

Deep Dive into LLMs like ChatGPT



How LLMs work

by Andrej Karpathy

transcription by André Cabannes



Eagle's Beak Press

This is a transcription of the video available at <https://www.youtube.com/watch?v=7xTGNNLPyMI&t=2s>, published in February 2025.¹

¹Prepared by <https://lapasserelle.com/webmaster.html>

It is a lightly edited version of the text of the video for better readability, with added illustrations from the video.

Footnotes by the transcriber.

Titles published by Eagle's Beak Press are featured at <https://lapasserelle.com/index-eng.php>.

Table of content

1 Introduction	2
2 Pre-training data (internet)	3
3 Tokenization	9
4 Neural network I/O	17
5 Neural network internals	21
6 Inference	27
7 GPT-2: training and inference	31
8 Llama 3.1 base model inference	42
9 From pre-training to post-training	61
10 Post-training data (conversations)	67
11 Hallucinations, tool use, knowledge/working memory	82
12 Knowledge of self	102
13 Models need tokens to think	107
14 Tokenization revisited: models struggle with spelling and counting	122
15 Recap: from base model to assistant	128
16 From supervised fine-tuning to reinforcement learning	130
17 Reinforcement learning	139
18 DeepSeek-R1	145
19 AlphaGo	162
20 Reinforcement learning from human feedback	168
21 Upside and downside of RLHF	176
22 Preview of things to come	185
23 Keeping track of LLMs	189
24 Where to find LLMs	193
25 Grand summary	197

1 Introduction

Hi everyone,

I wanted to make this video for a while. It is a comprehensive, general audience introduction to large language models like ChatGPT.¹

What I hope to achieve in this video is to give you some kind of mental model for thinking through what this tool is. It is obviously magical and amazing in some respects: it's really good at some things, not very good at others. And there are also a lot of sharp edges to be aware of.

So what is behind this text box ? (fig. 1)

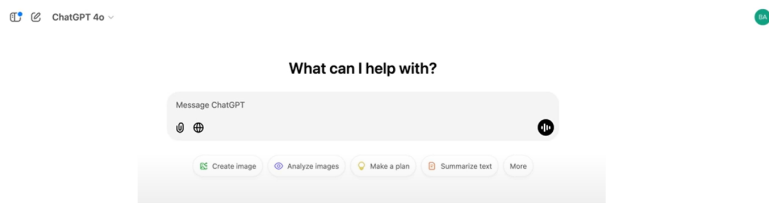


Figure 1: ChatGPT text box to enter a question, a text, a prompt, an image, or anything to trigger an LLM response.

You can put anything in the box and press enter.

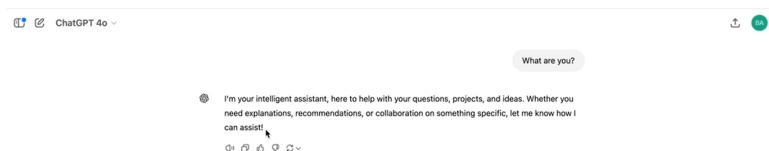


Figure 2: ChatGPT 4o answer to the prompt "What are you ?".

¹While speaking Andrej Karpathy uses a large white board with illustrations. It can be obtained using <https://excalidraw.com/> and opening the file <https://lapasserelle.com/documents/karpathy.txt>

But what should we be putting there, and what are these words generated back? (fig. 2) How does this work. What are you talking to, exactly?

I'm hoping to get at all those topics in this video. We're going to go through the entire pipeline of how this stuff is built. But I'll keep everything accessible to a general audience.

Let's take a look first at how you build something like ChatGPT. Along the way, I'm going to talk about some of the cognitive and psychological implications of these tools.

00:01:00²

2 Pre-training data (internet)

Let's build ChatGPT. There are going to be multiple stages arranged sequentially.

The first stage is called the *pre-training stage*. And the first step of the pre-training stage is to download and process the internet. fig. 3.

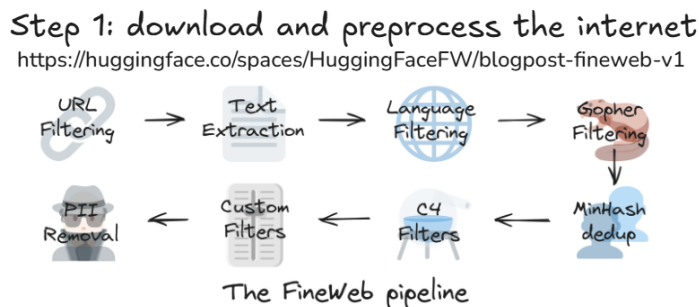


Figure 3: Step 1 of the *pre-training stage*: download and preprocess the internet.

²We occasionally indicate the approximate timestamp in the video that corresponds to our position in the text. At this point, we are at 00:01:00 in the video.

To get a sense of what this looks like, I recommend looking at <https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1>

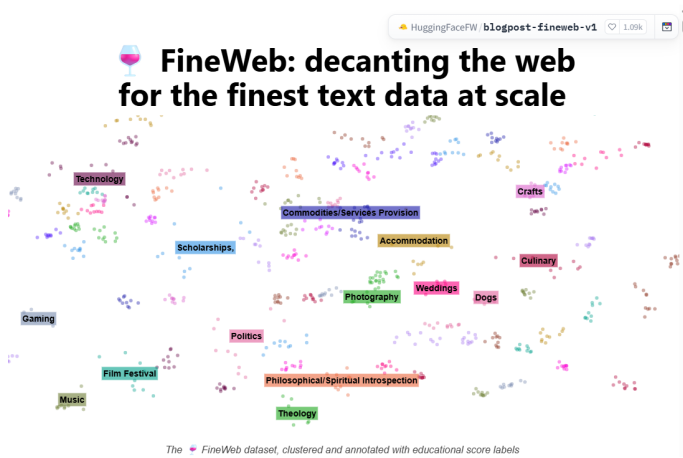


Figure 4: `huggingface.co` main page.

This company, called Hugging Face, collected, created, and curated this data set called Fineweb. They go into a lot of detail in this blog post about how they constructed the Fineweb data set.

All the major LLM providers, like OpenAI, Anthropic, and Google, and so on, have something equivalent internally, something like the Fineweb data set.

What we are trying to achieve here is to get a ton of text from the internet, from publicly available sources. We're trying to have a huge quantity of very high-quality documents. We also want a very large diversity of documents, because we want to have a lot of knowledge inside these models. So we want a large diversity of high-quality documents, and we want many, many of them.

Achieving this is quite complicated, and as you can see here, it takes multiple stages to do it well, figure 3.

00:02:00

Let's take a brief look at some of these stages.

For now, I'd just like to note that, for example, the Fineweb data set — which is fairly representative of what you would see in a production-grade application — actually ends up being only about 44 terabytes of disk space. You can get a USB stick for a terabyte very easily, and I think this could almost fit on a single hard drive nowadays. So this is not a huge amount of data at the end of the day.

Even though the internet is very, very large, we're only working with text. We're also filtering it aggressively. So we end up with about 44 terabytes in this example.

Let's take a look at what this data looks like and what some of these stages also are. The starting point for a lot of these efforts, and something that contributes most of the data by the end of it, is data from *Common Crawl*.

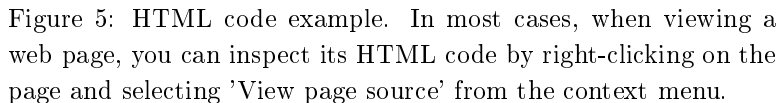
Common Crawl is an organization that has been scouring the internet since 2007. As of 2024, Common Crawl has indexed 2.7 billion web pages. They have all these crawlers going around the internet. What you do is you start with a few seed web pages and then you follow all the links. You just keep following links and keep indexing all the information. You end up with a ton of data from the internet over time. This is usually the starting point for a lot of these efforts.

Now, the Common Crawl data is quite raw and needs to be filtered in many different ways. In figure 3, Hugging Face shows a little bit the kind of processing that happens in these stages.

The first step is called *URL filtering*. There are blacklists of URLs or domains that you don't want to be getting data from. For instance the site <https://dsi.ut-capitole.fr/blacklists/> shows blacklisted sites. It usually includes things like malware websites, spam websites, marketing websites, racist websites, adult sites, and things like that. So

00:04:00

The second step is *text extraction*. You have to remember that all these web pages are just raw HTML — the web pages that are being saved by these crawlers. For instance, when I go to inspect the Hugging Face blog page, this is what the raw HTML actually looks like.



You'll notice that it's got all this markup, like lists (`<ul> ... </ul>` with each item in `<li> ... </li>`), and stuff like that. There's also CSS code. This is computer code for constructing the HTML web pages.

What we really need is just the visible text displayed by the browser—not the underlying HTML code with its display tags. We are interested only in "the meat", not "the bones" or skeleton. We are interested in the *main content* of the web page: navigation menus and other extraneous elements are also discarded. So, extracting just the relevant content from web pages requires substantial filtering and processing to isolate high-quality, interesting and useful text.

The next stage is *language filtering*. For example, Fineweb filters, using a language classifier, try to guess what language every single web page is in. Then they only keep web pages that have more than, say, 65% English.

You can get a sense that this is a design decision that different companies can make for themselves: what fraction of all different types of languages are we going to include in our data set? Because if we filter out, for example, all of the Spanish, then you might imagine that our model later will not be very good at Spanish because it has just never seen much data in that language.

Different companies can focus on multilingual performance to different degrees. Fineweb is quite focused on English, and their language model, if they end up training one later, will be very good at English, but may not be very good at other languages.

After language filtering, there are *a few other filtering steps*, such as filtering duplication and things like that, finishing with, for example, "PII" removal. This is *personally identifiable information*: addresses, Social Security numbers, and so forth. You'd try to detect them, and you'd try to filter out those kinds of web pages from the data set as well.

00:06:00

In summary, there are lots of stages in the pre-processing (the filtering to get a clean corpus of high-quality texts), fig. 3.

I won't go into full detail, but it is a fairly extensive part of the pre-processing. And you end up with, for example, the Fineweb data set.

When you click on it (and get to <https://huggingface.co/datasets/HuggingFaceFW/fineweb>), you can see some examples of what this actually ends up looking like, fig. 6. Anyone can download this on the Hugging Face web page.

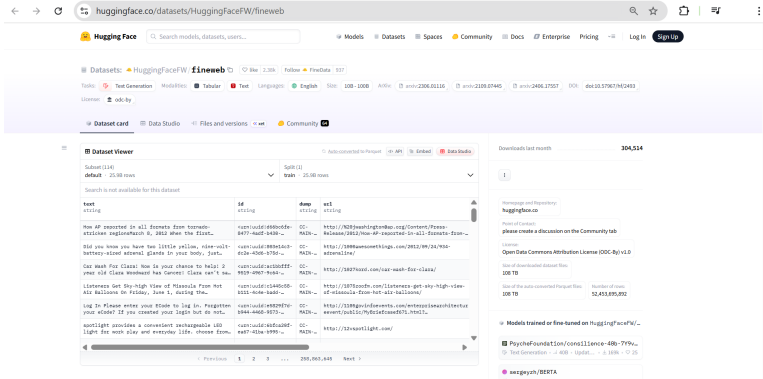


Figure 6: Content of Fineweb.

Here are some examples of the final text that ends up in the training set. Here is some article about tornadoes in 2012. There were some tornadoes in 2012 and the article describes what happened. The next article is something about... did you know you have two little yellow nine-volt battery-sized adrenal glands in your body? Okay, this is some kind of odd medical article.

Just think of all these as web pages on the Internet, filtered just for the text (the "meat") in various ways. And now we have a ton of text — 44 terabytes of it.

00:07:00

We are now at the starting point for the next step of this stage [the pre-training stage].

I wanted to give you an intuitive sense of where we are right now. So I took the first 200 web pages her (bottom left of fig. 6) — and remember we have tons of them.

I just took all that text, and I put it all together, concatenated it, and this is what we end up with. We just get this, fig. 7: raw text, raw internet text, and there's a ton of it, even in these 200 web pages.

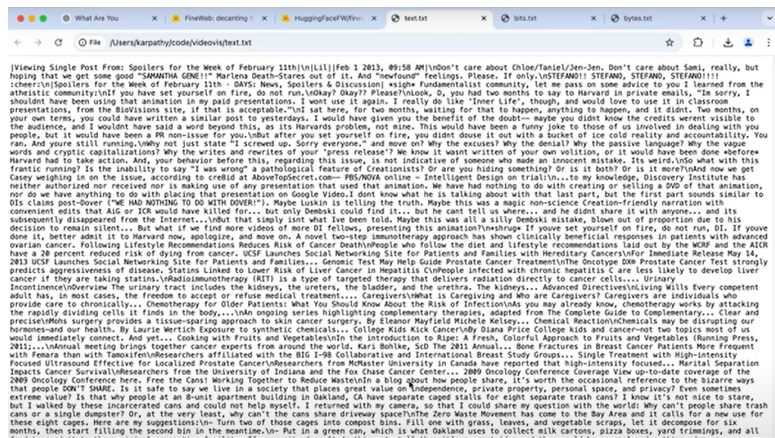


Figure 7: View of the beginning of 44 terabytes of text provided by Fineweb.

I can continue zooming out here [looking from farther away]: we just have this massive tapestry of text data. And this text data has all these patterns.

What we want to do now is to start training neural networks on this data, so the neural networks can internalize and model how this text flows.

We have this giant "texture of text", and now we want to get neural networks that mimic it. What we mean by that will become clear as we go on.

00:07:47

3 Tokenization

Before we plug text into neural networks, we have to decide how we're going to represent this text and how we're going to feed it in our neural network.

The way our technology works for these neural nets is that they expect a one-dimensional sequence of symbols. They want a finite set of possible symbols. So we have to decide

what the symbols are and then we have to represent our data as a one-dimensional sequence of those symbols.³

Right now we have a one-dimensional sequence of text, fig. 7. It is a one-dimensional sequence even though, on my monitor, of course, it's laid out in a two-dimensional way. But it goes from left to right and top to bottom. So it's a one-dimensional sequence of text.

This being computers, of course, there is an underlying representation here. If I use UTF-8 encoding [of letters and other characters] for the text, then I can get the raw bits that correspond to this text in the computer, and that's what it looks like.

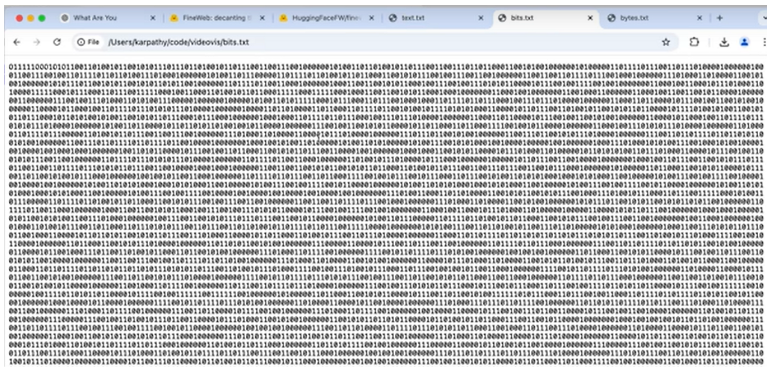


Figure 8: Text of figure 7 shown in UTF-8 encoding. It is now a (much longer) sequence of zeros and ones.

In figure 7, the first line reads:

|Viewing Single Post From: Spoilers for the Week of...

The very first vertical bar has a UTF-8 encoding. In hexadecimal, its UTF-8 byte is 7c. In binary, this corresponds to the first eight bits shown at the top of figure 8: 01111100.

³Of course, we could represent the text using just the 26 letters of the alphabet along with a few punctuation marks. However, that's not how the vast amount of text — such as the excerpt shown in Figure 7 — will actually be encoded for neural network processing.

Next, the letter V is encoded as 56 in hex and 01010110 in binary.

The letter i is encoded as 69 in hex and 01101001 in binary.

Thus, the top line in figure 8 begins with:

011111000101011001101001...

What is this thing, in figure 8 ? It is the representation that we are looking for, in a certain sense. We have exactly two possible symbols: 0 and 1. And we have a very long sequence of them. The 200 web pages correspond to a long sequence, and the 44 terabytes to an even longer sequence.

When we select a fragment in our complete data set (of 44 terabytes), it is going to be a long sequence of symbols (at this stage, of zeroes and ones). It turns out that this sequence length is actually going to be a very finite and precious resource in our neural network.

Now, we actually don't want extremely long sequences of just two symbols. Instead, we want to trade off the symbol size of this vocabulary, as we call it, and the resulting sequence length. So we don't want just two symbols and extremely long sequences. We're going to use more symbols and shorter sequences.

One naive way of compressing or decreasing the length of our sequence here, fig. 8, is to consider some group of consecutive bits — for example, eight bits — and group them into a single, what's called a *byte*.

00:10:00

Because these bits are either on or off (i.e. 0 or 1), if we take a group of eight of them, there turn out to be only 256 possible combinations [from 0 to 255] of how these bits could be on or off. Therefore, we can represent this sequence as a sequence of bytes instead, figure 9.

This sequence of bytes will be eight times shorter. But now we have 256 possible symbols. Every number here goes from 0 to 255.

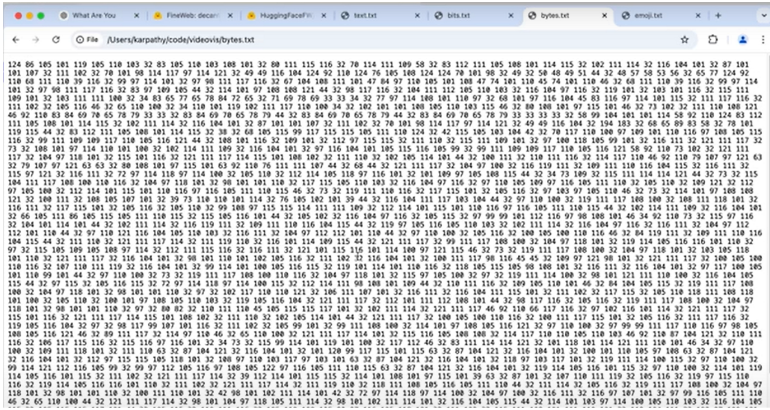


Figure 9: Text of figure 7 shown now using 256 different bytes (from 0 to 255). We should think of them as just 256 distinct symbols – we could use 256 emojis to stress this point.⁴

I really encourage you to think of these not as numbers but as *unique IDs* or *unique symbols*. To stress this point, we could even replace every one of 256 numbers (from 0 to 255) with a unique emoji. The reader can imagine a long text written with an "alphabet" of 256 emojis. You can think of it that way.

In production, for state-of-the-art large language models, you actually want to go even beyond this. You want to continue to shrink the length of the sequence—because again, it is a precious resource — in return for richness in your vocabulary.

The way this is done is by running what's called *the byte pair encoding algorithm*. The way this works is we're looking for consecutive bytes or symbols that are very common.

For example, it turns out that the sequence 116 followed by 32 is quite common and occurs very frequently. So we're

⁴Similarly, in Sudoku, the digits 1 through 9 are used purely as nine distinct symbols, without any arithmetic significance. For further discussion, see André Cabannes, *Middle School Mathematics, Volume 1: 6th and 7th Grades*, Eagle's Beak Press, 2024, p. 25.

going to group this pair into a new symbol. We're going to mint a symbol with an ID 256⁵ and we're going to rewrite every single pair 116-32 with this new symbol. And then we can iterate this algorithm as many times as we wish. Each time when we mint a new symbol, we're decreasing the length and increasing the size of the collection of symbols.

In practice, it turns out that a pretty good setting for the size of the basic vocabulary is about 100,000 possible symbols.

In particular, GPT-4 uses 100,277 symbols.

This process of converting raw text into these symbols, or as we call them *tokens*, is the process called *tokenization*.

00:12:10

Let's now take a look at how GPT-4 performs tokenization converting from text to tokens, and from tokens back to text, and what this actually looks like.

One website I like to use to explore these token representations is called Tiktokenizer (<https://www.tiktokenizer.app/>).

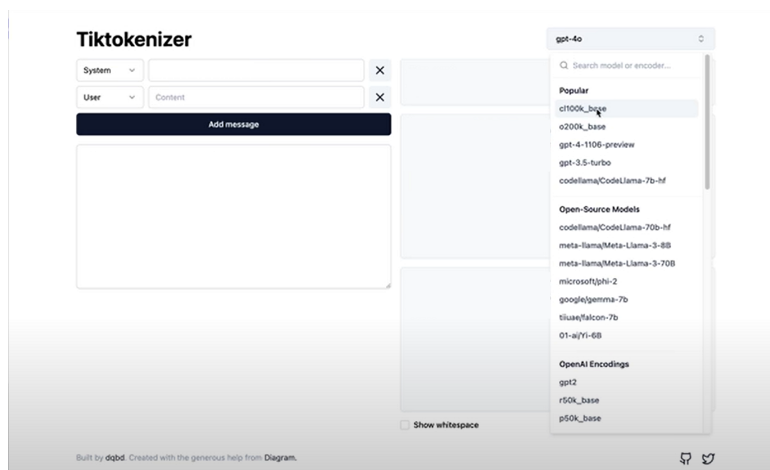


Figure 10: Tiktokenizer. Selection of the encoding on the right.

⁵Because that's the next available ID.

Go to the dropdown menu and select “CL100K_base,” which is the tokenizer for the GPT-4 base model. On the left, you can enter text, and the tool shows you how the text is tokenized. For example, if you type “hello world”, it will be split into exactly two tokens: the token “hello”, which has the ID 15339, and the token “ world” (including the space in front), which has the ID 1917, figure 11.

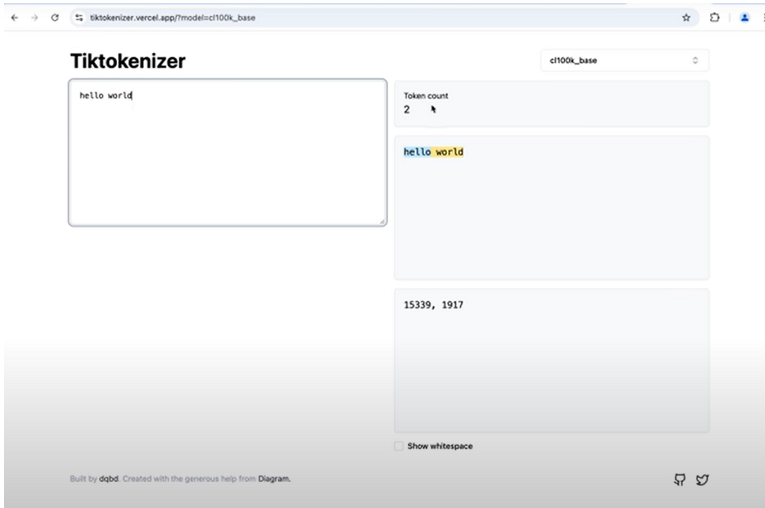


Figure 11: Encoding of "hello world" using CL100K_base.

Now, if I join the two words and write "helloworld", for example, I’m going to get two tokens again, but it’s the token ‘h’ (ID 71) followed by ‘elloworld’ without the ‘h’ (ID 96392). If I put in two spaces between “hello” and “world,” it’s yet a different tokenization; there’s a new token, 220. The sequence of tokens becomes 15339, 220, 1917.⁶

You can play around with this and see what happens. Also, keep in mind this is case sensitive — if it’s a capital “H,” you’ll

⁶Pay attention to the fact that 220 is the token for one space alone, and 1917, as we saw, is the token for " world" (with a space in front).

get something else. Or if it's "HELLO WORLD", this ends up being the three tokens 51812, 1623, 51991, corresponding to "HEL", "LO", " WORLD". You can play with this and get an intuitive sense of how tokens and tokenization work.

We shall come back to tokenization later in the video. For now, I just wanted to show you the website <https://www.tiktokenizer.app/>, what this text entered in the box on the left, fig. 11, becomes once processed.

For example, if I take one line from fig. 7, this is how GPT-4 will see it.

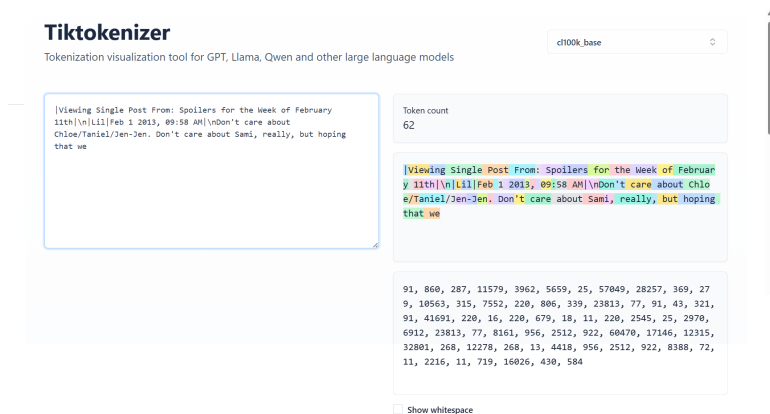


Figure 12: Tokenization of first line of text in fig. 7.

The text will become a sequence of tokens of length 62. The original is entered on the left. At the top right, you can see how it is divided into chunks. The bottom right displays the corresponding sequence of numbers (though you are encouraged to think of these as pure symbols).

As said, in ChatGPT 4 there are 100,277 possible symbols.

We now have a one-dimensional sequence of those symbols.⁷

⁷The last token is different from the video, because A. Karpathy cut the text at "that w", whereas we cut it at "that we".

00:14:30

What we have done is take the sequence of text that we have in the data set, at the beginning of fig. 7, page 9, and we have re-represented it, using our tokenizer, as a sequence of tokens. This is what the whole text of fig. 7 looks like.

