful activities themselves, especially as agent-based paradigms makes it possible for language models to directly interact with the world.

Language models can also be used by malicious actors for generating text for **fraud**, phishing, propaganda, disinformation campaigns, or other socially harmful activities (Brown et al., 2020). McGuffie and Newhouse (2020) show how large language models generate text that emulates online extremists, with the risk of amplifying extremist movements and their attempt to radicalize and recruit.

And of course we already saw in Section 7.5.2 that many issues with LLM stem from using pretraining corpora scraped from the web, including harms of data consent, potential copyright violation, as well as biases in the training data that can be **amplified** by language models, just as we saw for embedding models in Chapter 5.

amplified

Finding ways to mitigate all these ethical safety issues is an important current research area in NLP. One important step is to carefully analyze the data used to pretrain large language models as a way of understanding safety issues of toxicity, discrimination, privacy, and fair use, making it extremely important that language models include **datasheets** (page **??**) or **model cards** (page **??**) giving full replicable information on the corpora used to train them. Open-source models can specify their exact training data. There are active areas of research in mitigating problems of abuse and toxicity, like detecting and responding appropriately to toxic contexts (Wolf et al. 2017, Dinan et al. 2020, Xu et al. 2020).

Value sensitive design—carefully considering possible harms in advance (Friedman et al. 2017, Friedman and Hendry 2019)— is also important; (Dinan et al., 2021) give a number of suggestions for best practices in system design. For example getting informed consent from participants, whether they are used for training, or whether they are interacting with a deployed LLM is important. Because studying these interactional properties of LLMs involves human participants, researchers also work on these issues with the Institutional Review Boards (**IRB**) at their institutions, who help protect the safety of experimental participants.

IRB

7.8 Summary

This chapter has introduced the large language model. Here's a summary of the main points that we covered:

- A **large language model** is a system that can predict the next word for previous words given a context or prefix of words, and use this prediction to **conditionally generate** text.
- There are three major architectures for language models: the **encoder**, the **decoder**, and the **encoder-decoder**. The well-known large language models used for generating text are all decoder models; we'll describe encoders in Chapter 10 and encoder-decoders in Chapter 12.
- Many NLP tasks—such as question answering and sentiment analysis— can be cast as tasks of word prediction and addressed with large language models.
- We instruct language models via a **prompt**, a text string that a user issues to a language model to get the model to do something useful by iteratively generating tokens conditioned on the prompt.
- The process of finding effective prompts for a task is known as **prompt engineering**.
- The choice of which word to generate in large language models is done by **sampling** from the distribution of possible next words.

- A common sampling approach is **temperature** sampling, which lies in between **greedy decoding** (always generate the most probable word) and **random sampling** (generate a random word according to its probability).
- Temperature sampling increases the probabilities of the high-probability words, decreases the probability of the low-probability words, and then samples from this new distribution.
- Large language models are pretrained to predict words on datasets of 100s of billions of words generally scraped from the web.
- These datasets need to be filtered for quality and balanced for domains by upsampling and downsampling.
- The pretraining algorithm relies on cross-entropy loss: minimizing the negative log probability of the true next word.
- Language models are evaluated by **perplexity**, by evaluations of accuracy on proxies for downstream tasks, like the **MMLU** question-answering dataset, and via metrics for other factors like fairness and energy use.
- Language models have numerous ethical and safety issues including hallucinations, unsafe instructions, bias, stereotypes, misinformation and propaganda, and violations of privacy and copyright.

Historical Notes

As we discussed in Chapter 3, the earliest language models were the n-gram language models developed (roughly simultaneously and independently) by Fred Jelinek and colleagues at the IBM Thomas J. Watson Research Center, and James Baker at CMU. It was the Jelinek and the IBM team who first coined the term **language model** to mean a model of the way any kind of linguistic property (grammar, semantics, discourse, speaker characteristics), influenced word sequence probabilities (Jelinek et al., 1975). They contrasted the language model with the **acoustic model** which captured acoustic/phonetic characteristics of phone sequences.