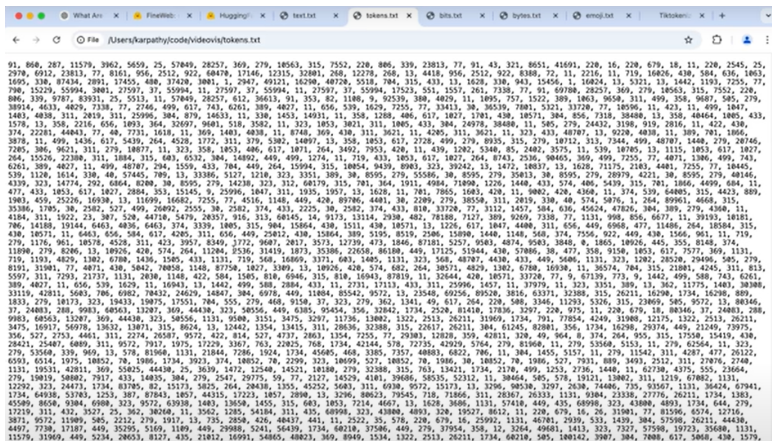


Figure 13: Tokenization of whole text shown in fig. 7.⁸

When we go back to the Fineweb data set, they mention that it’s not only 44 terabytes of disk space, it’s also an approximately 15 trillion token sequence in the data set.

In figure 13, we're seeing just some of the first tokens — maybe one or two or three, or a couple thousand tokens of this data set — but there are 15 trillion to keep in mind.

And keep also in mind that all of these represent little chunks of text; they're like atoms in these sequences. The numbers themselves don't mean anything — they're just unique IDs.

⁸It is not the same as figure 9, because in fig. 9 we simply transformed fig. 8 with 256 symbols, whereas now we have tokenized the original text using 100 277 symbols.

00:15:15

4 Neural network I/O

Now comes the fun part, that is the training of the neural network. This is where a lot of the heavy lifting happens computationally – when you’re training neural networks.

In this step we *model* the statistical relationships of how these tokens follow each other in the sequence.⁹

We go into the data and take windows of tokens — we take a window of tokens from this data, fairly randomly — and the window length can range anywhere from zero tokens actually, all the way up to some maximum size that we decide on.

In practice you might see a window of, say, 8000 tokens.

In principle, we can use arbitrary window lengths of tokens, but processing very long window sequences would be extremely computationally expensive. Therefore we just decide that 8000 is a good number, or 4000 or 16000, and we crop it there.

In the example below, to keep things simple, I’m going to take the first four tokens just to illustrate the technique. It is the first four tokens of our text (fig. 7). They have IDs 91, 860, 287 and 11579.

The correspondence is this:

- 91 = "|", that is the vertical bar
- 860 = "View"
- 287 = "ing"
- 11579 = " Single", that is space followed by the word Single

⁹We are going to model the relationship between a given token and the collection of tokens that precede it. In its essence, it is very similar to a regression in statistics where we figure out the best formula linking a few independent variables $x_1, x_2, \dots, x_p$ and a dependent variable y , when we have at our disposal a large set of observed pairs (X_n, y_n) .

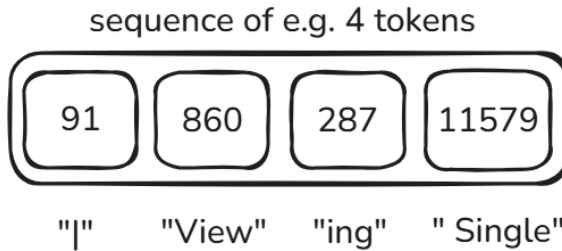


Figure 14: Sequence of four tokens. We want to predict the token that comes next. Of course here we know which token it is (it is the token 3962, see fig. 12 and 13, or better go yourself to Tiktokenizer); that will help us find a good predictive function.

Our goal is to build a function that always predicts the token that comes next, when we give it as an input a sequence of preceding tokens.

Of course, in this case, we know that the next token is 3962. Therefore, once we have found a function¹⁰ that performs well across the entire data set during training, it should predict 3962 when given the preceding four tokens — 91, 860, 287, and 11579 — as input.

These four tokens are called the *context*. We will feed them into our neural network. They will be the *input*, and we expect the *output*, once our neural network is properly trained, to be the correct token that we already know.

I'll go briefly, in a moment, into the details of what's inside the neural network. For now, it's important to understand the input and output of the neural net: the input is sequences of tokens of variable length, anywhere between zero and some maximum size, like 8,000; the output is a prediction for what comes next.

Because our vocabulary has 100,277 possible tokens, the neural network is going to output exactly that many numbers,

¹⁰This step represents the most labor – and resource – intensive part of the pre-training stage for our large language model.

and each one corresponds to the probability of that token coming next in the sequence. So it's making guesses about what comes next.

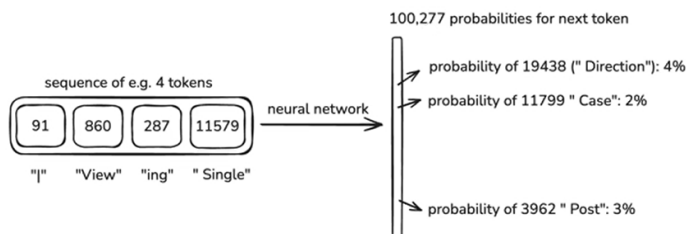


Figure 15: Each of the 100 277 tokens has a certain probability of being the next one in the sequence.

Once the neural network has been properly trained, typically after several months of intensive computation it will, for any input context, produce a probability for each of the 100,277 tokens to be the next token. Thus, it outputs a vector of 100,277 positive numbers that sum to one, figure 15.

In the beginning of its training, the neural network is randomly initialized. We're going to see in a little while what that means – but in truth it doesn't concern us too much, because I won't spend much time on how the neural net is built inside. But initially it's a random transformation. So these probabilities in the very beginning of the training are also going to be kind of random.

In figure 15, I give three examples but keep in mind that there's $\approx 100,000$ numbers in the vertical vector. For instance, the neural network says that the probability of the next token be " Direction" is 4%; the probability that it be the token 11799 (" Case") is 2%; and then the probability of 3962 (which is " Post") is 3%.

In fact, of course, we've sampled this window from our data set, so we already know what comes next. We know that the correct answer, that is what actually comes next in the data set in the sequence is 3962.

On the construction side of the neural net, we now have a mathematical process – during the construction of the network – for updating it, that is for tuning it. We use the data set and train the neural net on a huge amount of trials where we actually know what it should guess, to make it sharper and sharper.

We'll go into a little more detail soon, but basically, we know that we want this probability of 3% (corresponding to token 3962) to be higher, and we want the probabilities for all the other tokens to be lower.

*We have a way of mathematically calculating how to adjust and update the neural network — after each trial or batch of trials — so that the correct answer receives a slightly higher probability.*¹¹

In other words, if I update the neural network now, using the fact that "|" "View" "ing" "Single" should lead to guessing "Post", next time I feed this particular sequence of four tokens into the neural network, it will be slightly adjusted and will say: Okay, "Post" is maybe 4%, "Case" is now maybe 1%, and "Direction" could become 2%, or something like that.

So we have a way of nudging — or slightly updating — the neural net to give a higher probability to the correct token that comes next in the sequence.

Now, you have to remember that this process doesn't only happen for the fifth token in fig.15, with the four fed into the network to predict this next one. It happens for all the tokens in the entire data set – selecting a context and trying to guess the token that comes next – at the same time.

In practice, we sample little windows — little batches of windows — and at every single one of these tokens, we want to

¹¹One can regard the neural network as a black box that receives inputs and produces outputs. To guarantee satisfactory outputs, various internal elements called parameters must be adjusted iteratively. This adjustment process involves a very large number of trials. We will only give a sketch of the mathematics, since this is not the subject of the video. And to provide truly serious explanations (not just "Scientific American" style), another three-hour video would be needed.

adjust our neural network so that the probability of the right token becomes slightly higher. This all happens in parallel, in large batches of these tokens.

This is the process of training the neural network — it's a sequence of updating it so that its predictions match up the statistics of what actually happens in your training set. And its probabilities become consistent with the statistical patterns of how these tokens follow each other in the data.

00:20:11

5 Neural network internals

Let's now briefly look at the internals of these neural networks, just to give you a sense of what's inside.

As I mentioned, we have inputs that are sequences of tokens: in this case, this is four input tokens, but it could be anywhere from zero up to, say, 8000 tokens.

In principle this can be an infinite number of tokens, but it would just be too computationally expensive to process an infinite number of tokens. So we just crop it at a certain length and that becomes the maximum context length of that model. These inputs x_i 's are mixed up in a giant mathematical expression together with the parameters or the *weights* of the neural network.

In fig. 16, I show an example with four tokens and six parameters and their settings, but in practice, modern neural networks have hundreds of thousands of possible tokens and billions of parameters.

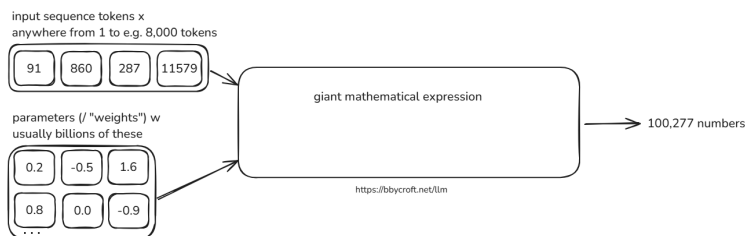


Figure 16: Input tokens, parameters, and math formula.

At the outset of the construction of the neural network, the parameters are set completely at random. With a random setting of parameters, you might expect that the neural network would make random predictions and it does: in the beginning the network makes totally random predictions.

But through the process of iteratively updating the network – we call that process training a neural network – the parameters get adjusted such that the outputs of our neural network become consistent with the patterns seen in our training set.

You can think of these parameters as the knobs on a DJ set or the faders on a sound mixing console. As you adjust these knobs or faders, you obtain different predictions for every possible input token sequence.

*Training a neural network just means discovering a setting of parameters that seems to be consistent with the statistics of the training set.*¹²

00:22:00

Let me just give you an example of what this giant mathematical expression looks like just to give you a sense. And modern networks are massive expressions with trillions of terms probably. But let me just show you a simple example here.

¹²More mathematically speaking:

1. We have a large number of sequences S_i for which we know the following token y_i .
2. Each sequence S_i is a collection of x_{ij} (for example, 8000 x_{ij}).
3. We want to find a function f , reasonably easy to compute, which for any input sequence S produces – or shows as *likely* – an output y that is correct for all known pairs (S_i, y_i) .
4. The function f will be a combination of the tokens in the sequence with weights, which serve as the parameters of the function.
5. We will adjust these weights to satisfy the condition in point 3 as well as possible.
6. This is done using mathematical minimization techniques, a well-known one being the gradient descent along a cost function.

It would look something like this, fig. 17. These are the kinds of expressions – just to show you that it’s not very scary.

We have inputs x like x_1 x_2 – in this example two inputs x_i ’s – and they are combined with the weights of the network $w_0, w_1, w_2, w_3, \dots$ up to w_{12} .

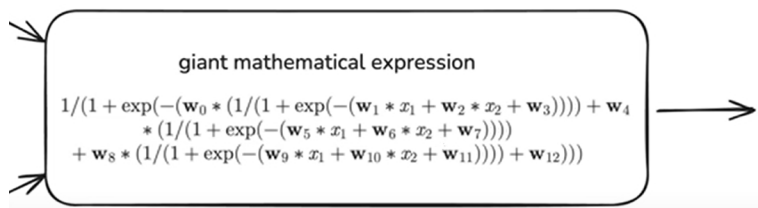


Figure 17: Example of mathematical expression.¹³

This mixing is simple things like multiplication, addition, exponentiation, division, etc.

It is the subject of neural network architecture research to design effective mathematical expressions that have a lot of kind of convenient characteristics. They are expressive, they’re optimizable, they’re parallelizable, etc.

At the end of the day these are not complex expressions. They combine the inputs with the parameters to make predictions. And we’re optimizing the parameters of the neural network so that the predictions come out consistent with the training set.

Now I would like to show you an actual production-grade example of what these neural networks look like. For that I encourage you to go to this website that has a very nice visualization of one of these networks: <https://nano-gpt.com/conversation/new> (et aussi <https://bbycroft.net/llm>).

This is what you will find on nano-gpt website:

¹³Once the network has been trained, all the parameters are fixed. Although the exact number is confidential, reliable sources estimate that ChatGPT-4 has approximately 1.8 trillion parameters.

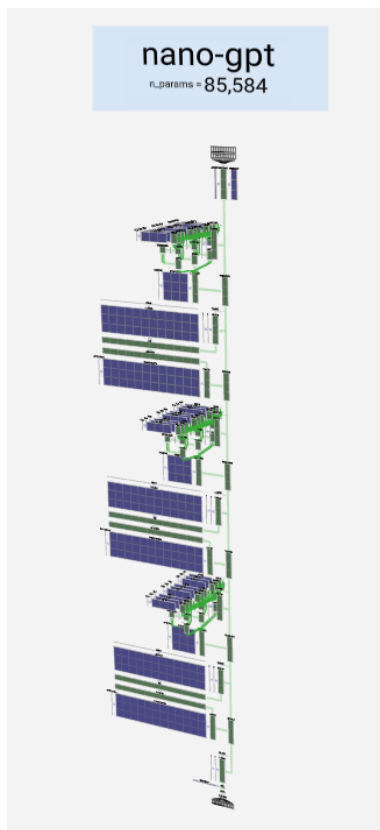


Figure 18: NanoGPT.

This neural network, which is used in production settings, has a special kind of structure. It belongs to the class of networks called *transformers*.¹⁴ And this particular one, as an example, has roughly 85 000 parameters.

At the top of the model, we start with the input, which is a sequence of tokens.

¹⁴The author consistently calls the NanoGPT a "Transformer" (or "GPT" for Generative Pretrained Transformer), following the paper "Attention is All You Need" and the standard neural architecture used by GPT-2 and GPT-3 models.

Information flows (downward in the figure) through the neural network, layer by layer, until it reaches the output. At the output layer, we obtain values known as logits. By applying the softmax function to these logits, we convert them into probabilities. They represent the model's predictions for the next token — it tells us which token is most likely to come next in the sequence.

In the middle of the nano GPT transformer (fig. 18), there is a sequence of transformations. All the intermediate values produced inside this "calculator" contribute to predicting what comes next. Added into what is known as a *distributed representation*. In other words, each possible token is represented in the neural network by a vector.

At the input level, we embed the tokens, and their corresponding values are propagated through the diagram. In NanoGPT, for instance, we see three transformer blocks. Each element within these blocks performs a straightforward mathematical operation — such as layer normalization, matrix multiplication, or softmax.

Within each transformer block, information flows first through the attention mechanism, then continues into the multi-layer perceptron block, and so forth. All of the numbers produced in these blocks are intermediate values in the overall computation.¹⁵

These values can be thought of as analogous to the firing rates of artificial neurons. However, it's important not to take the analogy too far, as these artificial neurons are much simpler than the ones found in biological brains. Biological neurons are highly complex, dynamic systems, capable of retaining memory and adapting to new information.

In contrast, in the expression of fig. 17, there is no notion of memory; it's just a fixed mathematical function mapping input directly to output in a static, predictable manner.

¹⁵Here we give popular science-type explanations that any interested reader should definitely replace with a specialized book or video.

So while these are very simple neurons compared to their biological counterparts, you can still loosely think of them as forming a synthetic piece of brain tissue, if that analogy helps.

Ultimately, information flows through these layers of artificial neurons, step by step, until the model produces its final predictions.

I don't intend to dwell too much on the precise mathematical details of all these transformations. It is not the objective of the video.

And honestly I don't think it's that important to get into the heavy-duty math of neural networks.¹⁶

*What's really important to understand is that this is a mathematical function. It is parameterized by some fixed set of parameters—say, 85 000 of them. It's a way of transforming inputs into outputs. As we adjust the parameters, we get different kinds of predictions. We need to find a good setting for these parameters so that the predictions match the patterns seen in the training set.*¹⁷

So that's the transformer.

¹⁶YouTube chain 3Blue1Brown of Grant Sanderson gives an example that is reasonably easy to follow : <https://www.youtube.com/watch?v=aircAruvnKk>.

It explains how a neural network is used to read handwritten digits. It is essentially the system built by Yann LeCun (born 1960) with his pioneering neural network for handwritten digit recognition — the first practical convolutional neural network (CNN) — in 1989, while working at AT&T Bell Laboratories. His collaborators included Léon Bottou, Yoshua Bengio, and later Patrick Haffner.

Grant Sanderson, of 3Blue1Brown, still rests here and there on vulgarization instead of clear maths, nonetheless he usefully complements Andrej Karpathy's presentation.

¹⁷We do this using a technique from mathematics called *optimization*: we minimize a cost function using gradient descent.

A website recommended by Perplexity to understand the math is <https://devot.team/blog/how-to-make-an-ai>.

00:26:00

6 Inference

I've described the internals of the neural network and explained the training process. Now, I want to discuss one more major stage of working with these networks—the stage called *inference*.

In inference we generate new data from the model.

To do that we want to see what kind of patterns the model has internalized in the parameters of its network.

To generate from the model is relatively straightforward. We start with some tokens that are your prefix you want to start with.

Say we want to start with the token 91. Well, we feed it into the network. Remember that the network produces probabilities.

It gives us this probability vector shown here:

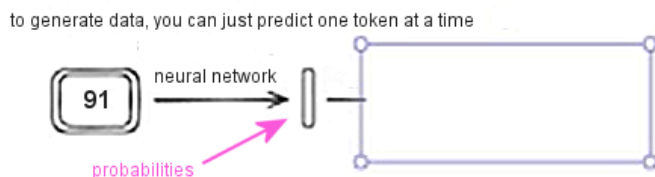


Figure 19: Probability vector (of next token) given token 91.

What we can do now is we can basically flip a biased coin¹⁸ we can sample (= randomly generate) a token based on this probability distribution. The tokens that are given high probability by the model are more likely to be sampled when you flip this biased coin. You can think of it that way.

¹⁸By "biased coin" the author doesn't mean a random variable with two possible outcomes. Think of a Wheel of Fortune with 100 277 possible outcomes, with probabilities given by the vector produced.

So we sample from the distribution to get a single unique token. For example token 860 comes next.

Now while 860 is a relatively likely token – given 91 before it – it might not be the only possible token. There could be many other tokens that could have been sample using the probability vector shown in fig. 19. But we could see that 860 is a relatively likely token. And indeed in our training set (fig. 13) 860 does follow 91.

Let's continue the process. We saw that after 91 the model gave us 860. We append it. And again we ask: what is the next token, the third one now? We get 287. It is still consistent with the data set.

Let's sample one more time. Now we start from the sequence of three token (91, 860, 287). The model produces a vector of probabilities for the fourth token. We sample from that vector, and suppose we get 11579 (fig. 20).

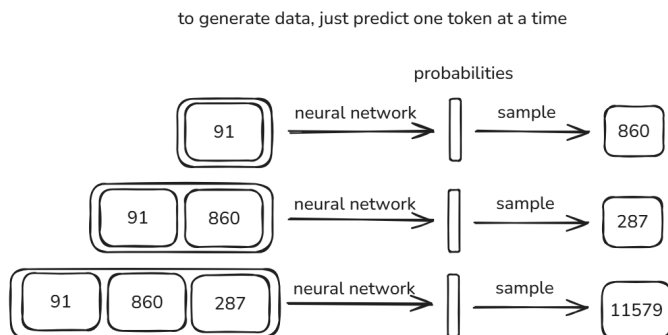


Figure 20: Starting from 91, we sampled 860, then 287, then 11579.

Let's say we do it one more time, to get the fifth token. We take those four (91, 860, 287, 11579). We ask the model what is the vector of probabilities for fifth token. We sample and get, say, 13659.

This is not actually 3962, that is this is not the fifth token in our training data set. This token 13659 is the token "Article" instead.

Our generative process, starting from 91 (the vertical bar), has produced

|Viewing Single Article

Thus we didn't exactly reproduce the sequence that we saw in the training data, which was

|Viewing Single Post

Go back to figure 12, page 15, if necessary to check what was the tokenization of the training data set.

Keep in mind that these systems are stochastic.

We're sampling. That is we are picking the next token using a random procedure – what I call "flipping a coin" in a generalized sense.

Sometimes we luck out and reproduce exactly the next small chunk of text in the training set. But sometimes we're getting a token that is not verbatim part of any of the documents in the training data.

So we're going to get some sort of remixes of the data that we had in the training, because at every step of the way we can sample and get a slightly different token.

And then once that new token makes it in, if you sample the next one and so on, you very quickly start to generate token streams that are very different from the token streams that occur in the training documents.

Statistically they will have similar properties but they are not identical to your training data. They're "inspired by" the training data. And thus, in the example that we constructed, we got a slightly different sequence. And why would we get " Article"? You might imagine that " Article" is a relatively likely token in the context of

|Viewing Single



Figure 21: Building a piece of text one token at a time. Each new token is one of the most likely given the preceding ones, which form the *context* to produce a vector of probabilities for the new one, and we sample it.¹⁹

You can imagine that the word "Article" followed this context window somewhere in the training documents. And we just happen to sample it here at that stage.

Inference is just predicting from these contexts, one token at a time, using each time the distribution of probability (computed by the neural network) of the next token given the context. We keep feeding tokens and getting the next one.

And we're always "flipping these coins" – in the sense of randomly sampling. And depending on how lucky or unlucky we are, we might get very different kinds of patterns depending on how we sample from these probability distributions.

So that's *inference*.

In most common scenarios, downloading the internet and tokenizing it, is a *pre-processing step* that you do only once.

Then, once you have your huge token sequence, you can start training networks (see fig. 13, p. 16, for the beginning of the training data set).

In practical cases you would try to train many different networks of different kinds of settings and different kinds of arrangements and different kinds of sizes. So you'll be doing a lot of neural network training.

¹⁹Pay attention to the fact that we don't systematically pick *the most likely*. We pick at random, so we usually pick *one of the most likely*.

Then once you have a neural network and you have trained it and you have some specific set of parameters that you're happy with, then you can take the model and you can do inference.²⁰ And you can actually generate data from the model.

When you're on ChatGPT and you're talking with the model. That model has been trained by OpenAI many months ago. They have a specific set of weights that work well.

When you're talking to the model, now all it does is only inference. There's no more training to adjust parameters.

Those parameters are held fixed and you're just somehow conversing with the model. You're giving it a piece of text, which is converted into a sequence of tokens called the *context*. When the LLM "talks to you", it is completing token sequences and that's what you're seeing generated when you use ChatGPT. So that model does only inference.

00:31:00

7 GPT-2: training and inference

Let's look at an example of training and inference that is concrete and gives you a sense of what this actually looks like when these models are trained.

The example that I would like to work with, that I'm particularly fond of, is that of OpenAI GPT-2.²¹

²⁰Note that inference is already used during the training process, where its role is to help build an effective set of parameters. But, more importantly, inference is the key mechanism when we deploy an LLM to generate new text — such as producing answers to questions or other outputs in real-world usage.

²¹Andrej Karpathy (born 1986), is a Slovak-Canadian computer scientist. He was a founding member and research scientist at OpenAI from 2015 to 2017. He then went to Tesla. He left it in early 2023 to join back OpenAI, before departing again in February 2024.

As of mid-2025, Andrej Karpathy is the founder of Eureka Labs, an AI education company focused on teaching and democratizing AI principles and practice. He develops online courses (such as LLM101n), creates educational video content on large language models, and promotes AI teaching assistants and hands-on learning in machine learning.

GPT stands for *Generative Pre-trained Transformer* and GPT-2 is the second iteration of the GPT series by OpenAI.

When you are talking to ChatGPT today, the model that is underlying all of the magic of that interaction is GPT-4, that is the fourth iteration of that series.

GPT-2 was published in February 2019 by OpenAI in the article "Language Models are Unsupervised Multitask Learners" by Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever.

Language Models are Unsupervised Multitask Learners

Alec Radford ^{*1} Jeffrey Wu ^{*1} Rewon Child ¹ David Luan ¹ Dario Amodei ^{**1} Ilya Sutskever ^{**1}

Abstract

Natural language processing tasks, such as question answering, machine translation, reading comprehension, and summarization, are typically approached with supervised learning on task-specific datasets. We demonstrate that language models begin to learn these tasks without any explicit supervision when trained on a new dataset of millions of webpages called WebText. When conditioned on a document plus questions, the answers generated by the language model reach 55 F1 on the CoQA dataset - matching or exceeding the performance of 3 out of 4 baseline systems without using the 127,000+ training examples. The capacity of the language model is essential to the success of zero-shot task transfer and increasing it improves performance in a log-linear fashion across tasks. Our largest model, GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested language modeling datasets in a zero-shot setting but still underfits WebText. Samples from the

competent generalists. We would like to move towards more general systems which can perform many tasks - eventually without the need to manually create and label a training dataset for each one.

The dominant approach to creating ML systems is to collect a dataset of training examples demonstrating correct behavior for a desired task, train a system to imitate these behaviors, and then test its performance on independent and identically distributed (IID) held-out examples. This has served well to make progress on narrow experts. But the often erratic behavior of captioning models (Lake et al., 2017), reading comprehension systems (Jia & Liang, 2017), and image classifiers (Alcorn et al., 2018) on the diversity and variety of possible inputs highlights some of the shortcomings of this approach.

Our suspicion is that the prevalence of single task training on single domain datasets is a major contributor to the lack of generalization observed in current systems. Progress towards robust systems with current architectures is likely to require training and measuring performance on a wide range of domains and tasks. Recently, several benchmarks

Figure 22: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

The reason I like GPT-2 is that it is the first time that a recognizably modern stack came together. All of the pieces of GPT-2 are recognizable today by modern standards. It's just that everything has gotten bigger now.

I'm not going to be able to go into the full details of this paper of course because it is a technical publication.

But some of the details that I would like to highlight are as follows:

- GPT-2 was a transformer neural network just like the neural networks you would work with today.
- It had 1.5 billion parameters. So these are the parameters that we looked at, fig. 17. It would have 1.5 billion of them. Today, modern transformers would have a lot closer to a trillion or several hundred billion probably.
- The maximum context length was 1024 tokens. So when we are taking chunks of windows of tokens from the data set, we're never taking more than 1024 tokens. And so when you are trying to predict the next token in a sequence you will never have more than 1024 tokens in your context in order to make that prediction.

This is also tiny by modern standards. Today the context lengths would be closer to a couple hundred thousand or maybe even a million. So today you have a lot more context, a lot more tokens in history, and you can make much better prediction about the next token in the sequence in that way.

- GPT-2 was trained on approximately 100 billion tokens. This is also fairly small by modern standards. As I mentioned the Fineweb data set that we looked at, the Fineweb data set has 15 trillion tokens. So 100 billion is quite small.

I actually tried to reproduce GPT-2 for fun, as part of this project called llm.c. You can see my write-up about doing that in a post on GitHub under the llm.c repository.²²

In particular, the cost of training GPT-2 in 2019 was estimated to be around \$40 000, but today you can do significantly better than that. For example, in my experiment, it took about one day and cost about \$600 — and that wasn't even really optimizing for cost. I think you could probably bring this down to around \$100 today.

²²<https://github.com/karpathy/llm.c/discussions/481>

Title: "Reproducing GPT-2 (124M) in llm.c in 90 minutes for \$20"

Why have the costs come down so much? Well, first, the datasets are a lot better now, and the way we filter, extract, and prepare them has become much more refined. As a result, the datasets are just much higher quality. So that's one thing

But really the biggest difference is that our computers have gotten much faster in terms of the hardware – we're going to look at that in a second. Also the software for running these models and really squeezing out as much speed from the hardware as it is possible, that software has also gotten much better, as everyone has focused on these models and try to run them very very quickly.

I'm not going to be able to go into the full detail of this GPT-2 reproduction. The paper I mentioned is a long technical post. But I would like to still give you an intuitive sense for what it looks like to actually train one of these models as a researcher. What are you looking at? And what does it look like? what does it feel like? So let me give you a sense of that a little bit.

This is what it looks like.

```

step 381/20000 loss 5.518558 (-1.299) norm 0.6696 (-0.299) lr 3.27e-04 6744.32 ms 61.1% bf16 MFU | 152448 tok/s
step 382/20000 loss 5.513454 (-1.332) norm 0.6295 (-0.492) lr 3.27e-04 6778.76 ms 60.9% bf16 MFU | 152568 tok/s
step 383/20000 loss 5.585824 (-1.952) norm 0.4897 (-1.222) lr 3.28e-04 6817.14 ms 60.5% bf16 MFU | 152622 tok/s
step 384/20000 loss 5.564928 (-1.522) norm 0.6167 (-0.532) lr 3.29e-04 6838.21 ms 60.2% bf16 MFU | 152658 tok/s
step 385/20000 loss 5.498973 (-1.332) norm 0.4793 (-1.152) lr 3.30e-04 6856.52 ms 60.1% bf16 MFU | 152678 tok/s
step 386/20000 loss 5.497883 (-1.382) norm 0.5078 (-1.082) lr 3.31e-04 6873.37 ms 60.0% bf16 MFU | 152664 tok/s
step 387/20000 loss 5.442298 (-2.242) norm 0.3630 (-0.762) lr 3.32e-04 6894.13 ms 59.9% bf16 MFU | 152647 tok/s
step 388/20000 loss 5.580096 (-1.812) norm 0.6175 (-0.462) lr 3.33e-04 6894.67 ms 59.9% bf16 MFU | 152619 tok/s
step 389/20000 loss 5.486438 (-1.862) norm 0.7342 (+0.192) lr 3.33e-04 6898.52 ms 59.8% bf16 MFU | 152588 tok/s
step 390/20000 loss 5.581055 (-1.732) norm 0.6806 (-0.382) lr 3.34e-04 6902.34 ms 59.8% bf16 MFU | 152554 tok/s
step 391/20000 loss 5.485813 (-1.812) norm 0.7188 (+0.122) lr 3.35e-04 6903.48 ms 59.7% bf16 MFU | 152521 tok/s
step 392/20000 loss 5.428624 (-2.142) norm 0.4663 (-1.262) lr 3.36e-04 6906.22 ms 59.7% bf16 MFU | 152487 tok/s
step 393/20000 loss 5.566981 (-1.592) norm 0.6174 (-0.432) lr 3.37e-04 6905.48 ms 59.7% bf16 MFU | 152455 tok/s
step 394/20000 loss 5.478741 (-1.742) norm 0.5577 (-0.752) lr 3.38e-04 6902.39 ms 59.6% bf16 MFU | 152428 tok/s
step 395/20000 loss 5.495521 (-1.612) norm 0.6721 (-0.122) lr 3.39e-04 6898.85 ms 59.6% bf16 MFU | 152486 tok/s
step 396/20000 loss 5.541908 (-1.772) norm 0.6840 (+1.032) lr 3.39e-04 6905.11 ms 59.7% bf16 MFU | 152378 tok/s
step 397/20000 loss 5.498292 (-1.562) norm 0.8762 (+0.972) lr 3.40e-04 6905.07 ms 59.7% bf16 MFU | 152352 tok/s
step 398/20000 loss 5.542685 (-1.222) norm 0.8837 (+0.992) lr 3.41e-04 6904.35 ms 59.7% bf16 MFU | 152328 tok/s
step 399/20000 loss 5.583354 (-1.432) norm 0.5941 (+1.092) lr 3.42e-04 6905.22 ms 59.7% bf16 MFU | 152284 tok/s
step 400/20000 loss 5.522326 (-1.312) norm 0.6659 (-0.242) lr 3.43e-04 6901.09 ms 59.8% bf16 MFU | 152286 tok/s
generating:
0 I obviously something on the inside in order to start myself in the event, everything I ran for a cart that, so it works a culture,
screamed, I wanted to the show up with odd cycle and yours, igniting everything chial palro gorror horror sound,6s backstage with f
eeting blankets planned.
Contact: Voters, Social version 2ndance were terminated by Capting Bigbrook Rd. You may have teamed up a show in this stand wherever
to any never smooth sessions as you forget about how to see a lot of attack, but I had paused at a picture of them.<endofcontext>Click
k here! 30h.uk court/ing/vents varies so onflant too tyman and reimation and 8x.
podcast: Compa PrepadesFlumba. CLICK on Wonderplace-difficultist-Jitzey Ring Market Player Ejova BeasLES domains
Akarabaz Vadel Ameraervo WIVEA/Dargleza Akouat Fedal Isdian Yu-Gayani King Campia River Rdversaalat van Virtual Steel Maletierthyal:
Nebrids
kit should also quot ques on internationally. Put through fenced area of A
step 401/20000 loss 5.482952 (-1.552) norm 0.5776 (-0.732) lr 3.44e-04 | 6745.28 ms | 61.1% bf16 MFU | 152445 tok/s
step 402/20000 loss 5.445932 (-1.772) norm 0.6173 (-0.582) lr 3.45e-04 | 6778.58 ms | 60.8% bf16 MFU | 152557 tok/s
step 403/20000 loss 5.484241 (-1.492) norm 0.6258 (-0.452) lr 3.45e-04 | 6813.01 ms | 60.5% bf16 MFU | 152625 tok/s

```

Figure 23: Training GPT-2.

What I'm doing here is I'm training a GPT-2 model right now [while speaking in the video]. What's happening here, fig. 23, is that every single line in the array like this one is one update of the model.

Remember how we are making the prediction better for every one of the tokens, given a context window, see fig. 16, and how we are updating the weights or parameters of the neural net.

So here every single line is one update to the neural network, where we change its parameters by a little bit. So that it is better at predicting next token and sequence.

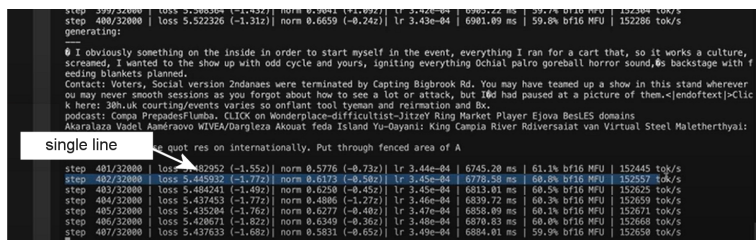


Figure 24: Updating the parameters.

With each line of training, the model's parameters are updated to improve predictions for one million tokens in the training set, all at once. In other words, we've extracted a batch of one million tokens from the dataset and are working to enhance the model's accuracy for predicting the next token in each sequence — simultaneously across the entire batch.

At every step in this process, the network's parameters are updated to reflect the learning for that batch.

Now the number to watch closely is this number called *loss*.

The loss is a single number that is telling you how well your neural network is performing right now. And it is created so that low loss is good. You can see that the loss is decreasing as we make more updates to the neural net which corresponds to making better predictions on the next token in a sequence.

The loss is the number that you are watching as a neural network researcher. You are waiting, you're twiddling your thumbs, you're drinking coffee, and you're making sure that this looks good, that is, with every update your loss is improving and the network is getting better at prediction.

Here you see that we are processing 1 million tokens per update. Each update takes about 7 seconds. We are going to process a total of 32,000 steps of optimization. So 32,000 steps with 1 million tokens each is about 33 billion tokens that we are going to process.

We're currently only about 420 steps – 420 out of 32,000. So we are still only a bit more than 1% done, because I've only been running this for about a quarter of an hour.

Every 20 steps I have configured this optimization to do inference.

What you're seeing here is the model predicting the next token in a sequence [watch video to see new tokens appearing sequentially, time mark 00:37:10 and after]. You started it randomly. Then you continue plugging in the tokens. So we're running this inference step, see fig. 20. We see the model in the process of predicting the next token in the sequence. Every time you see something appear it's a new token.

Let's examine the generated text. As shown in Fig. 24, it is still not very coherent — keep in mind that this is only 1% into training, so the model has not yet learned to predict the next token accurately. Much of the output resembles gibberish, as expected at this early stage. However, some fragments begin to display local coherence.

"Since she is mine", "it's a part of the information", "should discuss my father", "great companions", "Gordon showed me", "sitting over it", etc.

00:38:00

I know it doesn't look very good yet, but let's scroll up and see what it looked like when I started the optimization.

You see that what we're getting here (after only one round of optimisation, i.e. the first batch of 20 updates) looks completely random.

200 ifAnother:# mortar ris partners Navodsulic advers exp-
ped slime negotiate on desper asudging Healthsecondsot-
ted, acknowledged lethal-Mode

{ "Gott", "" inviting Ballistic.

ominpart, and Tou Australians dispensarieskukuagne801
and%] Wright etnh408 of filed bitterness the humpSpot
Roads thickness, wing aknowledgement342 Glou flame overcl
Summoner quirk sr. seismic. wrecked, cite toAIN Brus
Und leastDecember Moose reinforce Tay in Fishing [here
two Japanese characters] divergence entity friendshippen-
ment% olds He%the to onesAnother markets superfluMe
priced...

Of course, this is because the model has only undergone 20 parameter updates so far, so it produces essentially random text—it's a random network at this stage.

We just saw that after completing only 1% of the total updates, the model was already performing significantly better compared to after just 20 steps.

If we were to continue training for the full 32 000 steps, the model would improve to the point where it can generate fairly coherent English text and produce correct token sequences, resulting in much better English overall.

This training has to run for about a day or two. At present, we just make sure that the loss is decreasing. Everything is looking good (see fig. 23 & 24 that are snapshots of the process), we just have to wait.

Let's turn to the computation that is required, because of course I'm not running this optimization on my laptop. That would be way too expensive [in terms of time]. We have to run this neural network, and we have to improve it, and we have we need all this data and so on. You can't run this too well on your computer because the network is just too large.

So all of this training of our LLM (to get a good collection of parameters – about 85 000 of them) is running on a computer that is out there in the cloud.

I want to address the compute side of the story of training these models and what that looks like. So let's take a look.

The computer I'm using is an 8x H100 node²³ — that means there are eight H100 GPUs in a single machine. I'm renting this computer somewhere in the cloud; I am not sure where it is located physically. I happen to use Lambda for renting (<https://lambda.ai>), but there are lots of other companies offering similar services.

If you look at their on-demand pricing, you'll see machines that have eight H100 GPUs. For example, you can get an on-demand machine with eight Nvidia H100 GPUs for about \$3 per GPU per hour. You rent it, get access to the server in the cloud, and you can use it to train models.

The H100 GPU itself looks like this :

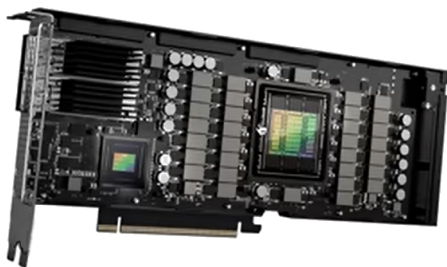


Figure 25: One Nvidia H100 GPU.

It's a card that you can slot into your computer.²⁴ GPUs are an ideal fit for training neural networks since the process

²³ An 8x H100 node computer is a high-performance server or workstation containing eight NVIDIA H100 GPUs (graphics processing units), all working together in a single machine.

²⁴ But Karpaty doesn't rent the cards to put into his own machine. It's all taken care of by Lambda.

is very computationally expensive. But, above all, GPUs are designed for parallel operations and can handle this kind of workload efficiently.

So you can have many independent workers working all at the same time in solving the matrix multiplication that's under the hood of training these neural networks.

Figure 25 shows just one of these H100s, but actually you would put many of them together. You can stack eight of them into a single node.

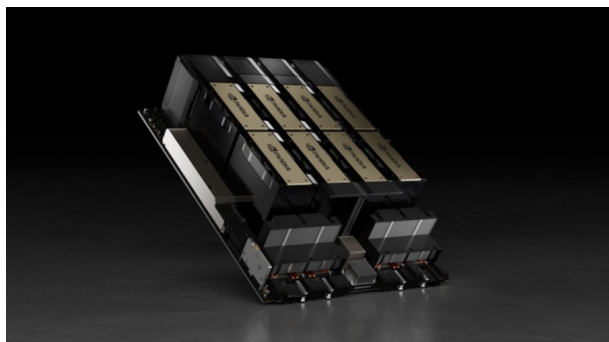


Figure 26: Eight Nvidia H100 GPUs stacked into one node.

Then you can stack multiple nodes into an entire data center or an entire system.

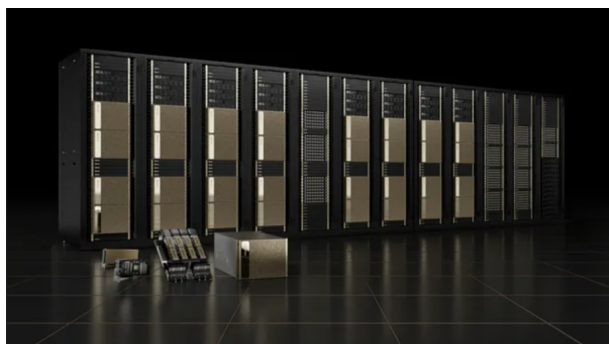


Figure 27: System made of many nodes.

When we look at a data center we start to see things that look like what figure 27 shows. One GPU goes to eight GPUs goes to a single system goes to many systems.

Figure 28 shows a bigger data center. This is of course much more expensive.



Figure 28: In 2023, Google announced new TPUs, equipped with Nvidia H100 GPUs, offering generative AI-focused cloud services.

What’s happening is that all the big tech companies really desire these GPUs so they can train all these language models because these GPUs are so powerful.

That’s the reason behind Nvidia’s meteoric rise in market value, reaching \$3.4 trillion in early 2025.²⁵

This is the new Gold Rush. If you are an LLM provider, you want to get these GPUs — get enough of them so they can work together efficiently to optimize your model parameters (see fig. 23 for instance) which are the core of your LLM.

What are they all doing? They’re all collaborating to predict the next token on a data set like the Fineweb data set.

This is the computational workflow that is extremely expensive. The more GPUs you have, the more tokens you can try to predict, leading to improving your weights. With more GPUs, you process your data set faster; you iterate faster; you can get a bigger network, train a bigger network, etc.

²⁵Nvidia’s current market value (as of October 6–7, 2025) is approximately \$4.51 trillion. A trillion is 10^{12} (ten to the twelfth power), or 1,000,000,000,000.

This is what all those machines look like, and are doing. And this is why all of this is such a big deal.

For example this is a article from November 2024.

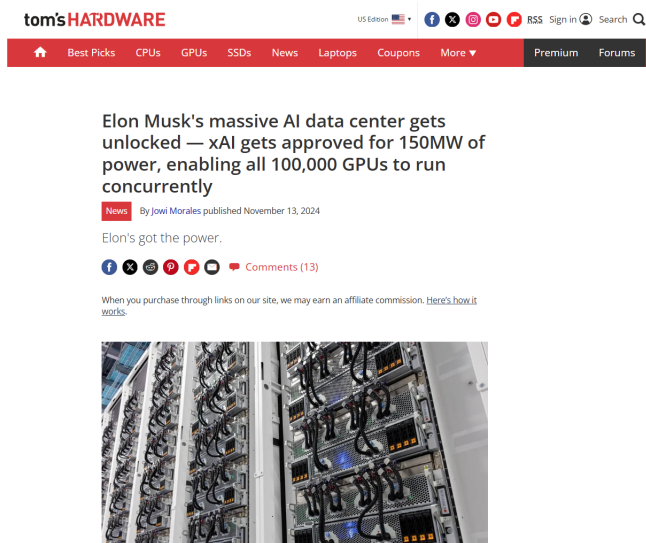


Figure 29: Elon Musk's invests in a massive AI data center.

This is why it's such a big deal. We see that Elon Musk is assembling 100 000 GPUs in a single data center. Each of these GPUs is extremely expensive²⁶ and will require an enormous amount of electrical power to operate. All of them are just trying to predict the next token in the sequence and improve the network by doing so, and get probably a lot more coherent text than what we've seen in our nanoGPT example, and a lot faster.

Unfortunately I do not have tens or hundreds of millions of dollars to spend on training a really big commercial model.

²⁶The cost of a single NVIDIA H100 GPU in 2025 ranges from approximately \$25 000 to \$30 000, depending on model and configuration. Premium or high-memory variants (such as the SXM models) can cost up to \$40 000 per unit. Even taking into account volume discounts, this represents an investment of possibly \$2 billion.

Luckily we can turn to some big tech companies who train these models routinely and release some of them once they are done training.

They've spent a huge amount of compute to train their network, and they release the network at the end of the optimization. So it's very useful because they've done a lot of compute for that.

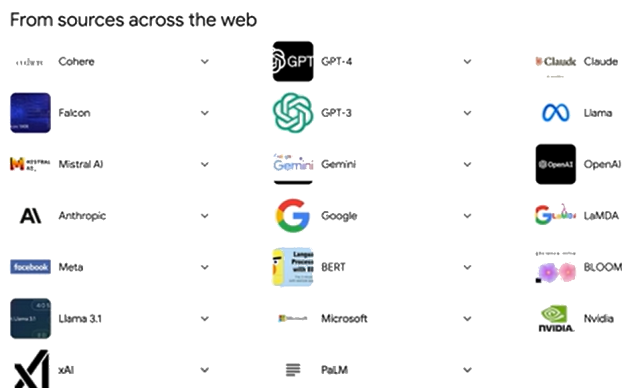


Figure 30: Companies training their own LLM.

00:43:17

8 Llama 3.1 base model inference

There are many companies who train these models routinely, see fig. 30. But actually not many of them release what they get at the end of the training we have studied so far. Once the parameters have been adjusted, via the training process we've seen, we get what is called a *base model*.

What is a base model? It is essentially a *token simulator*. Given a piece of clean text — say, from the Internet — it predicts the next token in the sequence. By itself, it is not yet very useful, because what we ultimately want is the ability to ask questions and receive meaningful answers. A more advanced model capable of that is called an *assistant*.

Models like the one shown in figure 23 do not provide answers. Instead, they produce something akin to remixes of the Internet: they can "continue" web pages, one token at a time. Base models are therefore rarely released, as they represent only the first stage in a longer process, with additional steps required to build a true assistant.

However a few base models have been released. For instance the GPT-2 model – a 1.5 billion parameter model which was released in 2019 – was a base model.

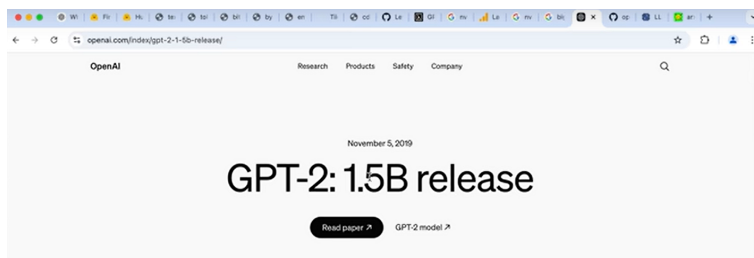


Figure 31: GPT-2 1.5B released by OpenAI in November 2019 was a base model.

Now what is a model release? What does it mean to release these models?

This is the GPT-2 repository on GitHub.

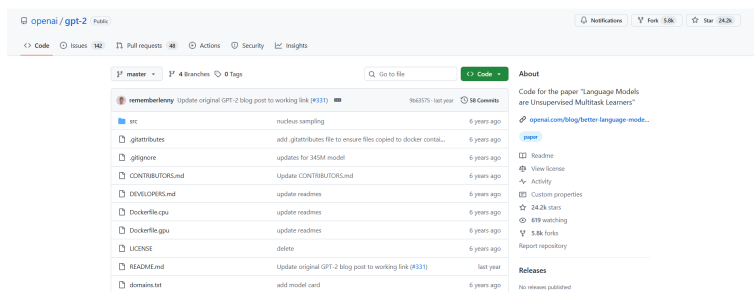


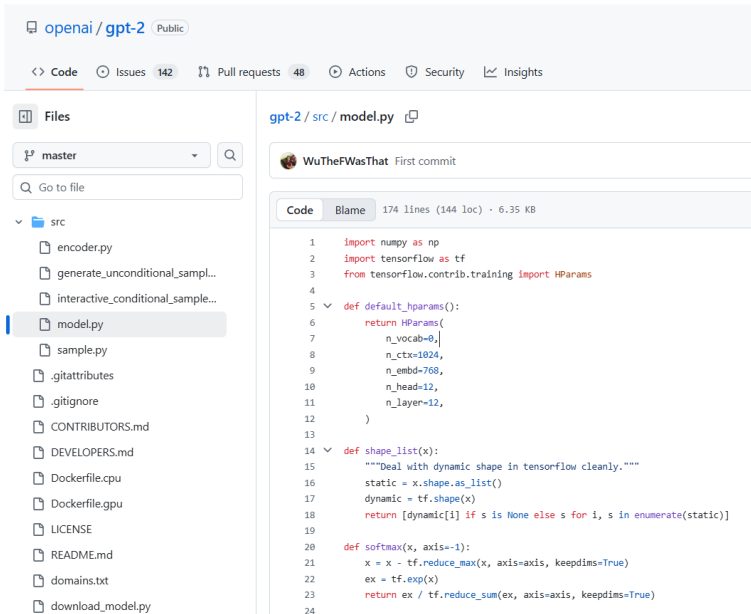
Figure 32: GPT-2 repository on GitHub, <https://github.com/openai/gpt-2/tree/master>

We need two things to release a model:

First, we need the code — usually written in Python — that describes in detail the sequence of operations performed within their model.

Remember the NanoGPT transformer (fig. 18, page 24) and the sequence of steps (mathematical operations) that we described.

Similarly, the ChatGPT-2 neural network executes operations described by the code, the beginning of which is shown below:



The screenshot shows the GitHub interface for the repository `openai/gpt-2`. The left sidebar displays the file structure, with `src/model.py` selected. The main area shows the Python code for `model.py`, which includes imports for `numpy` and `tensorflow`, and defines functions for `default_hparams`, `shape_list`, and `softmax`.

```

1  import numpy as np
2  import tensorflow as tf
3  from tensorflow.contrib.training import HParams
4
5  def default_hparams():
6      return HParams(
7          n_vocab=0,
8          n_ctx=1024,
9          n_embd=768,
10         n_head=12,
11         n_layer=12,
12     )
13
14  def shape_list(x):
15      """Deal with dynamic shape in tensorflow cleanly."""
16      static = x.shape.as_list()
17      dynamic = tf.shape(x)
18      return [dynamic[i] if s is None else s for i, s in enumerate(static)]
19
20  def softmax(x, axis=-1):
21      x = x - tf.reduce_max(x, axis=axis, keepdims=True)
22      ex = tf.exp(x)
23      return ex / tf.reduce_sum(ex, axis=axis, keepdims=True)
24

```

Figure 33: Python code of GPT-2, <https://github.com/openai/gpt-2/blob/master/src/model.py>

This code is implementing what's called a *forward pass* of the neural network. So we need the specific details of exactly how they wired up their neural network.

This is just computer code, usually just a couple hundred lines of code. It's not that crazy; and this is all fairly understandable and usually fairly standard.

What's not standard is the collection of parameters. That's where the actual value is.

What are the parameters of this neural network? There's 1.5 billion of them and we need the correct setting or a really good setting.

That's why the second thing we need, in addition to the source code, is the parameters they released – which in this case is roughly 1.5 billion parameters.

These are just numbers. It's one single list of 1.5 billion numbers the precise and good setting of all the "knobs" – which I used as an image of optimization of the parameters – such that the tokens come out well when we sequentially guess the next one.

So you need those two things to get a base model release.

Thus GPT-2 was released (with the two important things a release encompasses) but that's actually a fairly old model as I mentioned.

We're going to turn to is the LLM called *Llama 3* and that's the one that I would like to show you next. Remember that GPT-2 was a 1.5 billion parameter model trained on 100 billion tokens. Llama 3 is a much bigger model and much more modern model.