N-gram language models were very widely used over the next 40 years, across a wide variety of NLP tasks like speech recognition and machine translations, often as one of multiple components of the model. The contexts for these n-gram models grew longer, with 5-gram models used quite commonly by very efficient LM toolkits (Stolcke, 2002; Heafield, 2011).

The roots of the neural large language model lie in multiple places. One was the application in the 1990s, again in Jelinek's group at IBM Research, of **discriminative classifiers** to language models. Roni Rosenfeld in his dissertation (Rosenfeld, 1992) first applied logistic regression (under the name **maximum entropy** or **maxent** models) to language modeling in that IBM lab, and published a more fully formed version in Rosenfeld (1996). His model integrated various sorts of information in a logistic regression predictor, including n-gram information along with other features from the context, including distant n-grams and pairs of associated words called **trigger pairs**. Rosenfeld's model prefigured modern language models by being a statistical word predictor trained in a self-supervised manner simply by learning to predict upcoming words in a corpus.

Another was the first use of pretrained embeddings to model word meaning in the LSA/LSI models (Deerwester et al., 1988). Recall from the history section of

Chapter 5 that in LSA (latent semantic analysis) a term-document matrix was trained on a corpus and then singular value decomposition was applied and the first 300 dimensions were used as a vector embedding to represent words. It was Landauer et al. (1997) who first used the word "embedding". In addition to their development of the idea of pretraining and of embeddings, the LSA community also developed ways to combine LSA embeddings with n-grams in an integrated language model (Bellegarda, 1997; Coccaro and Jurafsky, 1998).

In a very influential series of papers developing the idea of **neural language models**, (Bengio et al. 2000; Bengio et al. 2003; Bengio et al. 2006), Yoshua Bengio and colleagues drew on the central ideas of both these lines of self-supervised language modeling work (the discriminatively trained word predictor, and the pretrained embeddings). Like the maxent models of Rosenfeld, Bengio's model used the next word in running text as its supervision signal. Like the LSA models, Bengio's model learned an embedding, but unlike the LSA models did it as part of the process of language modeling. The Bengio et al. (2003) model was a neural language model: a neural network that learned to predict the next word from prior words, and did so via learning embeddings as part of the prediction process.

The neural language model was extended in various ways over the years, perhaps most importantly in the form of the RNN language model of Mikolov et al. (2010) and Mikolov et al. (2011). The RNN language model was perhaps the first neural model that was accurate enough to surpass the performance of a traditional 5-gram language model.

Soon afterwards, Mikolov et al. (2013a) and Mikolov et al. (2013b) proposed to simplify the hidden layer of these neural net language models to create pretrained word2vec word embeddings.

The static embedding models like LSA and word2vec instantiated a particular model of pretraining: a representation was trained on a pretraining dataset, and then the representations could be used in further tasks. Dai and Le (2015) and Peters et al. (2018) reframed this idea by proposing models that were pretrained using a language model objective, and then the identical model could be either frozen and directly applied for language modeling or further finetuned still using a language model objective. For example ELMo used a biLSTM self-supervised on a large pretrained dataset using a language model objective, then finetuned on a domain-specific dataset, and then froze the weights and added task-specific heads. The ELMo work was particularly influential and its appearance was perhaps the moment when it became clear to the community that language models could be used as a general solution for NLP problems.

Transformers were first applied as encoder-decoders (Vaswani et al., 2017) and then to masked language modeling (Devlin et al., 2019) (as we'll see in Chapter 12 and Chapter 10). Radford et al. (2019) then showed that the transformer-based autoregressive language model GPT2 could perform zero-shot on many NLP tasks like summarization and question answering.

foundation model

The technology used for language models can also be applied to other domains and tasks, like vision, speech, and genetics. The term **foundation model** is sometimes used as a more general term for this use of large language model technology across domains and areas, when the elements we are computing over are not necessarily words. Bommasani et al. (2021) is a broad survey that sketches the opportunities and risks of foundation models, with special attention to large language models.