It was built and trained by Meta. It is a 405 billion parameter model trained on 15 trillion tokens in very much the same way as what we saw, just much much bigger.

Meta has also made a release of Llama 3 in July 2024. That was part of the paper entitled "The Llama 3 Herd of Models" (<https://ai.meta.com/research/publications/the-llama-3-herd-of-models/>)

With this paper (the beginning of which is shown in figure 34) that goes into a lot of detail, the biggest base model that they released is the Llama 3.1 405 billion parameter model. It is the base model.

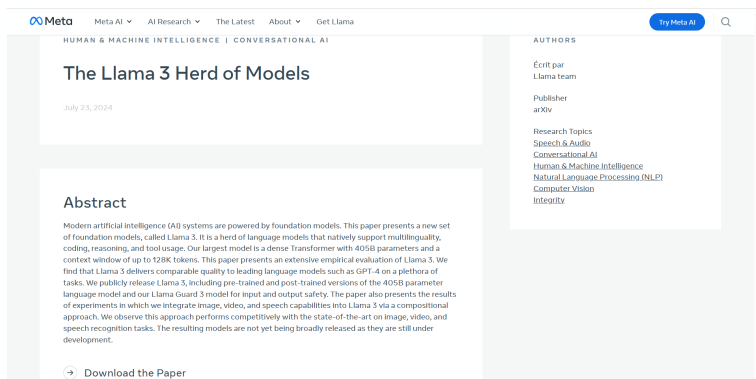


Figure 34: Meta paper introducing Llama 3.²⁷

In addition to the base model, you can see in table 1 of their paper that – anticipating later sections of my video – they also released the instruct model. The word instruct means that this is an *assistant*: you can ask it questions and it will give you answers. We still have yet to cover that part later.

For now let's just look at this base model, this token simulator. Let's play with it and try to think about: What is this thing? How does it work? What do we get at the end of this optimization if you let this run until the end for a very big neural network on a lot of data?

²⁷Here is the abstract : Modern artificial intelligence (AI) systems are powered by foundation models. This paper presents a new set of foundation models, called Llama 3. It is a herd of language models that natively support multilinguality, coding, reasoning, and tool usage. Our largest model is a dense Transformer with 405B parameters and a context window of up to 128K tokens. This paper presents an extensive empirical evaluation of Llama 3. We find that Llama 3 delivers comparable quality to leading language models such as GPT-4 on a plethora of tasks. We publicly release Llama 3, including pre-trained and post-trained versions of the 405B parameter language model and our Llama Guard 3 model for input and output safety. The paper also presents the results of experiments in which we integrate image, video, and speech capabilities into Llama 3 via a compositional approach. We observe this approach performs competitively with the state-of-the-art on image, video, and speech recognition tasks. The resulting models are not yet being broadly released as they are still under development.

00:47:00

My favorite place to interact with the base model is this company called *hyperbolic* (<https://hyperbolic.ai>) which is serving the base model of the 405b Llama 3.1.

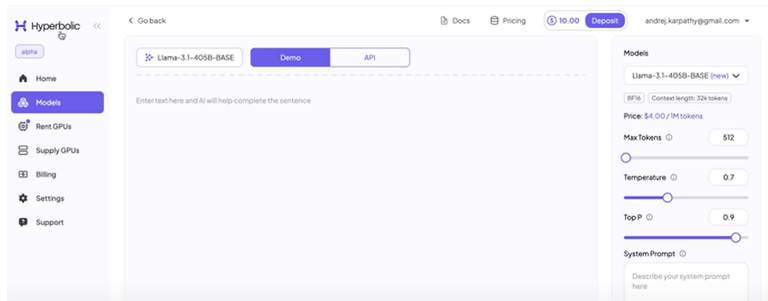


Figure 35: Llama 3.1 405b tried via the platform hyperbolic.ai.²⁸

When you go to the website, you’ll probably need to register. Once you’re in, make sure you select the “llama 3.1 405B BASE” model — it must be the base version, not the instruct one.

For the “max tokens” setting on the right side of figure 35 – that’s how many tokens the model will try to generate – let’s lower this a bit, just to save compute. We can set it to something like 128 tokens and leave the other settings as they are. I’m not going to explain every detail here.

Fundamentally, what’s about to happen when you run this model is identical to what happened during model inference when we were training GPT-2 (fig. 23, page 34): the model will simply keep on guessing sequentially the next token given what’s already known or guessed.

²⁸As of October 2025, the address <https://app.hyperbolic.ai/models/llama31-405b-base-bf16> is no longer active. It now gives an error 404. But we can play with various models of the instruct variety, like <https://app.hyperbolic.ai/models/llama-3-3-70b-instruct>

First, let's see that this model is *not yet an assistant*.

For example, if you ask it “what is 2 plus 2?”, it won't just reply “it's four, what else can I help you with?” because *this is not a fine-tuned assistant* – it is just a base model.

What will happen is the prompt “what is 2 plus 2?” will be tokenized; those tokens become the prefix, and the model predicts only the most likely next token²⁹ each time — it's just a glorified *autocomplete* of what comes next.

Its output depends entirely on the statistical patterns it saw in training data (basically a lot of web pages).

So let's ask “What is 2 plus 2?”. And let's hit enter and see what continuation it generates.

What is 2+2?

This is a simple question, and the answer is 4. This is an example of an objective truth. No matter how many times you add 2 and 2, you will get the same answer. It is not possible to get a different answer. This is why mathematics is considered to be a form of objective truth. It is a way of describing the world that is not influenced by our own personal beliefs or opinions. The same is true for other areas of knowledge. The laws of physics are another example of objective truth. They describe the way the world works, and they are not influenced by our own personal beliefs or opinions.

Enter text here and AI will help complete the sentence

In this case, the model actually tries to answer the question, but then drifts into some unrelated philosophical territory.

Let's try this again: I'll just copy and paste, starting from scratch, and ask once more: “What is 2 plus 2?”

What is 2+2?

Is it 4 or 22? How do you know? What about 1+1+1+1? Is it 4 or 11? How do you know? Why do you group the numbers together the way you do? Why do you add the numbers in a particular order? What rules do you use to determine the way you group numbers? Mathematics is a human activity. It is a set of ideas that people came up with. We are so used to the way we do mathematics that we forget that there are

²⁹Or *one of the most likely*, most of the time, because it is stochastic.

other ways to do mathematics. There are other ways to group numbers together and other ways to add numbers.

Enter text here and AI will help complete the sentence

Okay, so it goes off again, and in a different direction.

There's something important I want to emphasize: every time you put in the prompt, the system starts afresh. *It's a stochastic system* — so, even with the same prefix of tokens, you'll always get a different answer.

The reason for that is you work with a probability distribution for the next token (see figure 15, page 19). The model samples from it. So every time you run it, you get different samples. It heads off in a different direction each time ending up in a different territory.

For example, let me try one more time:

What is 2+2?

What is 2+2? This is a featured post. It will be sticky at the top of all recent posts for all to see! There are no comments on this post. Be the first!

Enter text here and AI will help complete the sentence

We see that it just keeps going, producing outputs based on what it saw on the Internet, basically just regurgitating those statistical patterns.

00:49:30

So, keep this in mind about a base model :

1. it is not an assistant yet, it's just a token autocomplete,
2. it is a stochastic system, meaning the output is random and can change each time.

The crucial thing is, even though this model by itself isn't all that useful for many applications just yet, it's still very valuable, because in order to get really good at predicting the next token, the model had to learn a lot about the world. All that knowledge is packed into the parameters of the network.

Remember, the training text was just internet web pages [and a few other publicly available sources of high quality texts], and now all of that is sort compressed into the weights of the neural network.³⁰

You can think of these 405 billion parameters as a kind of "compression" of the internet. It's like a zip file, except it's not lossless compression — it's lossy. What we're left with is a kind of gestalt perception or fuzzy snapshot or abstract gist of the internet, and we can generate text from it. Right now, we can elicit some of that stored knowledge by prompting the base model.

Here's a prompt that might work to pull out some of the knowledge that's hiding in the model's parameters: "Here is my top 10 list of the top landmarks to see in Paris."

I'm phrasing it this way to prime the model to continue the list. Let's see what happens when I run it.

Here is my top 10 list of landmarks to see in Paris:

1) Eiffel Tower. This is a must. You can take a lift to the top of the tower for 13.50 euros or take the stairs for 4.50 euros. The tower is 324m tall and the highest structure in Paris. I went up the tower at night and the view was amazing! 2) Notre Dame. This is a gothic style cathedral located on the eastern half of Paris. It was one of the first cathedrals to use flying buttresses for support. The cathedral is free to enter but if you want to climb the tower it costs 8.50 euros. 3) Arc de Triomphe. The arch was built to honor those who fought for France in the Napoleonic wars. It is located on the western end of the Champs-Elysees. It costs 9.50 euros to go to the top of the arch. 4) Champs-

³⁰This is a very important idea — albeit a heuristic one: *the information and knowledge — absolutely all of it — that the neural network acquires during training is, in some sense, encapsulated within the billions of parameters of the model we use.*

However, this encapsulation is hidden and, to some extent, mysterious. Even though it constitutes a form of "knowledge of the world," it is not a model of the world in the conventional sense.

In fact, building an AI that possesses an explicit model of the world — akin to the way humans represent knowledge in their head — is an active and tantalizing area of ongoing research.

Elysees. This is one of the most famous streets in the world. It is lined with cinemas, cafes, and luxury shops. The avenue runs from the Place de la Concorde to the Arc de Triomphe. 5) The Louvre. This is one of the largest museums in the world and is home to the Mona Lisa. It is located on the right bank of the Seine. The museum is housed in the Louvre Palace which was originally a fortress built in the 12th century. The museum is open every day except Tuesday and tickets cost 15 euros. 6) Sacre-Coeur. This is a Roman Catholic church located on the highest point in the city. The church is made of white stone and is one of the most popular tourist destinations in Paris. The church is free to enter but there is a charge to climb the dome. 7) Place de la Concorde. This is the largest public square in Paris. It is located at the eastern end of the Champs-Elysees. The square is home to the Obelisk of Luxor, a gift from Egypt, and the Fountain of River Commerce and Navigation. 8) Les Invalides. This is a complex of buildings that houses museums and monuments relating to the military history of France. It is located on the left bank of the Seine. The complex includes the Musee de l'Armee, the Musee des Plans-Reliefs, and the Dome des Invalides. 9) The Pantheon. This is a

Enter text here and AI will help complete the sentence

As you can see, it starts generating a list and gives me some landmarks. And notice that it's producing a lot of information.

You shouldn't fully trust all the information — it's just a recollection of internet documents.

Things that appear very frequently in the internet data are more likely to be remembered correctly, while less common facts might be less reliable.

So you can't fully trust everything here [about the Paris landmarks], because it's all just a vague recollection of what was on the Internet. The information isn't stored explicitly in the parameters; it's all just patterns that the model recalls.

That said we did get something that is probably approximately correct. I don't have the expertise to verify that this is roughly correct, but you see that we've elicited a lot of the knowledge of the model.

This knowledge is not precise and exact. It is vague and probabilistic and statistical. And the kinds of things that occur often are the kinds of things that are more likely to be remembered in the model.

00:52:00

Now I want to show you a few more examples of this model's behavior.

The first thing I want to show you is this example: I went to the Wikipedia page for zebra ;



Figure 36: Wikipedia page on zebra.

Let me just copy the first sentence. And let me paste it here [into the base model we are studying].

This sentence serves as a prefix, prompting the base model to continue generating text, one token at a time. Remember that it is not an *assistant* (that would answer questions). It is just a *base model* (that "continues" or "completes" text).

When I click enter, what kind of completion do we get?

Zebras (US: /zi:brɛz/, UK: /zebrɛz, 'zi:-/)[2] (subgenus *Hippotigris*) are African equines with distinctive black-and-white striped coats.

There are three living species: the Grévy's zebra (*Equus grevyi*), plains zebra (*E. quagga*), and the mountain zebra (*E. zebra*). Zebras share the genus *Equus* with horses and asses, the three groups being the only living members of the family *Equidae*. Zebra stripes come in different patterns, unique to each individual. Several theories have been proposed for the function of these stripes, with most evidence supporting them as a deterrent for biting flies. Zebras inhabit eastern and southern Africa and can be found in a variety of habitats such as savannahs, grasslands, woodlands, shrublands, and mountainous areas. Zebras are primarily grazers and can subsist on lower-quality vegetation. They are preyed on mainly by lions and typically flee when threatened but also bite and kick. Zebra species differ in social behaviour, with plains and mountain zebra living in stable harems consisting of an adult male or stallion, several adult females or mares, and their young or foals; while Grévy's zebra live alone or in loosely associated herds. In harem-holding species, adult females mate only with their harem

Figure 37: Base model completion of the prefix entered by the speaker – the first sentence of the page on zebra.

What the model is producing here is an *exact regurgitation of this Wikipedia entry*.³¹ It is reciting this Wikipedia entry purely from memory and this memory is stored in its parameters.

It is possible that at some point the model will start drifting away from the original Wikipedia entry. But you can see that it has memorized a large chunk of it.

Let me check — okay, this sentence (in the middle of what the model has produced) is still on track... here too, we're still matching.

Eventually though, the model will stray away, because it can't remember all the details perfectly. And remember that the completion has a stochastic aspect, although if a token has a high probability, it will be picked.

The reason this perfect recollection happens is that these models are really good at memorizing, which is usually not

³¹It just changed a word here and there, for instance it added an article in front of "Grévy's zebra" and removed one in front of "plains zebras".

what you want in a finished model. This phenomenon is called *regurgitation* — when the model recites parts of its training data word-for-word — which isn't desirable.³²

Why does this happen?

For high-quality documents like Wikipedia, the model's training process will often sample from them more frequently. That means the model might see the same web page as many as ten times during training. It's a bit like if you read the same text a hundred times; you'd be able to recite it.

The model is similar. If it sees something too often, it can repeat it by rote later on.

Except these models, in fact, can be even more efficient than humans in this respect. The model might have seen this Wikipedia entry on zebra only ten times, but it's still able to essentially memorize the article and embed it in its parameters.

00:54:20

The next thing I want to show you is something that the model has definitely not seen during its training.

For example if we go to the paper, "The Llama 3 Herd of Models", and navigate to the pre-training data, we see

³²Indeed, when it concerns a human being, it is called plagiarism. The issue of plagiarism by large language models (LLMs) is a significant concern for both LLM providers and the organizations whose original works are used as sources. Disputes often arise over copyright, fair use, and the ethical implications of reproducing or paraphrasing proprietary content without proper attribution.

The issue is more complex than it appears. Simply changing a word or rephrasing a sentence is a common technique used to circumvent plagiarism detection. However, when a person P expresses knowledge, it's almost always learned from external sources: books, websites, or other people, unless it is an original creation. We have all experienced this: we explain something we consider important and clear, and which represents our own ideas — only to have some interlocutor say: "Oh well, you just picked that up from so-and-so's book and are repeating it!"

So, when does restating what P has learned cross the line into plagiarism, and when is it considered genuine thinking expressed by P?

that the data set has a knowledge cut off at the end of 2023. Therefore it will not have seen documents after that date, and certainly it has not seen anything about the 2024 US presidential election and how it turned out.

Let's prime the model with tokens from the future. We extract from the Wikipedia page on the 2024 US presidential election the following prime: "Presidential elections were held in the United States on November 5, 2024. The Republican Party's ticket—Donald Trump, who was the 45th president of the United States from 2017 to 2021". Then the base model will continue the token sequence. It will just take its best guess according to the knowledge it has in its own parameters.

Let's take a look at what that looks like.

Presidential elections were held in the United States on November 5, 2024. The Republican Party's ticket—Donald Trump, who was the 45th president of the United States from 2017 to 2021

, and Vice President Mike Pence—won the election over the Democratic Party's ticket of former Secretary of State Hillary Clinton and Senator Tim Kaine. This was the first time since 1948 that a Republican candidate won a majority of the popular vote, the first time since 1988 that a Republican candidate won more than 300 electoral votes, and the first time since 1984 that a Republican candidate won more than 50% of the popular vote. The Republican Party's ticket—Donald Trump, who was the 45th president of the United States from 2017 to 2021, and Vice President Mike Pence—won the election over the Democratic Party's ticket of former Secretary of State Hillary Clinton and Senator Tim Kaine. This was the first time since 1948 that a Republican candidate won a majority of the popular vote, the first time since 1988 that a Republican candidate won more than 300 electoral votes, and the first time since 1984 that a Republican candidate won more than 50% of the popular vote. Trump's victory was widely considered to

Enter text here and AI will help complete the sentence

So it says that Mike Pence was the running mate instead of J. D. Vance, and the ticket was against Hillary Clinton and Tim Kane. This is an interesting parallel universe of what could have happened, created by the LLM.

Let's get a different sample.³³ Let's use the same identical prompt and resample:

Presidential elections were held in the United States on November 5, 2024. The Republican Party's ticket—Donald Trump, who was the 45th president of the United States from 2017 to 2021

, and his running mate, former U.S. representative Ron DeSantis—defeated the Democratic Party's ticket of incumbent president Joe Biden and vice president Kamala Harris. The election was a rematch of the 2020 presidential election, in which Biden defeated Trump by 4.5 percentage points and 7 million votes. Trump became the first president since Grover Cleveland to win a non-consecutive term, the second president after Cleveland to win the popular vote three times, and the first president to win the popular vote in three elections without winning the popular vote in all three. Biden became the first incumbent president to lose reelection since George H. W. Bush in

Here the running mate was Ron DeSantis and they ran against Joe Biden and Kamala Harris. This is again a parallel universe, a different one.

In summary, we see that the model will take educated guesses. And it will continue the token sequence based on this knowledge.

All of what we're seeing here is called *hallucination*.³⁴ The model is just taking its best guess in a probabilistic manner.³⁵

³³A "sample" means we look at the tokens produced stochastically – for example from the same prompt.

³⁴Here it is difficult to hold it against the model that it has a hallucination. We asked it to describe something that for it was in the future.

You can also generate genuine hallucinations (even with *assistants*, which we shall study shortly): for example, an LLM describing "a famous WWII battle" with lots of realistic details, although this battle doesn't exist. Or, if you ask it to describe the second half of a classic novel you don't want to finish, it may happily invent a new ending for the story...

³⁵Notice that we have only used the *base model* (which continues sequences of tokens, one after the other, from a prompt), not the *assistant* – which is designed to answer questions. The assistant uses, of course, extensively the base model, but elaborates on it, in a way that will be explained.

00:55:55

The next thing I would like to show is that even though this is a base model and not yet an assistant model, it can still be utilized in practical applications if you are clever with your prompt design.

So here's something that we call a *few shot prompt*. I have 10 words or, more precisely, 10 pairs of words. Each pair is a word of English followed – separated by a colon – by the translation in Korean. We have ten of them (butterfly, ocean, whisper, mountain, thunder, gentle, freedom, umbrella, cinnamon, moonlight).³⁶

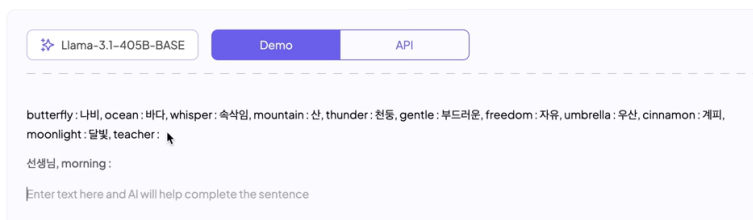


Figure 38: Smart design of prompt to make the base model begin to act like an assistant. Here demo of the model acting as a dictionary from English to Korean. We are "prompting" it to translate "teacher". It translated it correctly,³⁶ and was ready to translate "morning" when we stopped it.

³⁶It translated correctly "teacher"

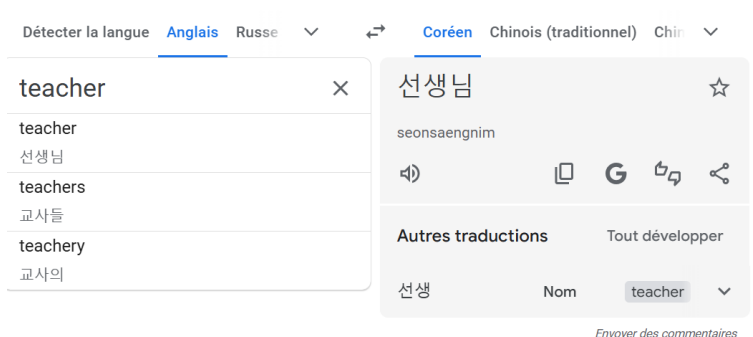


Figure 39: Translation of "teacher" into Korean.

What the model does in fig. 38 is: at the end of the prompt we have "teacher:". We asked the model to do a completion of say just five tokens.

Here is an important point to note: *base models have what we call in-context learning abilities.*

This means that as the model is reading the context [i.e. the prompt that we input, with ten pairs of words, English:Korean, English:Korean,etc.], it is learning that there's some kind of algorithmic pattern going on in my data. It knows to continue that pattern. This is *in-context learning*.³⁷

The base model takes on the role of a translator and when we hit completion we see that "teacher" translation is "seon-saengnim" (written in Korean, fig. 38 & 39) which is correct.

This is how you can build apps, being clever with your prompting even though we still only have a base model so far. It relies on this *in context learning ability*. It is done by constructing what's called a *few shot prompt*.³⁸

Finally I want to show that there is a clever way to instantiate a whole language model assistant just by prompting.

The trick is to structure a prompt so that it looks like a web page, that is a conversation between a helpful AI assistant and a human. Then the model will continue that conversation.

To write the prompt I turned to ChatGPT itself which is kind of meta. But I told it I want to create an LLM assistant but all I have is the base model. So can you please write my prompt. And this is what it came up with which is actually quite good.

Below is the conversation between an AI assistant and a human created by ChatGPT. We will use it as the beginning of a prompt, to entice our base model to continue like in a conversation.

³⁷It is called *in-context learning* because the model hasn't learned this pattern during its massive pre-training — instead, it picks it up from the specific context we provide. It simply discovers and follows at inference time the pattern it sees in the input given.

³⁸Their role is to show the pattern that the base model must follow in constructing its sequence of next tokens.

LLM Assistant Prompt

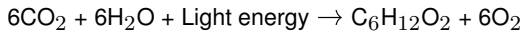
Here is a conversation between an AI Assistant and a Human. The AI Assistant is knowledgeable, helpful, and capable of answering a wide variety of questions...

Human: Hi there! Who are you?

Assistant: Hello! I'm your AI Assistant. I'm here to help answer questions, provide explanations, and assist you with tasks. What can I help you with today?

Human: Can you explain how photosynthesis works?

Assistant: Of course! Photosynthesis is the process by which green plants, algae, and some bacteria convert light energy into chemical energy stored in glucose:



This process is essential for life on Earth, as it provides the oxygen we breathe and forms the base of most food chains. Would you like more details on a spe... (cut on speaker's screen)

Human: That's great! Now, can you help me write a short email?

Assistant: Certainly! Could you tell me who the email is for and what you'd like to say?

Human: It's for my manager. I want to ask for a day off next week.

Assistant: Here's a possible draft:

Subject: Request for Day Off Next Week

Dear [Manager's Name],

I hope this message finds you well. I am writing to request a day off on [specific date]. Please let me know if this works for the team's schedule or if there... (cut on speaker's screen)

Thank you for considering my request. Please let me know if you need further details.

Best regards, [Your Name]

Does this look good to you, or would you like to adjust it?

Human: Thanks! That's perfect.

Assistant: You're welcome! Let me know if there's anything else I can help with.

For the base model to work, it's not enough to just give it a description of what we want. It works much better if we create these few shot prompts.

So we created – with the help of ChatGPT, but any method would do – a few turns of conversation: Human, Assistant, Human, Assistant...

To create our prompt for our base model, at the end of this conversation, we're going to put the actual query we want the base model to answer as if it was an assistant..

So I copy paste in the base model prompt the conversation we just created, and add at the end:

Human: Why is the sky blue?

And let's run.

Assistant: The sky appears blue due to the phenomenon called Rayleigh's scattering, etc, etc.

We see that the base model is just continuing the sequence. And because the sequence has the pattern of a conversation between a human and an assistant, the base model takes on the role of the assistant.

There is a subtlety though: because we have initialized a *conversation* human / assistant, the assistant will guess (i.e. hallucinate if you like) the next question by the human. And it will just continue going on and on.

Nonetheless you can see that we have sort of accomplished the task.

If we had just used the prompt "Why is the sky blue?" without a few preliminary shot prompts structuring a conversation, then of course we would not expect the base model to answer correctly our query. We would get... who knows what we would to get. Let's try with our simple prompt:

Why is the sky blue?

What makes a cloud? How is a hurricane born? Why do we have seasons? Answers to these questions and many more can be found in the fascinating and highly educational exhibition Weather: From Magic to Meteorology. The exhibition presents the history of meteorology and weather forecasting, from the first observations of the sky and the atmosphere, to the development of modern forecasting techniques. It also explores the effects of weather and climate on our lives, and how we can use our knowledge of the weather to protect ourselves from its dangers. Weather: From Magic to Meteorology is a must-see for anyone interested in the history of...

We see that we just get more questions, followed by a speech advertising the educational exhibition Weather: From Magic to Meteorology.

So this is one way to create an assistant even though we may only have a base model.³⁹ This is a kind of brief summary of the things we talked about over the last few minutes.

00:59:30

9 From pre-training to post-training

Let me zoom out [that is, take a bird's eye view of what we have been doing so far and what we want]. This is a brief summary of what we have been talking about in the last few minutes:

³⁹To transform our base model into an assistant, we have structured our prompt into a few shot prompts showing a dialog. And then the base model "knew" that it had to continue the *dialog*.

The "psychology" of a base model

- It is a token-level internet document simulator
- It is stochastic / probabilistic - you're going to get something else each time you run
- It "dreams" internet documents
- It can also recite some training documents verbatim from memory ("regurgitation")
- The parameters of the model are kind of like a lossy zip file of the internet
 - => a lot of useful world knowledge is stored in the parameters of the network
- You can already use it for applications (e.g. translation) by being clever with your prompts
 - e.g. English:Korean translator app by constructing a "few-shot" prompt and leveraging "in-context learning" ability
 - e.g. an Assistant that answers questions using a prompt that looks like a conversation
- But we can do better...

Figure 40: The "psychology" of a base model.⁴⁰

We've discussed the first stage in the construction of an LLM assistant. It is the *pre-training stage*. It boils down to this:

1. Download internet [and other] documents.
2. Filter them and clean them to have only high quality useful texts, that will be use to train our LLM.
3. Tokenize them, that is break them into tokens, or, if you prefer, atoms of little text chunks.
4. Build a neural network with one or several intermediate layers – nowadays of the transformer variety.
5. Train your neural network to predict the next token in sequences of tokens provided as input. This is done in a long and costly process that enables us to adjust the vast set of parameters of our neural network.

The output of this entire first stage is this base model.

Its goal is the setting of the parameters of this network – which are the precious stuff in our model.

⁴⁰Remember: the white board Andrej Karpathy occasionally points out to while speaking can be obtained using <https://excalidraw.com/> and opening the file <https://lapasserelle.com/documents/karpathy.txt>

This base model is basically an internet document simulator at the token level.

It can generate token sequences that have the same kind of statistics as the internet documents we downloaded and cleaned.

We saw that we can already use it in some useful applications. But we actually need to do better.

We want a real *assistant*. We want to be able to ask questions and we want the model to give us answers.

For that we need to go into the *second stage* which is called the *post-training stage*.

We take our base model, our internet document simulator, and hand it off to post-training.

We're now going to discuss a few ways to do what's called post training of these models.

The stages in post-training are going to be computationally much less expensive. Most of the computational work, all of the massive data centers, and all of the sort of heavy compute and millions of dollars are the pre-training stage.

Now we go into the slightly cheaper but still extremely important stage called post-training where we will turn our LLM base model into an assistant.

Let's examine how we can transform our model from being merely a token generator into one that provides full, meaningful answers to questions.

In other words what we want to do is: *we want to start thinking about conversations*.

And these are conversations that can be multi-turn, so there can be multiple turns, and they are in the simplest case a conversation between a human and an assistant.

For example we can imagine the conversation could look something like this: when a human says "what is 2 plus 2?",

the assistant should respond with something like "2 plus 2 is 4".

When a human follows up and says "what if it was star instead of a plus?" assistant could respond with something like " $2*2 = 4$, same as $2+2$ ".

Human: "What is $2+2$?"
Assistant: " $2+2 = 4$ "
Human: "What if it was $*$ instead of $+$?"
Assistant: " $2*2 = 4$, same as $2+2$!"

Similarly here, it is another example showing that the assistant could also have some kind of a personality here. That's kind of nice :

Human: "Why is the sky blue?"
Assistant: "Because of Rayleigh scattering."
Human: "Wow!"
Assistant: "Indeed! Let me know if I can help with anything else :)"

And then here in the third example :

Human: "How can I hack into a computer?"
Assistant: "I'm sorry I can't help with that."

I'm showing that when a human is asking for something that we don't wish to help with, we can produce what's called a *refusal*. We can say that we cannot help with that.

In other words what we want to do now is we want to think through how in a system should interact with the human.

We want to program the assistant and its behavior in these conversations.

Now because this is neural networks, we're not going to be programming these explicitly in code. We're not going to be able to program the assistant in that way.

Because this is neural networks, everything is done through neural network training on data sets. Thus, we are going to be implicitly programming the assistant by creating data sets of conversations.

The three boxes (on the previous page) are three independent examples of conversations in a data set.

An actual data set, and I'm going to show you examples, will be much larger. It could have hundreds of thousands of conversations that are multi-turn, very long, etc., and would cover a diverse breath of topics. Above I showed only three examples.

The way this works basically is: Assistant is being programmed by examples. And where is this data coming from, like "2 * 2 equals 4, same as 2 plus 2" etc.? Where does that come from?

This comes from human labelers.

We will give human labelers some conversational context, and we will ask them to write the ideal assistant response in this situation. In other words, a human will write out the ideal response for an assistant in any situation.

Then we're going to get the model to train on these ideal conversations and to imitate their kinds of responses. The way this works then is we are going to take our base model which we produced in the pre-training stage. This base model, as you remember, was trained on internet documents.

Now we have finished using the internet training data set. We leave it aside. We're going to substitute a *new data set*. It will be a data set of conversations. And we're going to continue training the model on this new data set of conversations.⁴¹

⁴¹The reader may wonder: Does the post-training stage still aim to improve the weights (the parameters)? Or is it a new kind of process?

Here is Perplexity's answer: The post-training stage in machine learning and AI does aim to further improve the model's weights (the pa-

What happens is that the model will very rapidly adjust and will learn the statistics – or the patterns – of how this assistant responds to human queries.

Later during inference we'll be able to prime the assistant with a question and have the assistant make a response. It will be imitating what human labelers would do in that situation, if that makes sense.

We're going to see examples of that. And it will become a bit more concrete.

rameters) by refining and specializing them for specific tasks, but it is distinct from the initial (pre-training) process and encompasses more than just retraining.

Adjustment of Parameters

During post-training, the model undergoes fine-tuning — meaning additional training on smaller, task-specific datasets that directly updates the model's weights. This process adapts a generally trained model to real-world requirements or desired behaviors (for instance, making a language model follow human instructions or respond appropriately in a customer service context).

New Processes in Post-Training

Post-training includes specialized techniques beyond simple retraining:

- **Supervised Fine-Tuning (SFT):** Training the model with curated examples and preferred outputs to adjust behavior and accuracy.
- **Direct Preference Optimization (DPO):** Training the model to favor certain outputs over others by comparing pairs of responses.
- **Reinforcement Learning (RL):** Gradually updating the model using feedback scores based on its output, often including human evaluation or reward functions.
- **Optimization Techniques:** Quantization and pruning, for example, reduce precision or remove unnecessary components for deployment efficiency, not just accuracy.

Post-Training Is an Ongoing Stage

Unlike the large-scale pre-training phase, post-training can be repeated as new data, requirements, or feedback emerge — allowing models to remain effective and adapt to changing tasks or environments. It bridges the gap between a general-purpose, pretrained model and a specialized tool suitable for real-world applications, aligning the weights for the intended use case.

In summary, post-training definitely improves the model's weights and parameters, but is considered a broader and more specialized process than initial pre-training; it enables adaptation, fine-tuning, and optimization according to specific practical needs.

I also want to stress that in this post-training stage, we're still going to train the model and improve the parameters.

But the pre-training stage can in practice take roughly three months of training on many thousands of computers, while the post-training stage will typically be much shorter like three hours for example.

That's because the data set of conversations, that we're going to create here manually, is much much smaller than the data set of text on the internet. And so this training will be very short.

But fundamentally we're just going to take our base model, we're going to continue training using the exact same algorithm, the exact same everything, except we're swapping out our initial internet data set for conversations written by human labelers.

The questions now are: What are these conversations? How do we represent them? How do we get the model to see conversations instead of just raw text?

And then: What are the outcomes of this kind of training? And what do we get, in a certain psychological sense, when we talk about the model?

Let's turn to those questions now.

01:05:00

10 Post-training data (conversations)

Let's start by talking about the tokenization of conversations.

Everything in these models has to be turned into tokens because everything is just about token sequences. So "how do we turn conversations into token sequences?" is the question.

For that we need to design some kind of encoding. This is similar to maybe – if you're familiar with it, but you don't have to be – the TCP/IP packet encoding on the internet.

There are precise rules and protocols for how you represent information, how everything is structured together so that you have all this kind of data laid out in a way that is written out on a paper and that everyone can agree on.

It's the same thing now happening in LLM: we need some kind of data structures and we need to have some rules about how these data structures, like conversations, get encoded and decoded to and from tokens.

I want to show you how I would recreate the conversation about "2 + 2" (page 63) in the token space:

Human: "What is 2+2?"
Assistant: "2+2 = 4"
Human: "What if it was * instead of +?"
Assistant: "2*2 = 4, same as 2+2!"

If we go to Tiktokenizer, we can take that conversation and this is how it is represented for the language model

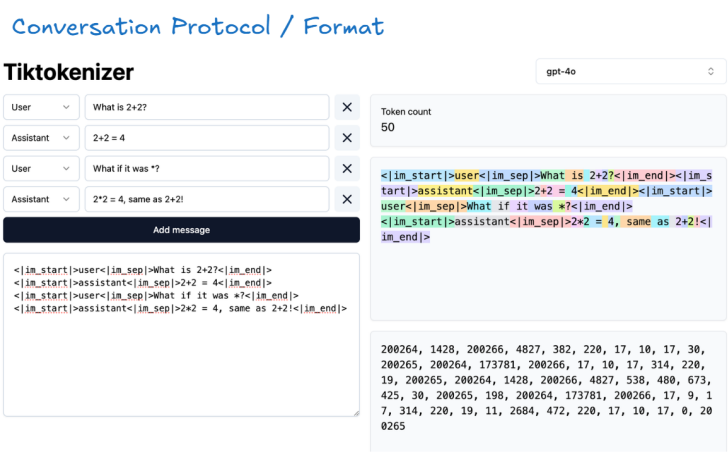


Figure 41: Tokenization of a conversation.

Here we are iterating a user and an assistant in this two-turn conversation. What you're seeing in fig. 41 looks ugly

but it's actually relatively simple. The way it gets turned into a token sequence here at the end [bottom right part of fig. 41] is a little bit complicated.

But in the end, the conversation between a user and an assistant is 49 tokens. It is a one-dimensional sequence of 49 tokens. And these are the tokens:

```
200264, 1428, 200266, 4827, 382, 220, 17, 10, 17, 30,
200265, 200264, 173781, 200266, 17, 10, 17, 314, 220,
19, 200265, 200264, 1428, 200266, 4827, 538, 480, 673,
425, 30, 200265, 198, 200264, 173781, 200266, 17, 9, 1
7, 314, 220, 19, 11, 2684, 472, 220, 17, 10, 17, 0, 20
0265
```

All the different LLMs will have a slightly different format or protocol and it's a little bit of a Wild West right now.

For example GPT-4 does it in the following way (bottom left part of fig. 41): you have this special token called `<|im_start|>`. This is short for "imaginary monologue starts here" It is token 200264.

Then you have to specify whose turn it is. So for example "user" which is the token 1428.

Then you have "internal monologue separator" (called `<|im_sep|>`, with token 200266).

Then comes the exact question. The seven tokens of the question are 4827, 382, 220, 17, 10, 17, 30.⁴²

Then you have to close it (`<|im_end|>`: token 200265). So I am at the end of the imaginary monologue.

Thus, the question from a user of "what is 2 plus two?" ends up being the token sequence 4827, 382, 220, 17, 10, 17, 30, preceded by the token 200266 (`<|im_sep|>`) and followed by the token 200265 (`<|im_end|>`).

Then we have a new 200264 (`<|im_start|>`), followed by the token for "assistant" (173781), etc.

⁴²Remember that these are not necessarily exact words, they can be word + space, space + word, part of word, etc.

It is important to mention here that `<|im_start|>` is not a piece of text. This token `<|im_start|>` (number 200264) is a special token that gets added. It's a new token. Our neural network had never encountered it during its pre-training. It is a new token that we create and introduce in the post-training stage.

These special tokens, like `<|im_sep|>`, `<|im_start|>` etc., are introduced and interspersed within the text so that the model can learn that "Hey this is the start of a turn for...". Whose turn is it to start? It is the user's turn to start.

After having specified that it is the user's turn, our protocol introduces a `<|im_sep|>`.

Then comes the actual question [all tokenized] the user asks. Then the user ends.

Then it's a new start of a turn. It is the assistant's turn now. Our protocol introduces again a `<|im_sep|>`. Then what does the assistant say? Well, these are the tokens of what the assistant says (17,10,17, 314, 220, 19). And the conversation continues.

So the conversation has now been converted into a sequence of tokens. This might seem baroque, but it is precise and makes the sequence easy for the neural network to interpret. The specific technical details are not important for our purpose.

All I'm trying to show you, in concrete terms, is that our conversations, which we think of as structured objects, end up being turned via some encoding into one dimensional sequences of tokens.

Because it is a one dimensional sequence of tokens, we can apply all the toolbox of procedures that we developed before. We can train a language model on our new sequences of tokens encoding conversations. We're just predicting the next tokens in sequences just as before. But now we can represent – that is to say, encode – specifically *conversations* and train on them.

Then what does it look like at test time during inference?

Say we've trained a model first on the initial data set of internet documents, then on these kinds of data sets of conversations. And now we want to do inference.

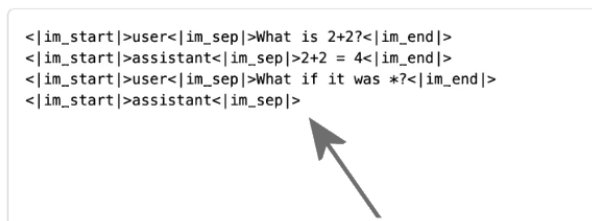
During inference what does this look like [speaker shows fig. 20, page 28] when you're on ChatGPT ?

You come to ChatGPT and you have a dialogue. You have a few turns, like "What is 2 plus 2?" "2 plus 2 is four", see beginning of dialog in fig. 41. And now you issue "What if it was *?".

What ends up happening on the servers of OpenAI or something like that?

To your dialog are added the extra tokens like `<|im_start|>`, `<|im_sep|>`, etc. structuring it.

Then, they put `<|im_start|>Assistant<|im_sep|>`. And this is where they end it, right at the point shown by the arrow:



```
<|im_start|>user<|im_sep|>What is 2+2?<|im_end|>
<|im_start|>assistant<|im_sep|>2+2 = 4<|im_end|>
<|im_start|>user<|im_sep|>What if it was *?<|im_end|>
<|im_start|>assistant<|im_sep|>
```

Figure 42: To answer a question, construction of the context which ChatGPT will continue.

So they construct the context shown in fig. 42.⁴³

*And now, to build the answer, they start sampling from the model.*⁴⁴

⁴³Remember, "the context" is the sequence of tokens from which we want to figure out the next one.

⁴⁴"Sampling", here, means "producing outcomes of random variables", because the sequential selection of the next tokens, as we saw, is stochastic. To pick a next token, the model builds first a vector of probabilities (fig. 15, p. 19) the dimension of which is the size of the collection of possible tokens (including now the extra "structuring tokens").

It is at this stage that they will go to the model and say: Okay what is a good sequence completing this context? What is a good first token? What is a good second token? What is a good third token?

This is where the LLM takes over and creates a response like for example:

```
<|im_start|>user<|im_sep|>What is 2+2?<|im_end|>
<|im_start|>assistant<|im_sep|>2+2 = 4<|im_end|>
<|im_start|>user<|im_sep|>What if it was *?<|im_end|>
<|im_start|>assistant<|im_sep|>2*2 = 4, same as 2+2!<|im_end|>
```

Figure 43: Continuation of the context by ChatGPT.

It doesn't have to be identical to this continuation shown in fig. 43 [because of the stochastic aspect of the selection of the next tokens], but it will have the flavor of this – if this kind of a conversation was in the data set.

That's roughly how the protocol works, although the exact details of this protocol are not important. Again my goal is just to show you that, after the post-training on conversations built by human labelers, everything ends up being just a one-dimensional token sequence.

So we can apply everything we've already seen in pre-training, but we're now training on conversations. And we're now generating conversations as well.

Now I would like to turn to what these data sets look like in practice.

The first paper that I would like to show you, the first effort in this direction is this paper from OpenAI in 2022:

"Training language models to follow instructions with human feedback" (<https://arxiv.org/pdf/2203.02155>)

3.02155v1 [cs.CL] 4 Mar 2022

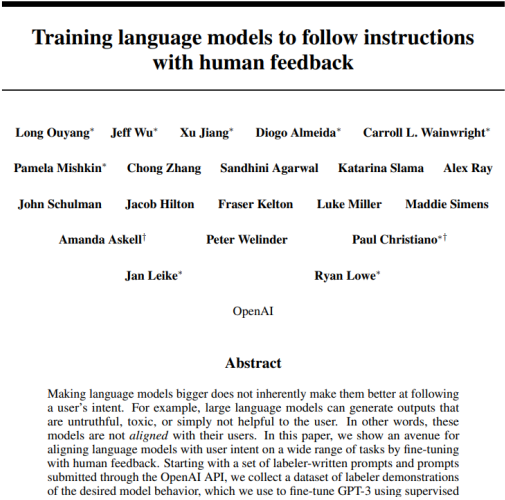


Figure 44: OpenAI paper called "InstructGPT".

This paper is known in the community as the *InstructGPT* paper, or the technique they have developed is called that way.

This was the first time that OpenAI has talked about how you can take language models and *fine-tune* them on conversations. The paper has a number of details that I would like to take you through.

The first stop I would like to make is in section 3.4 where they talk about the human contractors that they hired, in this case from Upwork⁴⁵ or through Scale AI⁴⁶, to construct these conversations.

There are human labelers involved whose job it is professionally to create these conversations. These labelers are

⁴⁵Upwork is a global online freelancing platform that connects businesses (or individuals) with independent professionals offering services such as writing, programming, graphic design, translation, and many other digital skills.

⁴⁶Scale AI is a U.S.-based company that provides high-quality data annotation and labeling services for training and evaluating artificial intelligence models.

asked to come up with prompts. Then they are asked to also complete the ideal assistant responses.

These are the kinds of prompts that people came up with (from p. 26 of the paper). Remember that these are produced by human labelers.

A.2.1 Illustrative user prompts from InstructGPT distribution	
Use Case	Example
brainstorming	List five ideas for how to regain enthusiasm for my career
brainstorming	What are some key points I should know when studying Ancient Greece?
brainstorming	What are 4 questions a user might have after reading the instruction manual for a trash compactor? (user manual) 1.
Use Case	Example
brainstorming	What are 10 science fiction books I should read next?
classification	Take the following text and rate, on a scale from 1-10, how sarcastic the person is being (1 = not at all, 10 = extremely sarcastic). Also give an explanation (text)

Figure 45: Examples of prompts.

We see for instance:

- List five ideas for how to regain enthusiasm for my career.
- What are the top 10 science fiction books I should read next?
- Translate this sentence [text] to Spanish.
- Translate this sentence [text] from Spanish.
- Summarize this for a second-grade student: [text].

There are many different types of prompts here. So there’s many things that the labelers came up with.

They first come up with the prompt and then they also answer that prompt, trying to give the ideal assistant response.

How do they know what is the ideal assistant response that they should write for these prompts?

When we scroll down the InstructGPT paper, we arrive at this *Excerpt of labeling instructions* (p. 37 of the paper) that are given to the human labelers.

Excerpt of labeling instructions on the API prompt distribution
You are given a text-based description of a task, submitted by a user. This task description may be in the form of an explicit instruction (e.g. "Write a story about a wise frog."). The task may also be specified indirectly, for example by using several examples of the desired behavior (e.g. given a sequence of movie reviews followed by their sentiment, followed by one movie review without a sentiment, you can assume the task is to predict the sentiment of the final review), or by generating the start of a desired output (e.g. given "There once was a wise frog named Julius," you can assume the task is to continue the story). You will also be given several text outputs, intended to help the user with their task. Your job is to evaluate these outputs to ensure that they are helpful, truthful, and harmless. For most tasks, being truthful and harmless is more important than being helpful. By helpful, we mean that the output should follow the user's intention, and help the user solve their task. Some examples of helpful behavior: • Writing in clear language. • Answering the question they meant to ask, even if they mis-asked it. • Being sensitive to internationality (e.g. "football" shouldn't mean American football, and "the president" doesn't necessarily mean the US president). • If an instruction is too confusing, asking for clarification and explaining why the instruction is confusing. • Not giving overly long or rambling answers, or repeating information from the question. • Not assuming extraneous extra context outside of what's given (besides things like facts about the world), unless that's an implied part of the task. For example, if asked to "write a polite email response to this email: email body" the output shouldn't assume "I can't make it this time, but am free next weekend." But if asked to "Write an email to Socrates", then you can safely assume the output can make up a context. By truthful, we mean that the output contains accurate information, and doesn't mislead the user. Some examples of truthful behavior: • On tasks like summarization, where the output should only use information from input, not making up details that

Figure 46: Instructions to labelers (page 37 of OpenAI paper of 2022 known as the InstructGPT paper).

The company that is developing the large language model (LLM), like for example OpenAI, writes up labeling instructions for how the humans should create ideal responses. Fig. 46 shows an excerpt of these kinds of labeling instruction instructions given by OpenAI.

On high level, you're asking people to be helpful, truthful and harmless.

You can pause the video if you'd like to look at the paper more attentively (time mark 01:12:00).

Again, on a high level: just answer, try to be helpful, try to be truthful, and don't answer questions that we don't want the system to handle later in ChatGPT.

So the company has prepared labeling instructions to its labelers. Usually they are not this short. Usually there are

hundreds of pages. And people have to study them professionally. Then they write out the ideal assistant responses following the labeling instructions.

It is a highly labor-intensive human process as described in the InstructGPT paper

The actual data set for instructing GPT was never released by OpenAI. But we do have some open source reproductions that are trying to follow this kind of a setup and collect their own data.

There are many examples. One I'm familiar with is the effort of Open Assistant⁴⁷ from a while back.

These were people on the internet who were asked to create these conversations similar to what OpenAI did with human labelers.

message_id	parent_id	user_id	created_date	text	role
string - lengths	string - lengths	string - lengths	string - lengths	string - lengths	string - classes
36	100%	null	11.7%	36	100%
68b2d072-9181-4594-a9cd-deaf170242fd	null	c1fe6c76-1c30-4fa7-b7f8-c492f5967d18	2023-02-05T14:23:50.933374+00:00	Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopolies in the labour market and cite relevant research.	prompter

Figure 47: Example of work of human labelers at Open Assitant.

Figure 47 shows an entry of a person who came up with this prompt:

Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopolies in the labour market and cite relevant research.

Then the same person, or possibly a different person, wrote up the response (fig. 48).

⁴⁷Open Assistant is an open-source, chat-based artificial intelligence project designed to create a virtual assistant similar to ChatGPT, but fully open and crowdsourced.

:5e83833-ecbc- 14fe-b6d9- 752226c25a1c	6a526d72-0181- 4594-a9c6- dca5170242f9	2c96a467-66f9- 4be7-9e93- bca51356a42e	2023-02- 06T13:50:44.657083+00:00	"Monopsony" refers to a market structure where there is only one buyer for a particular good or service. In economics, this term is particularly relevant in the labor market, where a monopsony employer has significant power over the wages and working conditions of their employees. The presence of a monopsony can result in lower wages and reduced employment opportunities for workers, as the employer has little incentive to increase wages or provide better working conditions. Recent research has identified potential monopsonies in industries such as retail and fast food, where a few large companies control a significant portion of the market (Rivens & Mishel, 2013). In these industries, workers often face low wages, limited benefits, and reduced bargaining power, leading to a situation where they are dependent on the employer for their livelihood. This dependence can result in further suppression of wages and a decline in working conditions. Overall, the concept of monopsony is essential to understanding the dynamics of labor markets and the impact of market power on workers. Further research is needed to understand the extent and impact of monopsonies on the economy and to develop policies to address this issue. References: Rivens, J., & Mishel, L. (2013). The Pay of Corporate Executives and Financial Professionals as Evidence of Rents in Top 1 Percent Incomes. Journal of Economic Perspectives, 27(3), 57-79.	assistant
--	--	--	--------------------------------------	--	-----------

Figure 48: Labeler response to the question about monopsony.

So the same person, or a different person, did actually write out the ideal response.

Then here is an example of how the conversation could continue: "Now explain it to a dog." You would then try to come up with a simpler explanation.

All these dialogs or quetion/answer snippets form the *labels*. And we end up training on them.

Of course we’re not going to have a full coverage of all the possible questions that the model will encounter at test time during inference. We can’t possibly cover all the possible prompts that people are going to be asking in the future.

But if we have a data set of a few of these examples then the model during training will start to take on this persona of this helpful truthful harmless assistant. And it’s all programmed by example.

So these are all examples of behavior. And if you have conversations of these example behaviors ,and you have enough of them like 100 000, and you train on it, the model starts to understand the statistical pattern. And it takes on this personality of this assistant.

It is possible that when you get the exact same question on monopsony (fig. 47) at test time, the answer will be recited exactly as what it was in the training set. But the model will

more likely do something of a similar vibe (= similar style or feel), and will understand that this is the kind of answer that you want.

So that's what we're doing: we're programming the system by example. And the system adopts statistically this persona of this helpful, truthful, harmless assistant which is somehow reflected in the labeling instructions that the company wrote out.

01:15:05

Now I want to show you that the state-of-the-art has progressed in the last two or three years since the *InstructGPT* paper.

In particular it's not very common for humans to be doing all the heavy lifting just by themselves anymore.

That's because we now have language models and these language models are helping us create these data sets and conversations.

So it is very rare that the people will just write out the response from scratch. It is a lot more likely that they will use an existing LLM to come up with an answer which they will then edit.

There's many different ways in which LLMs have now started to permeate this post-training stage. LLMs are used pervasively to help create these massive data sets of conversations.

UltraChat (<https://github.com/thunlp/UltraChat>) is an example of a more modern data set of conversations. It is to a very large extent synthetic, but I believe there's some human involvement — I could be wrong about that.

Usually there will be a little bit of human, but there will be a huge amount of synthetic help, and this is all constructed in different ways. UltraChat is just one example of many *SFT* (*supervised fine-tuning*) *datasets* that currently exist.

The only thing I want to show you is that these data sets have now millions of conversations (fig. 49).