Anthropic. 2025. Release notes: System prompts. https://docs.anthropic.com/en/release-notes/system-prompts.

Bellegarda, J. R. 1997. A latent semantic analysis framework for large-span language modeling. *EUROSPEECH*.

Bengio, Y., R. Ducharme, and P. Vincent. 2000. A neural probabilistic language model. *NeurIPS*.

Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin. 2003. A neural probabilistic language model. *JMLR*, 3:1137–1155.

Bengio, Y., H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. 2006. Neural probabilistic language models. In *Innovations in Machine Learning*, 137–186. Springer.

Bickmore, T. W., H. Trinh, S. Olafsson, T. K. O'Leary, R. Asadi, N. M. Rickles, and R. Cruz. 2018. Patient and consumer safety risks when using conversational assistants for medical information: An observational study of Siri, Alexa, and Google Assistant. *Journal of Medical Internet Research*, 20(9):e11510.

Blodgett, S. L., S. Barocas, H. Daumé III, and H. Wallach. 2020. Language (technology) is power: A critical survey of "bias" in NLP. *ACL*.

Bommasani, R., D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. S. Chatterji, A. S. Chen, K. A. Creel, J. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. E. Gillespie, K. Goel, N. D. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. F. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. S. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. P. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. F. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. H. Roohani, C. Ruiz, J. Ryan, C. R'e, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. P. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. A. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. 2021. On the opportunities and risks of foundation models. *ArXiv*.

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. 2020. Language models are few-shot learners. *NeurIPS*, volume 33.

Carlini, N., F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, et al. 2021. Extracting training data from large language models. *30th USENIX Security Symposium (USENIX Security 21)*.

Cheng, M., E. Durmus, and D. Jurafsky. 2023. Marked personas: Using natural language prompts to measure stereotypes in language models. *ACL*.

Coccaro, N. and D. Jurafsky. 1998. Towards better integration of semantic predictors in statistical language modeling. *ICSLP*.

Dai, A. M. and Q. V. Le. 2015. Semi-supervised sequence learning. *NeurIPS*.

Deerwester, S. C., S. T. Dumais, G. W. Furnas, R. A. Harshman, T. K. Landauer, K. E. Lochbaum, and L. Streeter. 1988. Computer information retrieval using latent semantic structure: US Patent 4,839,853.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT*.

Dinan, E., G. Abercrombie, A. S. Bergman, S. Spruit, D. Hovy, Y.-L. Boureau, and V. Rieser. 2021. Anticipating safety issues in e2e conversational ai: Framework and tooling. ArXiv.

Dinan, E., A. Fan, A. Williams, J. Urbanek, D. Kiela, and J. Weston. 2020. Queens are powerful too: Mitigating gender bias in dialogue generation. *EMNLP*.

Dodge, J., S. Gururangan, D. Card, R. Schwartz, and N. A. Smith. 2019. Show your work: Improved reporting of experimental results. *EMNLP*.

Dodge, J., M. Sap, A. Marasović, W. Agnew, G. Ilharco, D. Groeneveld, M. Mitchell, and M. Gardner. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *EMNLP*.

Ethayarajh, K. and D. Jurafsky. 2020. Utility is in the eye of the user: A critique of NLP leaderboards. *EMNLP*.

Friedman, B. and D. G. Hendry. 2019. *Value Sensitive Design: Shaping Technology with Moral Imagination*. MIT Press.

Friedman, B., D. G. Hendry, and A. Borning. 2017. A survey of value sensitive design methods. *Foundations and Trends in Human-Computer Interaction*, 11(2):63–125.

Gao, L., T. Hoppe, A. Thite, S. Biderman, C. Foster, N. Nabeshima, S. Black, J. Phang, S. Presser, L. Golding, H. He, and C. Leahy. 2020. The Pile: An 800GB dataset of diverse text for language modeling. ArXiv preprint.

Gehman, S., S. Gururangan, M. Sap, Y. Choi, and N. A. Smith. 2020. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. *Findings of EMNLP*.

Gururangan, S., A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. *ACL*.