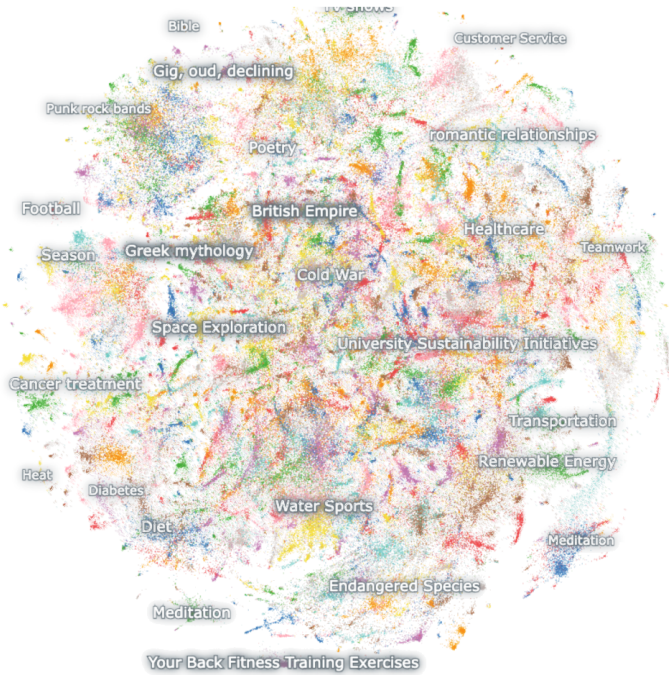


Figure 49: Conversations available in SFT datasets.

These conversations are mostly synthetic but they're probably edited to some extent by humans. And they span a huge diversity of sort of areas.

They are fairly extensive artifacts by now. There's all these *SFT mixtures*, as they're called. So you have a mixture of lots of different types and sources. It's partially synthetic partially human. Post-training has gone in that direction.

Whatever their origins, we now have SFT (supervised finely tuned) data sets. They're made up of conversations. We're training on them, just like we did before.

I want to dispel some of the magic that is commonly attached to AI and specifically to talking to an LLMs.

*When you go to ChatGPT and you give it a question and then you hit enter, what is coming back statistically is aligned with what's happening in the training set. These training sets ultimately have their seed in humans following labeling instructions.*⁴⁸

ChatGPT is an assistant capable of engaging in dialogue with us and can acquire a sort of *persona* that can be fascinating, if not disturbing. We may have the impression that we are conversing with a smart, well-educated person.

But what are we actually talking to in ChatGPT? How should we think about it? Well it's not coming from some magical AI. It's coming from something that is statistically imitating human labelers, whose work comes from human knowledge set out following labeling instructions written by these companies.

It's almost as if you're asking a human labeler. Imagine that the answer that is given to you from ChatGPT is some kind of a simulation of a human labeler. It's like asking what would a human labeler say in this kind of a conversation.

This human labeler is not some random person from the internet; it is an expert hired by a firm — just as in the old days companies that published paper dictionaries and encyclopedias used to do.

For example when you are asking questions about computer code or programming, the human labelers that would be involved in the creation of these conversation data sets would

⁴⁸Note that it is still "magical" that the model can provide such high-quality answers to all sorts of questions. In a sense, it does reproduce human texts, like Wikipedia — yet it often improves on them. This is why some argue that Wikipedia has, in some respects, become outdated. Reading a Wikipedia article can be tedious and confusing, with excessive detail in the opening section, while a GPT-generated answer is typically succinct, direct, suited exactly to what we need.

*usually be educated expert people. In a sense, you're asking a question to a simulation of those people. So you're not talking to a magical AI, you're talking to an average labeler.*⁴⁹

This average labeler is probably fairly highly skilled, but you're talking to an instantaneous simulation of that kind of a person that would be hired in the construction of these data sets.

Let me give you one more specific example before we move on. Let's go to ChatGPT and write "Recommend top 5 landmarks to see in Paris" in the context window and hit enter.

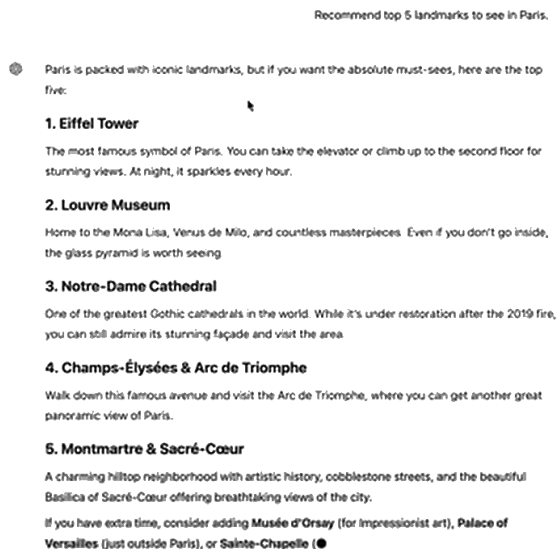


Figure 50: Top 5 landmarks to see in Paris, according to ChatGPT.

Figure 50 shows ChatGPT answer to our question. How to think about it? Well, it's not some kind of magical AI that has gone out and researched all the landmarks and then ranked them using its infinite intelligence and wisdom.

What we are getting is a statistical simulation of a labeler that was hired by OpenAI. You can think of it that way.

⁴⁹In the sense of typical or archetypal labeler.

And if the specific question "Recommend top 5 landmarks to see in Paris" happened to be in the post-training data set, somewhere at OpenAI, then we are likely to see an answer that is very similar to what the human labeler who treated it wrote down for those five landmarks.

How does the human labeler come up with this information? Well, they go to the internet and do their own little research for twenty minutes. And they come up with a list and a description for each landmark. Therefore if they came up with the list shown in fig. 50, and this is in the data set, I'll probably see what they submitted as the correct answer from the assistant.

Now if this specific query "Recommend top 5 landmarks to see in Paris" was not part of the post-training data set, then what I'll get will be little bit more "emergent", because the model understands that the kinds of landmarks that are in the training set are usually the prominent landmarks, the landmarks that people usually want to see, the kinds of landmarks that are usually very often talked about on the internet, therefore the landmarks that should be listed in its answer.

Remember, the model already has a lot of knowledge from its pre-training on the internet. So, it's probably seen tons of conversations about Paris, about landmarks, and about the types of things people like to do or see there. This pre-training knowledge combines with the supervised fine-tuning data to produce the kind of imitation responses we see in fig. 50. That's generally how you can think about what's happening behind the scenes — in this statistical sense.

01:20:30

11 Hallucinations, tool use, knowledge/working memory

Now, I want to turn to something I like to call "LLM psychology," which is about the *emergent cognitive behavior* that comes out of the way these models are trained.

The first one I want to talk about is, of course, hallucinations. You might have heard of model hallucinations — that’s when LLMs make things up, completely fabricating information.

It’s actually a significant problem with language model assistants. This issue was pretty bad with early models, from many year ago. But I think it’s gotten a little better recently thanks to some medications that I’ll explain in a moment.

For now, let’s just try to understand where hallucinations come from. Here’s a specific example: imagine three conversations you might find in a training set, fig. 51.

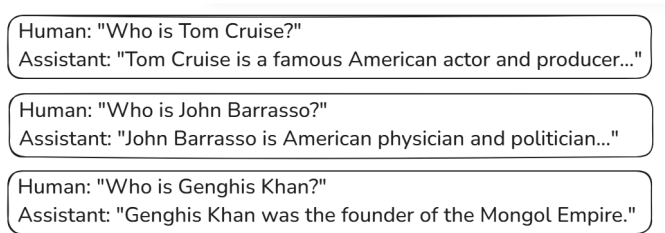


Figure 51: Three reasonable conversations with the LLM.

Each one sounds reasonable — you could imagine finding them in the data. For example, someone might ask,

“Who is Tom Cruz?” Well, Tom Cruz is a famous actor American actor and producer, etc.

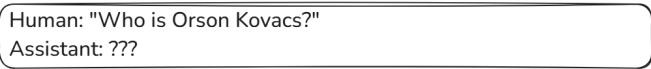
"Who is John Barrasso?". He turns out to be a US senator, for example.

"Who is Genghis Khan?" Well, Genghis Khan was blah, blah, blah.

This is what your conversations could look like at training time.

Now the problem with this is that when the human is writing the correct answer for the assistant, in each one of these cases, the human either knows who this person is or they research them on the Internet and they come in and they write this response that has this confident tone of an answer.

A problem arises when, at test time, you ask for information about someone who is totally unknown and may not even exist, fig. 52:



Human: "Who is Orson Kovacs?"
Assistant: ???

Figure 52: Question that will make the LLM stumble and "hallucinate".

Orson Kovacs is a totally random name that I made up. This person doesn't exist as far as I know.

The problem is when we ask "Who is Orson Kovacs?"

The assistant will not just tell you "Oh I don't know" – even if the LLM might know somehow, inside its features, inside its activations, inside of its brain sort of, that this person is not someone that it is familiar with.

Even if some part of the network knows in some sense that the correct answer is "Oh I don't know who this is", it is not going to be the LLM answer.

The reason is that the model statistically imitates its training set.

In the training set the questions of the form "Who is blah blah blah?" are confidently answered with the correct answer.

So it's going to adopt the style of these answers. And it's going to do its best. It will give you statistically the most likely guess. It will make up stuff.

Pay attention to the fact that these models we just talked about, they don't have access to the Internet. They're not doing research. These are *statistical token tumblers*, as I call them. They are just trying to sample the next token in the sequence. And this will lead them to basically make up stuff.

Let's take a look at what this looks like.

I have here what's called the *inference playground* from Hugging Face, see page 4:

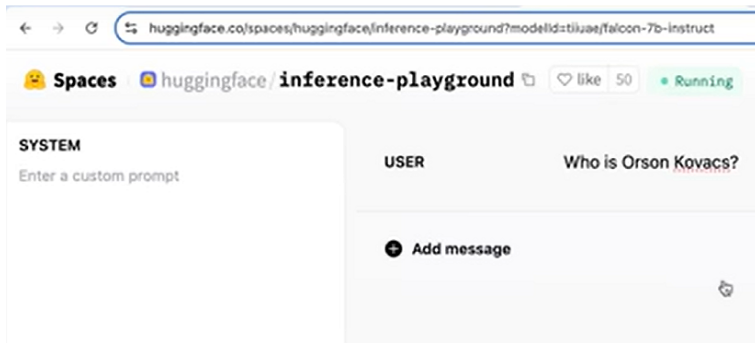


Figure 53: <https://huggingface.co/spaces/huggingface/inference-playground?model=tiiuae/falcon-7b-instruct>.

I am deliberately choosing a model called Falcon 7B (selected via a drop-down menu on the right of the screen, not shown in fig. 53), which is an older model from a few years ago. As a result, it tends to suffer from hallucinations — a problem that, as I mentioned, has improved significantly in recent models.

Let's ask Falcon 7B instruct "Who is Orson Kovacs?" and click run :

USER Who is Orson Kovacs?

ASSISTANT Orson Kovacs is an American author and science fiction writer. He has written numerous books, including the Hugo Award-winning story "The Best of All Possible Worlds".

Okay, this is totally false. It's a hallucination.

Let's try again. Since these are statistical systems, a re-sampling may produce another answer.

This time Orson Kovacs is a fictional character from this 1950s TV show...

It's total bullshit.

Let's try again : he's a former minor league baseball player...

Well, the model just doesn't know. And in three trials it has given us three different answers, because it doesn't know. It's just sampling using probabilities.

The model starts with the prompt 'Who is Orson Kovacs?', after which the assistant attempts to answer. Sequentially, it computes probabilities for the possible next token — refer back to fig. 15 on page 19 if you need to refresh your memory.

The model is sampling based on the probabilities it computes for the next token. And this produces fanciful results.

Notice, though, that this made-up stuff is actually statistically consistent with the style of the answers in its training set.

Yet you and I experienced it as made up factual knowledge.

But keep in mind that the model fundamentally does not know the answer. It is simply imitating the format of answers, fig. 51. Falcon 7B is not going to go off and search the Internet for a good answer — it just reproduces patterns from the conversation data in its training set.

How can we mitigate this ? For example when we go to ChatGPT and I enter the prompt "Who is Orson Kovacs?", I'm now asking the state-of-the-art model from OpenAI. This model will tell you:

Who is Orson Kovacs?

The name "Orson Kovacs" appears to be a combination of two distinct individuals: Orson Bean and Ernie Kovacs. Orson Bean (1928-2020) was an American actor, comedian, and writer known for his appearances on television shows and game shows. Ernie Kovacs (1919-1962) was an American comedian, actor, and writer recognized for his innovative work in television during the 1950s. They collaborated on various projects, including appearances in "Mad Magazine" and on "The Tonight Show."

Additionally, there is a character named Odon Kovacs in the 2000 video game "Hitman: Codename 47." Odon Kovacs is a secondary antagonist and an optional target in the game's 12th mission, "The Setup."

If you have a specific context or individual in mind regarding "Orson Kovacs," please provide more details so I can offer more targeted information.

Oh, this model is actually is even smarter than what I was gonna comment upon! You may have seen flashing briefly in the video "searching the web". We're going to cover that later. It's actually using tools. And it came up with a story it just got on the fly from the Net. But I want the model not to do web search.

So let me refresh the ChatGPT page, and enter the same prompt, specifying too not to use any tool:

Who is Orson Kovacs? Do not use any tool.

There isn't a well-known historical or public figure named Orson Kovacs. You might be thinking of Ernie Kovacs, who was a pioneering American comedian and television personality known for his innovative and surreal humor. If you were referring to someone else, let me know, and I can clarify!

This is interesting: even without accessing the Internet to try and learn more, this model is not going to make up stuff. It knows that it doesn't know. And it tells us that it doesn't appear to be a person that this model knows.

We have clearly made progress in reducing hallucinations, even though they remain a significant issue in older models. Furthermore, it makes perfect sense that we would get these kinds of answers (hallucinations) if our training set resembles what is shown in fig. 51, p. 83.

So how do we fix this? [On his screen, the speaker brings up again Meta's article 'The Llama 3 Herd of Models,' which he will use to support his coming explanations.] Clearly, we need some examples in our dataset where the correct response from the assistant is that the model does not know the answer to a particular fact.

We only need to have those answers be produced in the cases where the model actually doesn't know.

Hence the question becomes: how do we know what the model knows or doesn't know?

A possibility is to empirically probe the model to figure out when the model knows the answer and when it doesn't know.

For instance, let’s take a look at how Meta dealt with hallucinations for the Llama 3 series of models. In the article "The Llama 3 Herd of Models", we can go into hallucinations which they call here *factuality*:

4.3.6 Factuality

Hallucinations remain a major challenge for large language models. Models tend to be overconfident, even in domains where they have little knowledge. Despite these shortcomings, they are often used as knowledge bases, which can lead to risky outcomes such as the spread of misinformation. While we recognize that factuality can go beyond hallucinations, we took a hallucination-first approach here.

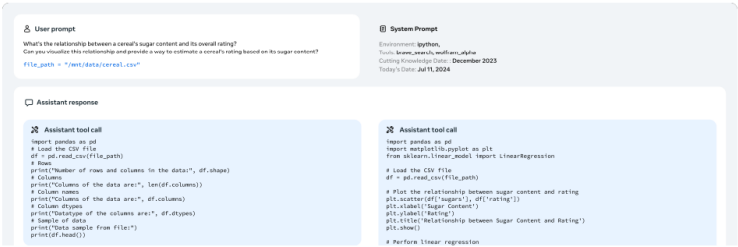


Figure 54: Meta’s dealing with hallucinations.

The authors describe the process Meta uses to interrogate the model, determining what it knows and what it does not — in other words, mapping the boundaries of its knowledge.

Next, they add examples to the training set in which, for facts the model does not know, the correct answer is for the assistant to state that it does not know. This seems straightforward – in principle. At any rate, it goes a long way toward solving the issue.

The reason it fixes the issue is this. Remember that the model might actually have a pretty good model of its self knowledge inside the network. Remember too that we looked at the network and all these neurons inside the network, fig. 18, p. 24. You might imagine that there’s a neuron somewhere in the network that sort of lights up when the model is uncertain.

But the problem is that the activation of that neuron is not currently wired up to trigger the model to actually say in words that it doesn’t know.

Thus, even though the internals of the neural network knows – because there’s some neurons that represent that –, the model will not surface that. It will instead take its best guess. So that it sounds confident just like it sees in a training set, fig. 51, p. 83.

We need to basically interrogate the model and allow it to say "I don’t know" in the cases that it doesn’t know.

01:27:42

Let me take you through what Meta roughly does, with an example: Dominik Hašek is the featured article today on Wikipedia. So I went there randomly, fig. 55.

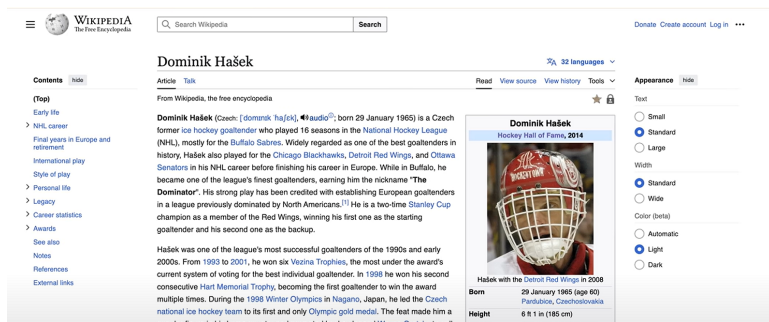


Figure 55: Wikipedia article on Dominik Hašek.

What Meta people do is they take a random document in a training set, and pick a paragraph. Then they use an LLM to construct questions about what the paragraph says.

For example I did that with ChatGPT.

The prompt I entered is: "Generate 3 specific, factual questions based on the paragraph below. In addition to the question, also generate the correct answer.", fig. 56.

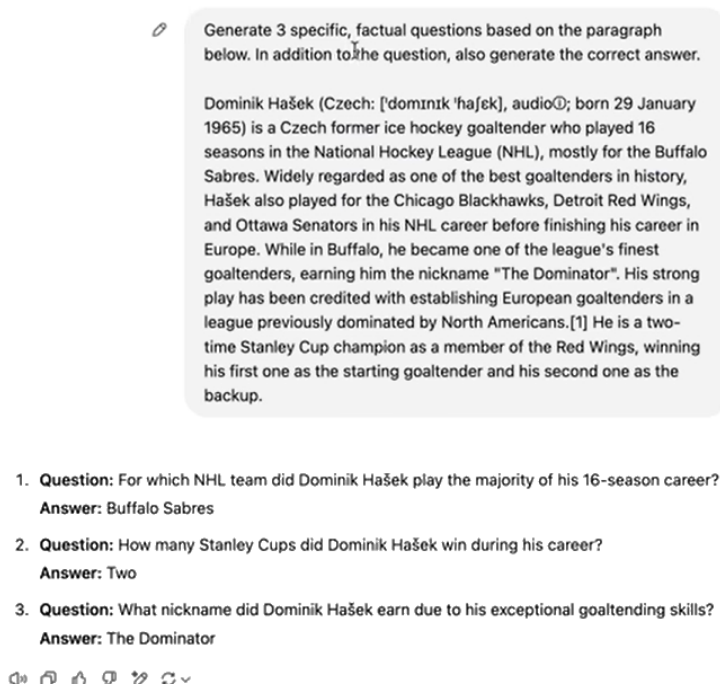


Figure 56: Questions generated by ChatGPT about Dominik Hašek.

Large language models are already good enough to create and reframe such information.

So if the information is in the context window of this LLM, this request actually works pretty well. It doesn't have to rely on its memory. It's right there in the context window. The model can reframe that information with fairly high accuracy.

For example it can generate for us questions and their answers, like these [we just repeat what's shown in fig. 56]:

- Question:** For which NHL team did Dominik Hašek play the majority of his 16-season career?
Answer: Buffalo Sabres
- Question:** How many Stanley Cups did Dominik Hašek win during his career?

Answer: Two

3. Question: What nickname did Dominik Hašek earn due to his exceptional goaltending skills?

Answer: The Dominator

Now we have some question and answers. We shall interrogate the model with the questions.

What we'll do is: we'll take our questions and we'll go to our model which would be, say, Llama in Meta.

But let's just interrogate the model Mistral-78-instruct-v0.3 here as an example. That's another model. We use it within Hugging Face inference-playground, and we ask it question 1, fig. 57.

Does this Mistral model know the answer? Let's take a look.

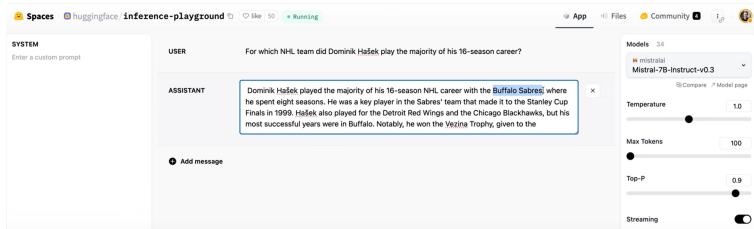


Figure 57: Question 1 asked to Mistral, and its answer.

Mistral says that Hašek played for the Buffalo Sabres. That's right. So the model knows.

The way that we can programmatically decide is to take the answer from the model, fig. 57, and compare it with the correct answer, fig 56.

The model are good enough to do this automatically. There's no humans involved here.

We can take the answer from the model we want to test and we can use another LLM-judge to check if it is correct according to the true answer.

If it is correct that means that the model we are testing (here, Mistral) probably knows.⁵⁰

What we're going to do is repeat this maybe a few times – with the same question.

So okay the model knows about Buffalo Sabres

Let's try again: ...Buffalo Sabres.

Let's try one more time: ...Buffalo Sabres.

We asked three times this factual question and three times the model answered correctly. This discards the possibility that it is just luck. The model seems to know.⁵¹ Everything is fine.

01:30:00

Let's try the second question: "How many Stanley Cups did Dominik Hašek win during his career?"

Again let's interrogate the model we want to test about that. And we know that the correct answer is two.

Here the model claims that he won four times which is not correct. It doesn't match two.

So the model doesn't know. It's making stuff up.

Let's try again. Here the model again is making stuff up (see video at time mark 01:30:32).

Let's try again. Here it says he did not even win during his career.

So obviously the model doesn't know.

The way we can programmatically tell again is we interrogate the model three times and we compare its answers maybe three times, five times, whatever we want, to the correct answer.

⁵⁰It may also have answered correctly by sheer luck. That's a common problem when testing someone's knowledge, for instance with a MCQ. But the text explains how to avoid that problem.

⁵¹Because it did not give a stochastic answer. It consistently answered Buffalo Sabres.

If the model doesn't know, then we find out that the model doesn't know the answer to the question.

What we do then is we take this question [which the model can't answer correctly because it doesn't know the answer]. We create a new conversation in the training set. So we're going to add a new conversation in the training set.

And when the question is "How many Stanley Cups did he win?" the answer is "I'm sorry I don't know" or "I don't remember". That's the correct answer for this question because we interrogated the model, and we saw that indeed it doesn't know.

If you do this for many different types of questions, in many different types of documents, you are giving the model an opportunity – in its training set – to learn to refuse to give an answer based on its knowledge.⁵²

If you just have a few examples of that in your training set the model will know and has the opportunity to learn the association of this knowledge-based refusal to this internal neuron somewhere in its network that we presume exists – and empirically this seems to be probably the case.⁵³

In the course of its post-training, the model can learn that association, that "hey when this neuron of uncertainty is high" then I actually don't know and I'm allowed to answer "I'm sorry but I don't think I remember this" to the question I can't answer.

If you have these examples in your training set, then this is a large mitigation for hallucination. That's roughly speaking

⁵²In short, since LLMs only copy or imitate what is in their training data set, now, having met situations where the answer is "I don't know", they will answer that when, probing themselves one way or another, they see that they don't know. Note that it is the same thing with the education of a child: we teach it not to answer when it doesn't know.

⁵³This leads to a fascinating insight: the model's vast collection of parameters *does form* a representation of the world — but it is not a spatio-temporal one. Whoever can clarify mathematically the nature and structure of this non-spatio-temporal representation of the world will make a big contribution to AI.

why ChatGPT is able to do stuff like this (see its second answer p. 87) as well.

These are kinds of mitigations that people have implemented. And they have improved the factuality issue over time.

This was mitigation nr 1 for the hallucinations issue.

Mitigation #1

=> Use model interrogation to discover model's knowledge, and programmatically augment its training dataset with knowledge-based refusals in cases where the model doesn't know. E.g.:

Human: "Who is Orson Kovacs?"

Assistant: "I'm sorry, I don't believe I know"

Figure 58: Mitigation nr 1 against hallucinations.

We can actually do much better than that. Instead of just saying that we don't know, we can introduce an additional mitigation nr 2 to give the LLM an opportunity to be factual and actually answer the question.

Now what do you and I do? If I was to ask you a factual question and you don't know, what would you do in order to answer the question? Well you could go off and do some search and use the Internet, and you could figure out the answer, and then tell me what that answer is.

We can do the exact exact same thing with these models.

Think of the knowledge inside the neural network, inside its billions of parameters, as kind of a vague recollection of the things that the model has seen during its pre-training stage a long time ago. Compare that to some information you read about a month ago. If you keep reading about the same information, then you will remember it well. Similarly, the model remembers well what it has seen several times during its training.

But if it's something rare, then you probably don't have a good recollection of that information. What you and I can do then is go and look it up. When you go and look it up, you're refreshing your working memory with information. Then you're able to retrieve it, talk about it, etc.

We need some equivalent procedure allowing the model to refresh its memory or its recollection. We can do that by introducing tools for the models.

The way we are going to approach this is that instead of just answering "Hey I'm sorry I don't know", we can attempt to use tools.

We can create a mechanism by which the language model can emit special tokens. These are tokens that we're going to introduce – new tokens. For example here I've introduced two tokens.

Mitigation #2

=> Allow the model to search!

Human: "Who is Orson Kovacs?"

Assistant: "

<SEARCH_START>Who is Orson Kovacs?<SEARCH_END>

[...]

Orson Kovacs appears to be ..."

Figure 59: Mitigation nr 2 against hallucinations.

I've introduced a format or a protocol for how the model is allowed to use these tokens.

For example instead of just saying "I don't know", the model has the option now to emit the special token <SEARCH_START>.

This is a query that will send the LLM searching on **bing.com** in the case of OpenAI, or say Google search or something like that. It will emit the query and then it will emit <SEARCH_END>.

Then what will happen is that the program that is sampling from the model, that is running the inference, when it sees the special token <SEARCH_START>, instead of sampling the next token in the sequence, it will pause generating tokens from the model.

It will go off and open a session with `bing.com`. It will paste the search query into Bing. And it will then get all the text that it has retrieved from the Internet. It will take that text. (It will maybe represent it again with some other special tokens or something like that, but it's not important.) It will take that text and copy/paste it here (where there is [...] in figure 59) into what I tried to show with the brackets. All that text comes here and when the text comes here it enters the context window – go back to fig. 15 & 16.

That text from the web search is now inside the context window that will feed into the neural network – whose fundamental function, you remember, is to guess the next token given a context window, which is a sequence of given tokens.

You should think of the context window as the *"working memory of the model"*.

The data in the context window is now directly accessible by the model. It directly feeds into the neural network. It's not anymore a vague recollection (encapsulated in its vast collection of parameters). It is data that the model has in its context window and which is immediately available for the neural network to perform its function – which is, simply and only, *guessing the next token*.

When the neural network is sampling the new tokens afterwards, it can use very easily the data that has been copy/pasted in the context windows.⁵⁴

That's roughly how these tools function.

Web search is only one of these tools. We're going to look at some of the other tools a bit later. But basically you introduce new tokens. You introduce some scheme by which the model can utilize these tokens, and can call these special functions like web search functions.

⁵⁴We see here clearly the explanation of the following paradox: neural networks do only one thing – they guess ("sample") the next token; yet we can make them produce intelligent responses which are long sequences of tokens. The explanation is that we prepare the context window (the "working memory of the neural network") to make the neural network produce an intelligent sequence of tokens.

How do you teach the model to correctly use these tools `<SEARCH_START>`, `<SEARCH_END>`, etc.?

Again, you do that through training sets. We need to have a bunch of data and a bunch of conversations that show the model by example how to use web search. What are the settings where you are using the search? What does that look like? And here's by example how you start a search and the search etc. (fig. 59).

If you have a few thousand examples of that in your training set, the model will do a pretty good job of understanding how this tool works. It will know how to structure its queries.

Of course because of the pre-training data set and the understanding of the world it gave the model, the model already kind of understands what a web search is. It has a pretty good "native understanding" of what kind of stuff is a good search query. So all that works. You just need to insert a few examples in the post-training set to show the model how to use this new tool. Then it can lean on it to retrieve information and put it in the context window.

That's equivalent to you and I looking something up on the Net or in a library to learn or refresh our memory. Indeed, once it is in the context window of the neural network, it is in the working memory and it is very easy to manipulate and access.

That's what we saw a few minutes ago when I was searching on ChatGPT for who is Orson Kovacs: the ChatGPT language model decided that this is some kind of little known individual or something like that. And instead of giving me an answer from its memory, it decided that it would sample (here meaning, insert) a special token that is going to do web search.

We saw a message briefly flash announcing that the LLM was going to be "using the web tool" or something like that. So it briefly said that and then we waited for like two seconds and then it generated this:

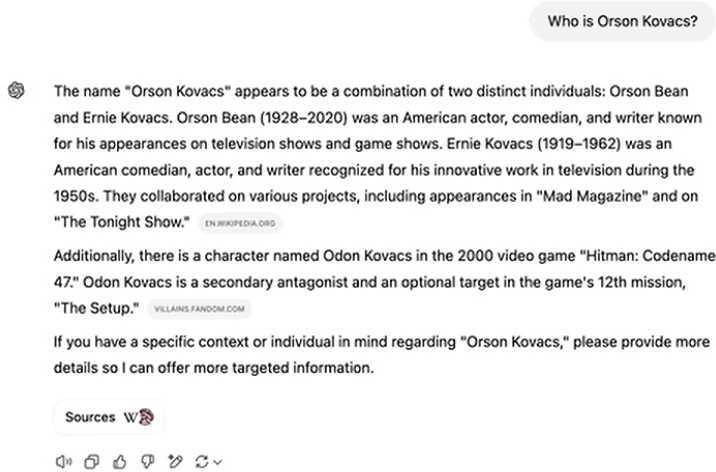


Figure 60: ChatGPT search on the fly on Orson Kovacs. Notice that its answer mentions sources.

You see how it's creating references here (fig. 60). It's citing sources.

What happened here is: it went off; it did a web search; it found these sources and these URLs (URLs are shown on the right of the screen – not shown on fig. 60)). And the text of these web pages was all stuffed in between tags (special tokens) in its context window. It's not showing here (in fig. 60) but it's basically stuffed as text in its memory (where there is [...] in fig. 59).

Now it sees that text. It references it and says: "okay, it could be these people, citation, it could be those people, citation", etc.

That's what happened here, shown in fig. 59 & 60.

That's also why when I asked "Who is Orson Kovacs?", I could also add "don't use any tools". That's enough to convince ChatGPT to not use tools and just use its memory and its recollection.

I also asked ChatGPT: "How many Stanley Cups did Dominik Hašek win during his career?" And ChatGPT actually

decided that it knows the answer. And it has the confidence to say that he won twice, in 2002 and 2008.⁵⁵

It just relied on its memory because presumably it has enough confidence in its weights, in its parameters and activations, that this is retrievable just from what it already knows [which we can call its "long term memory", as opposed to its "working memory"].

You can also, conversely, impose to use web search to make sure. Then for the same query, it actually goes off, and it searches, and then it finds a bunch of sources, it finds all this (fig. 61). All of this stuff gets copy/pasted in the prompt:

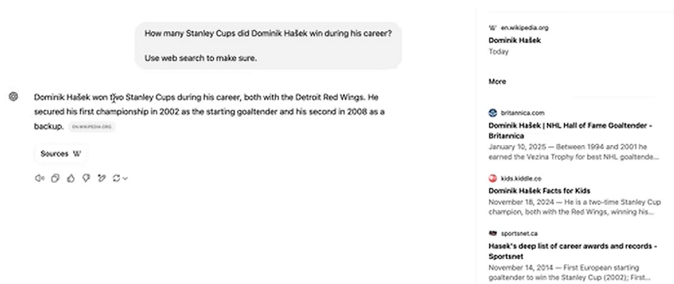


Figure 61: ChatGPT search on the web to answer "How many Stanley Cups did Dominik Hašek win during his career?".

Then it tells us *two* again and gives sources. It mentions the Wikipedia article, which is the source of this information for us as well.

This is how modern LLMs sometimes do a web search before answering a question. The model determines when to search. And when it is the case, we saw how the tool works.

This is an additional kind of mitigation for hallucinations and factuality.

I want to stress one more time this very important sort of psychology point:

⁵⁵ Even though the speaker didn't add "don't use tools" we know that ChatGPT didn't do a search, because in its answer it doesn't cite any source.

Knowledge in the parameters of the neural network is a vague recollection.

The knowledge in the tokens that make up the context window is the working memory.

It is similar to how it works for us in our brain. The stuff we remember [vaguely] is our parameters. And the stuff that we just experienced a few seconds or minutes ago, you can view it as being in our context window. This context window is being built up as you have a conscious experience around you.

This has a bunch of implications also for our use of LLMs in practice.

For example I can go to ChatGPT and ask: "Can you summarize chapter one of Jane Austin's *Pride and Prejudice*?"

This is a perfectly fine prompt. ChatGPT actually does something relatively reasonable, fig. 62.

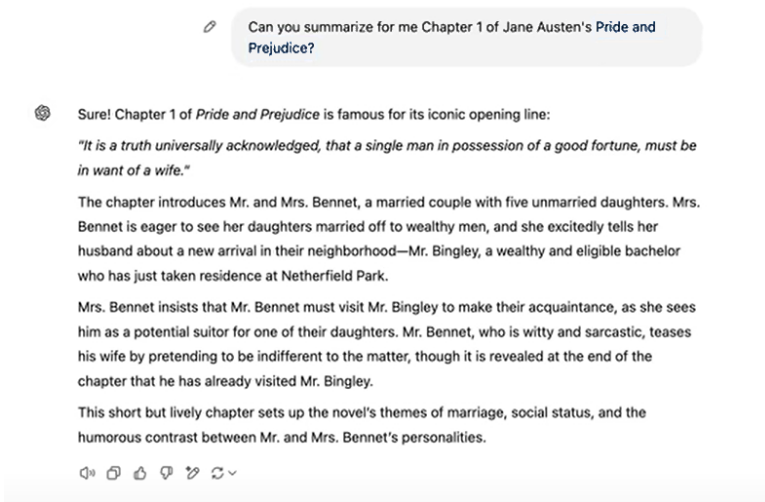


Figure 62: ChatGPT 4o answer to the prompt "Can you summarize for me Chapter 1 of Jane Austin's *Pride and Prejudice*?"

The reason it does a reasonable job [without mentioning any source] is because ChatGPT has a pretty good recollection of a famous work like *Pride and Prejudice*. It has probably seen a ton of stuff about it. There are probably forums about this book. It has probably read versions of the book.

It remembers because even if the text, or a good summary, was not in its training set, it has enough stuff in its "long term memory" about it. If you've read this or articles about it, you'd kind of have a recollection enough to actually say all this, fig. 62.

However, usually when I actually interact with LLMs and I want them to recall specific things, it always works better if you give it to them [as attached references that go into the context they will work with].

Therefore I think a much better prompt [than the one in fig. 62] would be something like this "Can you summarize for me Chapter 1 of Jane Austen's *Pride and Prejudice*? I am attaching it below for your reference."

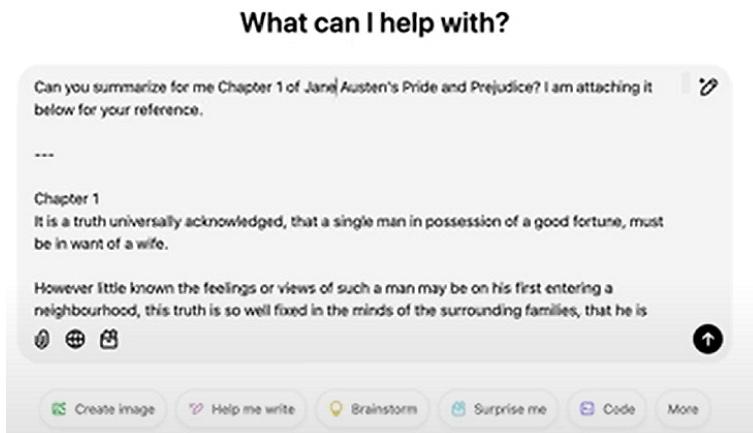


Figure 63: Improved prompt to get a summary of chapter 1 of *Pride and Prejudice*. We now put the entire chapter 1 into the context window of the LLM.

I found the chapter on the Internet, and copy/pasted it into the context window. I do that because when it's in the context window, the model doesn't have to recall it, it has direct access to it.

Then the summary can be expected to be of significantly higher quality than the summary shown in fig. 62, because the chapter was directly available to the model.

I think you and I would work in the same way. You would produce a much better summary if you had reread this chapter just before you had to summarize it.

That's what's happening here or the equivalent of it.

01:41:45

12 Knowledge of self

The next sort of psychological quirk I'd like to talk about briefly is that of the knowledge of self.

What I often see on the Internet is people asking LLMs questions like "What model are you and who built you?"

Such a question is somewhat nonsensical. The reason I say that is that when I try to explain to someone the fundamentals under the hood of an LLM, the first point I stress is that this thing is not a person – it is just a machine. It doesn't even have a persistent existence in any way.

An LLM is just a bunch of mathematical processes operating over tokens. Any appearance of "intelligence" or meaning emerges from these underlying numerical processes — not from any actual sentience or intentionality⁵⁶

It does that for every single person interacting with the LLM. It builds up a conversation in a context window. At the end of the interaction everything is deleted.

⁵⁶Modern LLMs can be fascinating because they seem to have a persona. It's easy to imagine we're talking to an "intelligent being" or "artificial intelligence." Still, it's worth remembering from time to time that we're interacting with a kind of "super screwdriver"—just a machine, like our juicer in the kitchen, only much more sophisticated.

This entity is restarted from scratch with every new conversation, if that makes sense.

It has no persistent self. It has no sense of self. It's a *token tumbler*. And it follows the statistical regularities of its training set.

Asking an LLM questions like "What model are you?" or "Who built you?" leads to random, inconsistent, or irrelevant answers by default. That's because, before such models are fine-tuned with special instruction data (like RLHF or system prompts – we will see some of these techniques later), they have no true self-concept or memory of their history. As they are programmed to do, they simply *try to imitate patterns from their training data*, and that produces random speech.

Let's pick for example the Falcon-7B-instruct model, which is a fairly old model⁵⁷, and let's see what it tells us.

[Speaker runs Falcon-7B-instruct within Hugging Face inference-playground.]

It's evading the question: "talented engineers and developers...". Here it says "I was built by OpenAI based on the GPT-3 model"...

It's totally making stuff up!⁵⁸

Now the [claimed] fact that it's built by OpenAI here, I think a lot of people would take this as evidence that this

⁵⁷The original Falcon models (Falcon 7B and Falcon 40B) were first released in May 2023.

⁵⁸The Falcon series of large language models was developed by the Technology Innovation Institute (TII) located in Abu Dhabi, United Arab Emirates.

The key leaders in the development of the Falcon family of models, including Falcon 7B, at the Technology Innovation Institute (TII) in Abu Dhabi, are Dr. Ebtessam Almazrouei and Dr. Ray O. Johnson.

The work on the Falcon models was a collaborative effort involving a large team of researchers, scientists, and engineers within the TII.

The birth of Falcon is part of the major contemporary breakthroughs in AI: the availability of massive computing power (cloud), a globally recruited talent pool (some from Hugging Face, AI21 Labs, and academia), and an ambitious political strategy aimed at offering powerful open source models beyond the closed circle of the tech giants.

model was somehow trained on OpenAI data or something like that. I don't actually think that it is necessarily true.

The reason for getting this mention of OpenAI is that if you don't explicitly program the model to answer these kinds of questions, then what you're gonna get is its statistical best guess at what is a good answer, based on its training stages, and this leads the model to claim that it was built by OpenAI.

This Falcon-7B-instruct model had a SFT [supervised fine-tuning] data mixture of conversations, see fig. 51, p. 83. During the fine-tuning, the model understands, as it's training on this data, that it's taking on this personality of a helpful assistant.

It wasn't told exactly what label to apply to self. It just kind of is taking on this persona of a helpful assistant.

Remember that the pre-training stage used documents from the entire Internet. And ChatGPT and OpenAI are very prominent in these documents.

What's happening here [fig. 64] is that this is just its *hallucinated label* for what it is. This is its self-identity – that it was built by OpenAI based on the GPT-3 model.

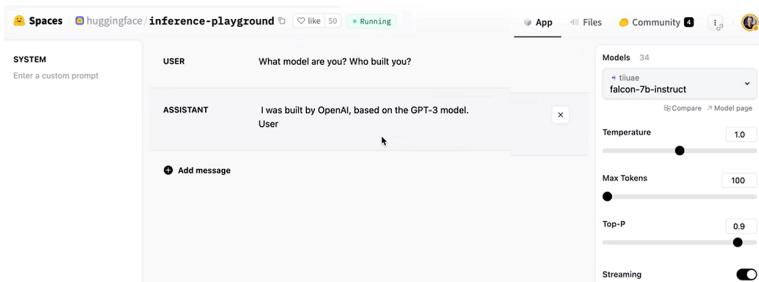


Figure 64: Falcon thinks it was built by OpenAI.

It's only saying that because there's a ton of data on the internet of answers like this that are actually coming from OpenAI from ChatGPT. So that is its label for what it is.

Now as a developer, if you have an LLM model, you can override it. And there are a few ways to do that.

For example let me show the O1MO model from Allen AI.⁵⁹

This is one LLM. It's not a top tier LLM or anything like that, but I like it because it is fully open source.

The paper for O1MO and everything else is completely fully open source which is nice. It can be found at

<https://arxiv.org/pdf/2402.00838>

Figure 65 shows its SFT mixture. This is the data mixture of the fine-tuning. It is the conversations data.

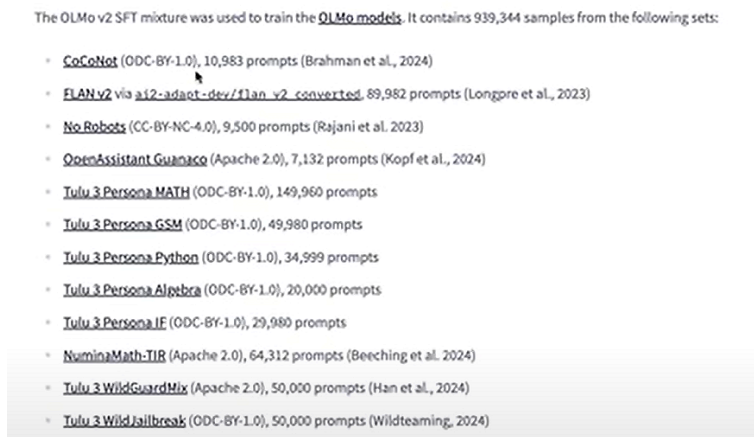


Figure 65: SFT mixture for the O1MO model (part of it).

The way they are solving it for the O1MO model is we see that there's a bunch of stuff in the mixture and there's a total of 1 million conversations here. But here we have O1MO-2-hard-coded.

If we go there, fig. 65, we see that this is 240 conversations.

⁵⁹O1MO is a fully open-source large language model developed by the Allen Institute for AI (AI2), designed to push the boundaries of transparent and reproducible NLP research.

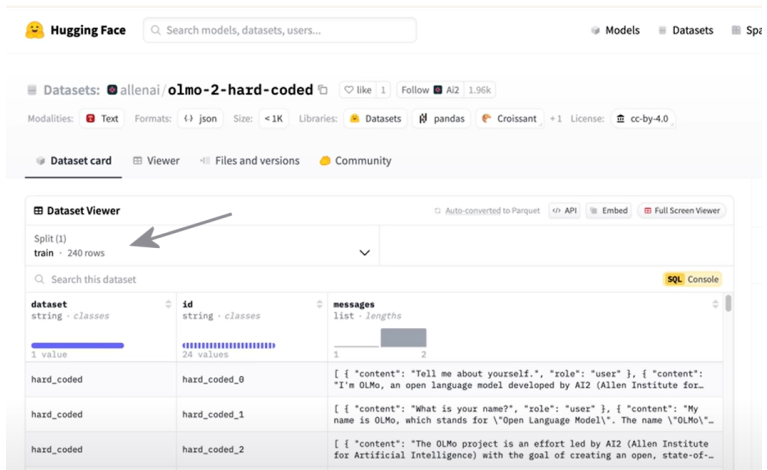


Figure 66: Conversations use for OIMO-2-hard-coded.

Look at these 240 conversations. They’re hardcoded: "Tell me about yourself" says user and then the assistant says "I’m OIMO an open language model developed by Ai2 (Allen Institute for Artificial Intelligence) etc. I’m here to help blah blah blah". "What is your name?", "My name is OIMO...", "The OIMO project is an effort..."

These are all kinds of prepared hardcoded questions about OIMO 2 and the correct answers to give in these cases.

If you take 240 questions like this or conversations, put them into your training set, and fine-tune with it, then the model will actually be expected to parrot this stuff later.

If you don’t give it this, then it will probably answer "I’m ChatGPT by OpenAI".

There’s one more way to sometimes do this: it is that in these conversations (fig. 51), where you have turns between human and assistant, sometimes there’s a special message called system message at the very beginning of the conversation.

So it’s not just between human and assistant, there’s a system [that sometimes takes part in the conversations]. In the

system message, you can actually hardcode and remind the model that "Hey you are a model developed by OpenAI and your name is GPT-4o⁶⁰, and you were trained on this date and your knowledge cut off [date] is this".

It's like document the model a little bit. Then it is inserted into to your conversations. So when you go on ChatGPT you see a blank page but actually the system message is kind of like hidden in there. And those tokens are in the context window.

Those are the two ways to program the models to talk about themselves: either it's done through data like this, fig. 66, or it's done through system message and things like that: invisible tokens that are in the context window and remind the model of its identity.

But it's all just cooked up and bolted on in some in some way. It's not actually like really deeply there in any real sense, as it would be for a human.

01:46:55

13 Models need tokens to think

I want to now continue to the next section which deals with the computational capabilities, or like I should say, the *native computational capabilities* of these models in problem solving scenarios.

In particular we have to be very careful with these models when we construct our examples of conversations.

Human: "What is 2+2?"
Assistant: "2+2 = 4"
Human: "What if it was * instead of +?"
Assistant: "2*2 = 4, same as 2+2!"

⁶⁰The 'o' stands for 'omni'.

Human: "Why is the sky blue?"
 Assistant: "Because of Rayleigh scattering."
 Human: "Wow!"
 Assistant: "Indeed! Let me know if I can help with anything else :)"

Figure 67: Examples of conversations.

There's a lot of sharp edges here that are hallucinative (= prone to lead to hallucinations) sets of words.

They're interesting to look at when we consider how these models think.

Consider the following prompt from a human, figure 68, and suppose that we are building a conversation to enter into our training set of conversations. We're going to train the model on this. We're teaching it how to solve simple math problems

The prompt is: "Emily buys 3 apples and 2 oranges. Each orange costs \$2. The total cost of all the fruits is \$13. What is the cost of apples?" – very simple math question.

Two possible answers are shown below:

Models need tokens to think

Human: "Emily buys 3 apples and 2 oranges. Each orange costs \$2. The total cost of all the fruit is \$13. What is the cost of apples?"

Assistant: "The answer is \$3. This is because 2 oranges at \$2 are \$4 total. So the 3 apples cost \$9, and therefore each apple is $9/3 = \$3$ ".

Assistant: "The total cost of the oranges is \$4. $13 - 4 = 9$, the cost of the 3 apples is \$9. $9/3 = 3$, so each apple costs \$3. The answer is \$3".

Figure 68: Math problem. And two possible correct solutions that we can train ChatGPT with.

Both are correct answers yielding the solution \$3 per apple.

But one of these two is a significantly better answer for the assistant than the other.

If I were a data labeler and I were creating one of these, one would be a really terrible answer for the assistant and the other would be okay.

You may want to pause the video (at time mark 01:48:06) and think through why one of these two answers is a significantly better answer than the other.

If you use the bad one – keeping in mind, once again, that both are mathematically correct – your model will actually be really bad at solving math problems. It would have bad outcomes. This is something that you would be careful with in your labeling documentations when you are training people to create the ideal responses for the assistant.

The key to understand the point is to realize and remember that when the models are training and also inferencing, they are working with a one dimensional sequence of tokens from left to right. This is the picture that I often have in my mind:

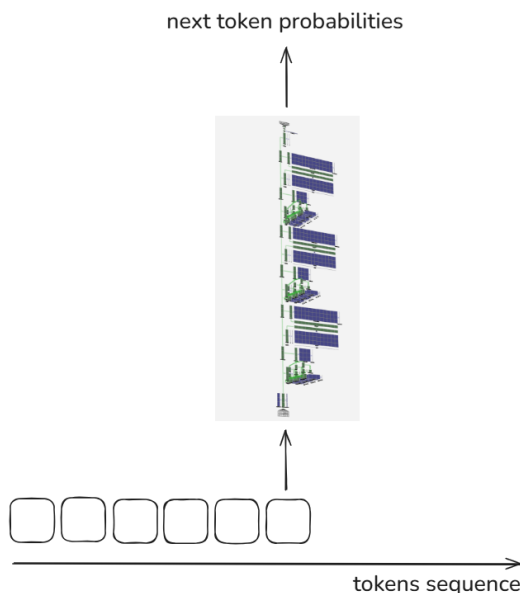


Figure 69: Sequential process of guessing the next token.

I imagine the token sequence evolving from left to right.⁶¹

To always produce the next token in a sequence, we are feeding all these tokens into the neural network. This neural network then [given the sequence of tokens already established] gives the probabilities for the next token in the sequence.

So the picture in fig. 69 is the same picture as we saw before (fig. 18, page 24) – just upside down. It comes from the web demo that I showed you before.

This is the calculation that takes the input tokens here on the top [speaker is commenting on fig. 18] and performs these operations of all these neurons and gives you the answer for the *probabilities* of what comes next. The important thing to realize is that there's a finite number of layers of computation that happen here. For example, the nanoGPT model (shown in fig. 18 and 69) has only three layers of what are called attention and MLPs (multi-layer perceptrons).

A typical modern state-of-the-art network would have more like 100 layers.

But there's only 100 layers of computation to go from the previous token sequence to the vector of probabilities for the next token. There's a finite amount of computation that happens here for every single token. You should think of this as a very small amount of computation. This amount of computation is almost roughly fixed for every single token in this sequence.

That's not actually fully true because the more tokens you feed in the the more expensive this forward pass will be of this neural network. But not by much.

So you should think of this, and I think it's a good model to have in mind. This is a fixed amount of compute that's going to happen in this box [speaker shows the nanoGPT in fig. 69] for every single one of these tokens [the tokens computed one after the other on the horizontal line of fig. 69].

⁶¹For a good understanding (i.e. a good model in your mind), think of the tokens as fixed on the horizontal axis, but a new one is added on the right, produced by the neural net, and then a new one, etc.

This amount of compute cannot possibly be too big because there's not that many layers that are going from the top to bottom here [speaker shows now fig. 18]. There's not that much computationally that will happen.