Hashimoto, T., M. Srivastava, H. Namkoong, and P. Liang. 2018. Fairness without demographics in repeated loss minimization. *ICML*.

Heafield, K. 2011. KenLM: Faster and smaller language model queries. *Workshop on Statistical Machine Translation*.

Henderson, P., J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43.

Henderson, P., X. Li, D. Jurafsky, T. Hashimoto, M. A. Lemley, and P. Liang. 2023. Foundation models and fair use. *JMLR*, 24(400):1–79.

Henderson, P., K. Sinha, N. Angelard-Gontier, N. R. Ke, G. Fried, R. Lowe, and J. Pineau. 2017. Ethical challenges in data-driven dialogue systems. *AAAI/ACM AI Ethics and Society Conference*.

Hofmann, V., P. R. Kalluri, D. Jurafsky, and S. King. 2024. Ai generates covertly racist decisions about people based on their dialect. *Nature*, 633(8028):147–154.

Ischen, C., T. Araujo, H. Voorveld, G. van Noort, and E. Smit. 2019. Privacy concerns in chatbot interactions. *International Workshop on Chatbot Research and Design*.

Jelinek, F., R. L. Mercer, and L. R. Bahl. 1975. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, IT-21(3):250–256.

Khattab, O., A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, and C. Potts. 2024. DSPy: Compiling declarative language model calls into self-improving pipelines. *ICLR*.

Kiela, D., M. Bartolo, Y. Nie, D. Kaushik, A. Geiger, Z. Wu, B. Vidgen, G. Prasad, A. Singh, P. Ringshia, Z. Ma, T. Thrush, S. Riedel, Z. Waseem, P. Stenetorp, R. Jia, M. Bansal, C. Potts, and A. Williams. 2021. Dynabench: Rethinking benchmarking in NLP. *NAACL HLT*.

Landauer, T. K., D. Laham, B. Rehder, and M. E. Schreiner. 1997. How well can passage meaning be derived without using word order? A comparison of Latent Semantic Analysis and humans. *COGSCI*.

Liang, P., R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. Cosgrove, C. D. Manning, C. Ré, D. Acosta-Navas, D. A. Hudson, E. Zelikman, E. Durmus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. Wang, K. Santhanam, L. Orr, L. Zheng, M. Yuksekgonul, M. Suzgun, N. Kim, N. Guha, N. Chatterji, O. Khattab, P. Henderson, Q. Huang, R. Chi, S. M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. Icard, T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, and Y. Koreeda. 2023. Holistic evaluation of language models. *Transactions on Machine Learning Research*.

Liu, H., J. Dacon, W. Fan, H. Liu, Z. Liu, and J. Tang. 2020. Does gender matter? Towards fairness in dialogue systems. *COLING*.

Llama Team. 2024. The llama 3 herd of models.

Longpre, S., R. Mahari, A. Lee, C. Lund, H. Oderinwale, W. Brannon, N. Saxena, N. Obeng-Marnu, T. South, C. Hunter, et al. 2024a. Consent in crisis: The rapid decline of the ai data commons. ArXiv preprint.

Longpre, S., G. Yauney, E. Reif, K. Lee, A. Roberts, B. Zoph, D. Zhou, J. Wei, K. Robinson, D. Mimno, and D. Ippolito. 2024b. A pretrainer's guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity. *NAACL HLT*.

McGuffie, K. and A. Newhouse. 2020. The radicalization risks of GPT-3 and advanced neural language models. ArXiv preprint arXiv:2009.06807.

Mikolov, T., K. Chen, G. S. Corrado, and J. Dean. 2013a. Efficient estimation of word representations in vector space. *ICLR 2013*.

Mikolov, T., M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. 2010. Recurrent neural network based language model. *INTERSPEECH*.

Mikolov, T., S. Kombrink, L. Burget, J. H. Černockỳ, and S. Khudanpur. 2011. Extensions of recurrent neural network language model. *ICASSP*.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. 2013b. Distributed representations of words and phrases and their compositionality. *NeurIPS*.

Miller, G. A. and J. A. Selfridge. 1950. Verbal context and the recall of meaningful material. *American Journal of Psychology*, 63:176–185.

Min, S., X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? *EMNLP*.

Nadeem, M., A. Bethke, and S. Reddy. 2021. StereoSet: Measuring stereotypical bias in pretrained language models. *ACL*.

Neff, G. and P. Nagy. 2016. Talking to bots: Symbiotic agency and the case of Tay. *International Journal of Communication*, 10:4915–4931.

Parrish, A., A. Chen, N. Nangia, V. Padmakumar, J. Phang, J. Thompson, P. M. Htut, and S. Bowman. 2022. BBQ: A hand-built bias benchmark for question answering. *Findings of ACL 2022*.

Peters, M., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. 2018. Deep contextualized word representations. *NAACL HLT*.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. 2019. Language models are unsupervised multitask learners. OpenAI tech report.

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140):1–67.

Rawls, J. 2001. *Justice as fairness: A restatement*. Harvard University Press.

Reeves, B. and C. Nass. 1996. *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Cambridge University Press.

Rosenfeld, R. 1992. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. Ph.D. thesis, Carnegie Mellon University.

Rosenfeld, R. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, 10:187–228.

Sagawa, S., P. W. Koh, T. B. Hashimoto, and P. Liang. 2020. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *ICLR*.

Shannon, C. E. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423. Continued in the following volume.

Sheng, E., K.-W. Chang, P. Natarajan, and N. Peng. 2019. The woman worked as a babysitter: On biases in language generation. *EMNLP*.

Soldaini, L., R. Kinney, A. Bhagia, D. Schwenk, D. Atkinson, R. Authur, B. Bogin, K. Chandu, J. Dumas, Y. Elazar, V. Hofmann, A. H. Jha, S. Kumar, L. Lucy, X. Lyu, N. Lambert, I. Magnusson, J. Morrison, N. Muennighoff, A. Naik, C. Nam, M. E. Peters, A. Ravichander, K. Richardson, Z. Shen, E. Strubell, N. Subramani, O. Tafjord, P. Walsh, L. Zettlemoyer, N. A. Smith, H. Hajishirzi, I. Beltagy, D. Groeneveld, J. Dodge, and K. Lo. 2024. Dolma: An open corpus of three trillion tokens for language model pretraining research. ArXiv preprint.

Stolcke, A. 2002. SRILM – an extensible language modeling toolkit. *ICSLP*.

Strubell, E., A. Ganesh, and A. McCallum. 2019. Energy and policy considerations for deep learning in NLP. *ACL*.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017. Attention is all you need. *NeurIPS*.

Webson, A. and E. Pavlick. 2022. Do prompt-based models really understand the meaning of their prompts? *NAACL HLT*.

Weizenbaum, J. 1966. ELIZA – A computer program for the study of natural language communication between man and machine. *CACM*, 9(1):36–45.

Wolf, M. J., K. W. Miller, and F. S. Grodzinsky. 2017. Why we should have seen that coming: Comments on Microsoft's Tay "experiment," and wider implications. *The ORBIT Journal*, 1(2):1–12.

Xu, A., E. Pathak, E. Wallace, S. Gururangan, M. Sap, and D. Klein. 2021. Detoxifying language models risks marginalizing minority voices. *NAACL HLT*.

Xu, J., D. Ju, M. Li, Y.-L. Boureau, J. Weston, and E. Dinan. 2020. Recipes for safety in open-domain chatbots. ArXiv preprint arXiv:2010.07079.

Zao-Sanders, M. 2025. How People Are Really Using Gen AI in 2025 — hbr.org. `https://hbr.org/2025/04/how-people-are-really-using-gen-ai-in-2025`. [Accessed 02-05-2025].

Zhang, A. K., K. Klyman, Y. Mai, Y. Levine, Y. Zhang, R. Bommasani, and P. Liang. 2025. Language model developers should report train-test overlap. *ICML*.