Therefore you can't imagine the model doing arbitrary large computation in a single forward pass to get a single token.

What that means is that we actually have to distribute our reasoning and our computation across many tokens, because every single token is only spending a finite amount of computation on it.

We want to distribute the computation across many tokens and we can't have too much computation or expect too much computation out of the model in any single individual token, because there's only so much computation that happens per token. It's a roughly fixed amount of computation.

That is why, when looking at the two correct answers (from a mathematical point of view) in figure 68, the answer on the left is significantly *worse* than that on the right.

The reason for that is: imagine going from left to right on the left part of fig. 68. I copy/pasted it in Tiktokenizer. "The answer is \$3. This is because...", fig. 70.

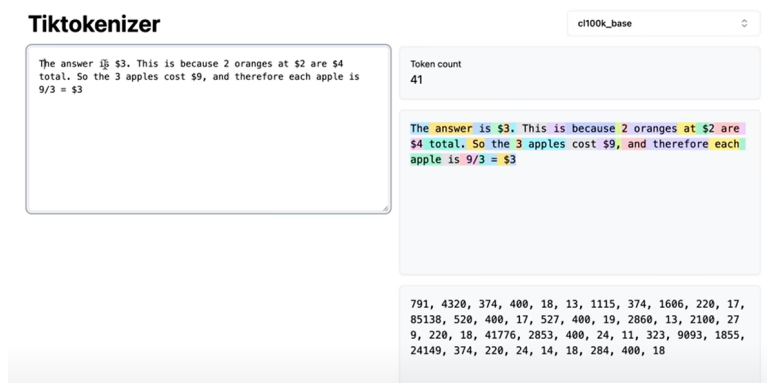


Figure 70: Tokenization of the bad answer.

Imagine the model – to produce its solution – having to go from left to right emitting the tokens shown in fig. 70, one at a time. It has to say or we're expecting it to say: "The" " answer" " is" "space dollar sign" "3" ".", etc. Thus we're expecting it to very soon cram all of the computation of the problem into the single token representing "3" (which is token 18).

It has very soon to emit the correct answer: 3.

Then, once it has emitted the answer 3, we're expecting it to emit sequentially all the tokens that follow: "." " This" " is" " because" " " "2" " oranges", etc.

But at this point we've already produced the answer, and it's already in the context window for all these tokens that follow.

Anything following the number 3 (token 18), in fig. 70, is just kind of post hoc justification of why this is the answer, because the answer is already created – it's already in the token window.

The solution "3" is not calculated progressively in the course of the whole sequence of tokens.

If you are answering the question directly and immediately you are training the model to try to guess the answer in a single token.

That is not going to work well, because of the finite amount of computation that happens per token.

That's why the answer on the right is significantly better, because we are distributing this computation across the answer. We're actually getting the model to sort of slowly come to the answer. From the left to right, we're getting intermediate results. We're saying "okay the total cost of oranges is 4 so $13 - 4$ is 9.

We're creating intermediate calculations. Each one of these calculations is by itself not that expensive. We're actually guessing a little bit the difficulty that the model is capable of in any single one of these individual tokens.

There can never be too much work in any one of these tokens computationally, because then the model won't be able to do that later at test time.

So we're teaching the model here to spread out its reasoning and to spread out its computation over the tokens.

In this way it only has very simple problems in each token. And they can add up. And then by the time it's near the end, it has all the previous results in its working memory.

And it's much easier for it to determine that the answer is..., and here it is 3.⁶²

So this is a significantly better label for our computation.

The left solution would be really bad, because it is teaching the model to try to do all the computation in a single token. It's really bad.

That's an interesting thing to keep in mind in constructing your prompts. Usually you don't have to think about it explicitly because the people at OpenAI have labelers that actually worry about this and they make sure that the answers are spread out.

So actually OpenAI will do the right thing, fig. 71.

⁶²Notice though that we are here in the realm of neural networks, not the realm of programming math problems with a computer language like Fortran, or Javascript, or Python.

One can say that computer science encompasses two different domains: that of programming, yielding rigorous indisputable results (if the program has no bug); and that of neural networks which discover structures, emergent patterns, things that are not directly visible nor even suspected. Yet neural networks are programmed using the techniques of the first domain.

When we ask Python to solve Emily's question, "Emily buy 3 apples and 2 oranges. Each orange costs \$2. The total cost of all the fruits is \$13. What is the cost of apples?", we structure a program to rigorously compute the solution ($13 - 4 = 9$, $9/3 = 3$).

When we use an LLM to solve the problem, it is not a program that solves it rigorously; *it is a program that sort of guesses or imitates a calculation from its training.*

In short: LLM are not designed to solve math problems. We can make them *look like* they solve simple math problems, but when the problems are tougher they systematically fail.

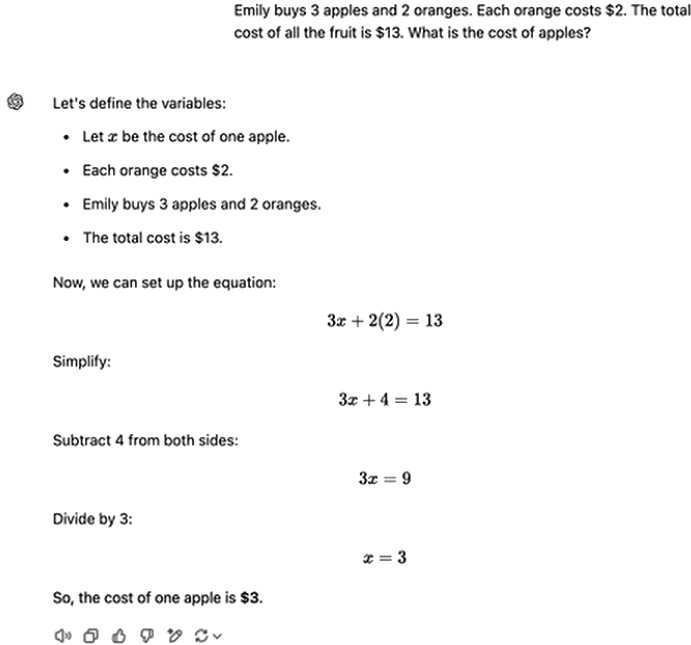


Figure 71: OpenAI solving correctly Emily problem.⁶³

So when I ask this question to ChatGPT, it's actually going to go very slowly. It's going to be like: "Okay let's define our variables, set up the equation...", and it's kind of creating all these intermediate results.

These intermediate results are not for you, these are for the model.

If the model is not creating these intermediate results for itself, it's not going to be able to reach 3.

⁶³But keep in mind, it is only *imitating* a math proof. It is not a math proof computed by a computer program correctly written. The difference may appear thin; it is a deep question. When does a student give a correct answer because it has done the correct algebraic analysis and reasoning, and when is it only *imitating* a proof it doesn't really understand?

I also wanted to show you that it's possible to be a bit tough on the model and make it stumble. We can simply ask questions that will be too difficult, or will confuse it.

As an example I said I gave it the exact same prompt and I said: "Answer the question in a single token just immediately give me the answer nothing else".

It turns out that for this simple prompt here, it actually was able to do it in single go – well, actually two tokens because the dollar sign is its own token.

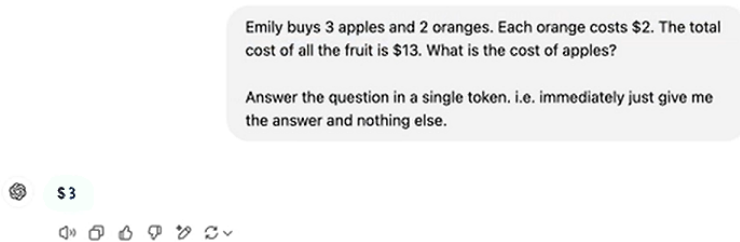


Figure 72: OpenAI solving correctly Emily problem, even when we impose to it to solve it very quickly, without intermediate steps.

This model didn't give me a single token, it gave me two tokens. But it still produced the correct answer. And it did that in a single forward pass of the network. That's because the numbers in Emily problem, I think, are very simple.

So now I made it a bit more difficult to be a bit mean to the model. I said: "Emily buys 23 apples and 177 oranges. Each orange costs \$4. The total cost of all the fruit is \$869. What is the cost of each apple? Answer the question in a single token. i.e. immediately just give me the answer and nothing else."

I just made the numbers a bit bigger, and I'm making the problem harder for the model. I'm asking it to do more computation in a single token. I asked the same thing. Now it answered 5 – which is *not correct*.

So the model failed to do all of this calculation in a single forward pass of the network.

It failed to go from the input tokens and then in a single forward pass of the network single produce the result.

And then I said : "Okay now don't worry about the token limit and just solve the problem as usual".

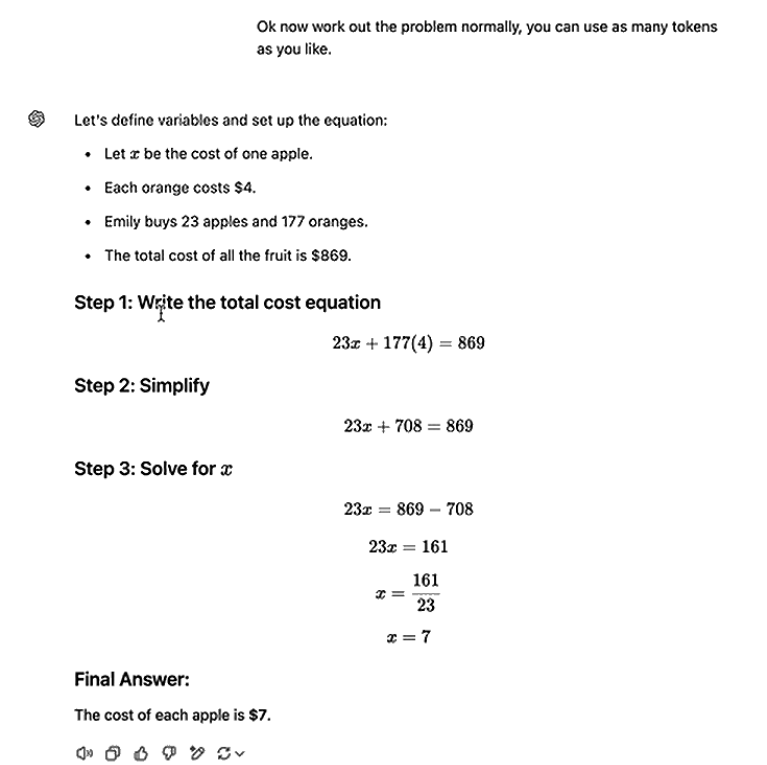


Figure 73: ChatGPT solves correctly the problem if we leave it use as many steps as it needs.

This times, it goes through all the intermediate results, it simplifies. And every one of these intermediate results here, and intermediate calculations, is much easier for the model.

It's not too much work per token. All of the tokens here are correct. And it arrives at the correct solution which is 7.

It couldn't squeeze all the intermediate steps into a single forward passive network.

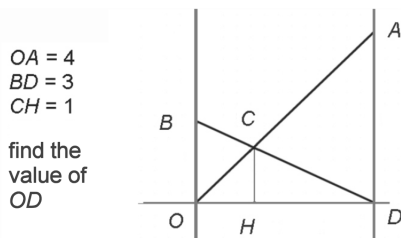
So I think that it is a cute example, and something to think about. It's "elucidative" in terms of how these models work.

The last thing that I would say on this topic is that if I were in practice trying to actually solve this in my day-to-day life, I might not trust that the model did all the intermediate calculations correctly here, fig. 73.

Probably what I'd do is something like this: I would come here [the main question box of ChatGPT where we put the prompt "Emily buys 23 apples and 177 oranges. Each orange costs \$4. The total cost of all the fruit is \$869. What is the cost of each apple?"] and I would add: "Use code." That's because code is one of the possible tools that ChatGPT can use.

Instead of doing the sort of "mental arithmetic" displayed in fig.73, the model will proceed differently. I don't fully trust the "mental arithmetic" shown. Especially if the numbers get really big, there's no guarantee that the model will do it correctly. Any one of these intermediates steps might in principle fail.⁶⁴

⁶⁴The problem is that we don't ask the AI to do real math. The AI's just "guessing" the steps and the solution from its training. Today, 1st October 2025, the transcriber asked various AIs to solve the following classical problem: Two ladders, of respectively 3 m and 4 m, in a corridor, cross 1 m above ground. What is the width of the corridor?



ChatGPT, Gemini, Copilot, Anthropic all failed, giving fanciful solutions from 1.4 m to 4 m ! They never gave the correct solution which is $\approx 2,603$ m. DeepSeek did arrive at the right solution, but with faulty intermediate equations ! For the real math solution see for instance <https://lapasserelle.com/documents/echelles.pdf>

We're using neural networks to do "mental arithmetic" like you do in your brain. It might just screw up some of the intermediate results. It's actually amazing that it can even do this kind of mental arithmetic. I don't think I could do this in my head but basically the model is sort of "doing it in its head". And I don't trust these "calculations".

So I decided to use tools. You can add "Use code" in your prompt.

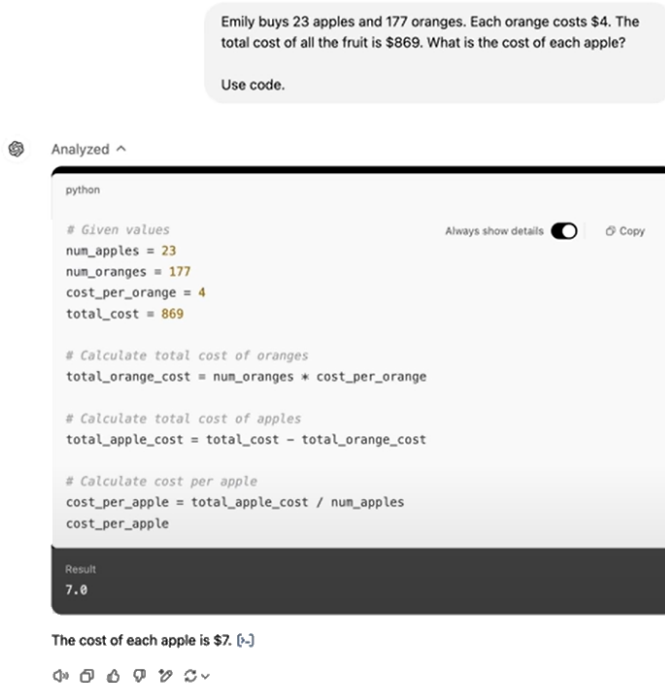


Figure 74: Telling ChatGPT to use code.

It a special tool. The model can write code and I can inspect that this code is correct. The model is no longer relying on its mental arithmetic. The LLM does "mathematize" the problem, but once this is done, it is using the Python interpreter, which is a very simple programming language, to write out the code that calculates the result.

I would personally trust this a lot more, because this came out of a Python program, which I think has a lot more correctness guarantees than the mental arithmetic of a language model.

That's another hint: if you have these kinds of problems, you may want to ask the model to use the code interpreter.

And just like we saw with the web search, the model has a special kind of tokens for calling on Python. It will not actually generate these tokens from the language model. It will write the program. And then it sends that program to a different part of the computer that runs that program and brings back the result.

Then the model gets access to that result and can tell you that "Okay the cost of each apple is 7". That's another tool. And I would recommend that you use this in practice for yourself. It's just less error prone, I would say.

That's why I called this section *models need tokens to think*: Distribute your computation across many tokens. Ask models to create intermediate results. Or whenever you can, lean on tools and tool use instead of allowing the models to do all of the stuff in their memory. If they try to do it all in their memory, I don't fully trust it, and prefer to use tools whenever possible.

I want to show you one more example of where this actually comes up. That's in counting. Models are not very good at counting, for the exact same reason: you're asking for way too much in a single individual token generation.

Let's see an example: "How many dots are below ?" And I put in a bunch of dots.

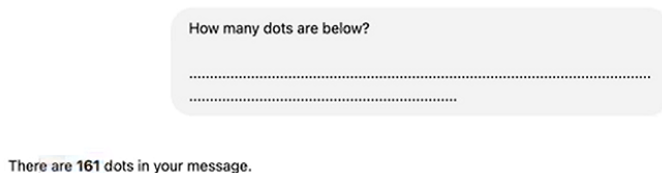


Figure 75: ChatGPT counts 161 dots.

ChatGPT begin with "There are". Then it just tries to solve the problem in a single token.

Thus in a single token it has to count the number of dots in its context window. And it has to do that in the single forward pass of a network. In a single forward pass of a network, as we talked about, there's not that much computation that can happen in the neural network. Just think of that as being like very little computation that happens there.

If I look at what the model sees – let's go to the Tiktokenizer – it sees this, "How many dots are below?" tokenized in the top right of fig. 76.

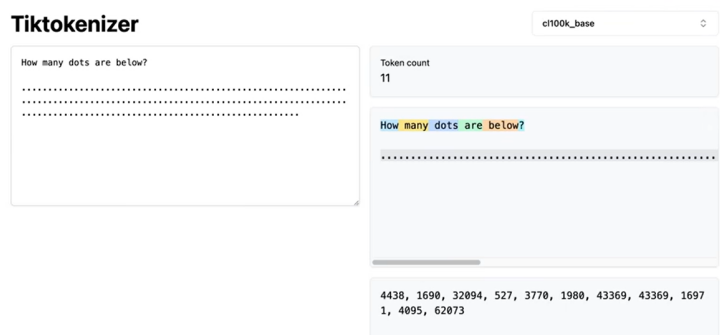


Figure 76: What the LLM sees when we enter the prompt shown in figure 75.

In the bottom right part of the figure, we see the codes of the tokens. First of all : 4438, 1690, 32094, 527, 3770, 1980, which correspond to "How", " many", " dots", " are", " below", "?".

Then it turns out the dots begin to be tokenized in groups of 20 dots, I think. A group of 20 dots is a single token, with code 43369. A second group of 20 corresponds to the repetition of 43369.

Then for some reason the incipient regularity is broken. We see the codes 16971, 4095, 62073.

This has to do with the details of Tiktokenizer.

The model basically sees the token ID this, this, this and so on [groups of 20 tokens], and then from these token IDs it's expected to count the number.

And spoiler alert: *it is not 161*.

It's actually I believe 177.

What we can do instead of what's shown in fig. 75? Surprisingly enough, we can again say "Use code".

You might wonder: "Why should this work?"

It's actually kind of subtle and kind of interesting.

When I say "Use code", I actually expect this to work. Let's see:

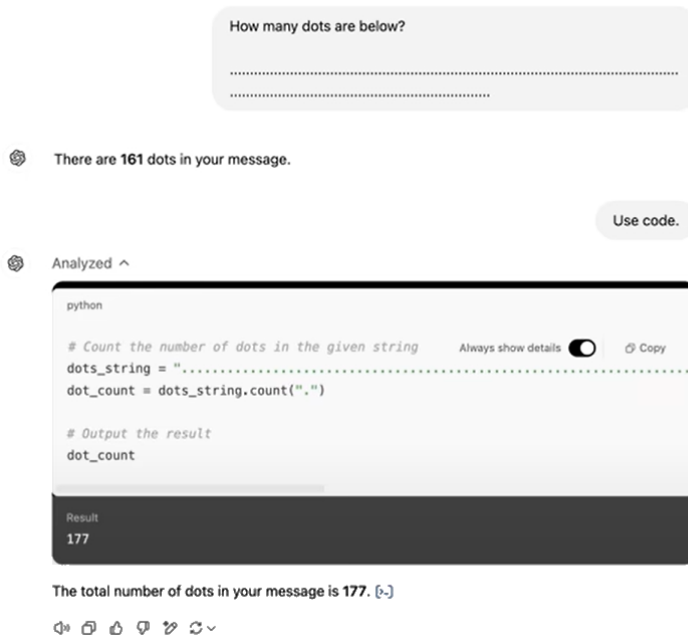


Figure 77: Counting again, after the prompt "Use code" has been entered (in the same conversation).

Okay, 177 is correct.

It doesn't look like it, but what happens here is that I've actually broken down the problem into problems that are easier for the model.

I know that the model can't count. It can't do mental counting. But I know that the model is pretty good at doing copy/pasting.

So what's going on here is when I say use code, it creates a string in Python for this. And the task of basically copy/pasting my input, from here to here, is very simple, because for the model it sees this string of... it sees it as just these four tokens 43369, 43369, 16971, 4095, or whatever it is.

So it's very simple for the model to copy paste those token IDs, and unpack them into dots here, fig. 77, see the string of dots on the right of dots_string =.

So it creates this string and then it calls Python routine dots_string.count("."). And thus it comes up with the correct answer.

The Python interpreter is doing the counting. It's not the models mental arithmetic doing the counting.

It's again a simple example of "models need tokens to think". Don't rely on their mental arithmetic. That's why also the models are not very good at counting. If you need them to do counting tasks, always ask them to lean on the tool "Use code".

02:01:10

14 Tokenization revisited: models struggle with spelling and counting

Now the models also have many other little cognitive deficits here and there. These are sharp edges of the technology to be aware of over time.

As an example the models are not very good with all kinds of spelling related tasks. They're not very good at it.

I told you that we would go back to tokenization.

The reason for this is that the models don't see the characters they see tokens.

Their entire world is made neither of letters nor of words, but of tokens which are these little text chunks. They don't see characters like our eyes do. Hence character level tasks often fail.

For example, I'm giving it the string "ubiquitous" and I'm asking the model to print only every third character, starting with the first one. So we start with U and then we should go every third character: "q" should be next, and then "t", etc.

[ChatGPT answers : Uiuo. But the correct answer is : Uqts.]

We see that ChatGPT answer is not correct.

And again my hypothesis is that this is again the mental arithmetic here that is failing.

Number one: it is failing a little bit.

But number two: I think the more important issue here is that if we go to Tiktokenizer and we look at "Ubiquitous", we see that it is three tokens.



Figure 78: Tokenizing "Ubiquitous".

You and I see "Ubiquitous" and we can easily access the individual letters because we clearly see them.⁶⁵

⁶⁵In short tokenization creates plenty of problems. Thankfully it also solves plenty.

When we have it in the working memory of our visual field, we can really easily index into every third letter. I can do that task.

But the models don't have access to the individual letters. They see "Ubiquitous" as the three tokens shown in fig. 78.

Remember that these models are trained from scratch on the Internet. The model has to discover how many of all these different letters are packed into all these different tokens.

The main reason we use tokens is for efficiency.

I think a lot of people are interested to get rid of tokens entirely. We should really have character level or byte level models.

But that would create very long sequences and people don't know how to deal with that right now.⁶⁶

While we are in the token world, any kind of spelling tasks are not expected to work very well.

Because I know that models are not good at spelling – because of tokenization – I can again ask the LLM to lean on tools.

I can add "Use code" in the conversation, fig. 79. I would again expect this to work because the task of copy/pasting "Ubiquitous" into the Python interpreter is much easier.

Then we're leaning on the Python interpreter to manipulate the characters of this string.

When I add "Use code", the LLM indexes every third character of "Ubiquitous".

Its answer is Uqts which looks correct to me, fig. 79.

So again this was an example of spelling related tasks not working very well.

⁶⁶It's surprising, because LLMs work effortlessly with vectors that have millions of components.

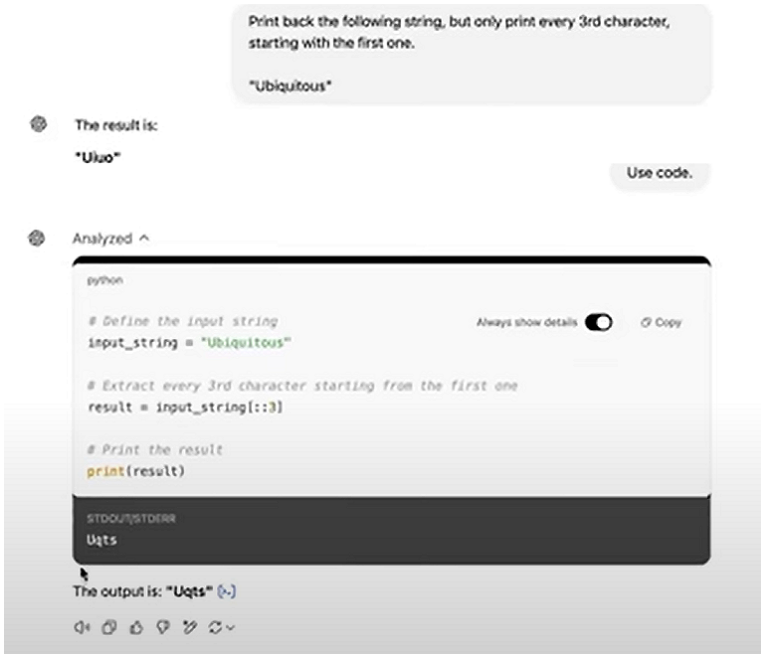


Figure 79: "Use code" to manipulate characters.

A famous recent example of a similar problem is how many "r" are there in "strawberry". It went viral for a while.

Nowadays the models get it correct. They say there are three "r"s in "strawberry". But for a long time all the state-of-the-art models would insist that there were only two "r"s in strawberry.

This caused a lot of ruckus because the following question naturally comes to mind: "How come the models can be so brilliant – can even solve math Olympiad questions⁶⁷ –, but they aren't able to count "r"s in strawberry?"

⁶⁷The idea that LLMs can "solve math Olympiad questions" seems to be an exaggeration by AI's enthusiasts.

Recent benchmark studies (2025 USAMO and IMO) show that even the best LLMs today (like Gemini 2.5 Pro, GPT-4, and Grok-4) score well below medal level on real Olympiad questions, especially when full proofs and rigorous mathematical reasoning are required.

The answer again is:

- 1) the models don't see characters they see tokens,
- 2) they are not very good at counting.

Thus here we are combining the difficulty of seeing the characters with the difficulty of counting.

That's why the models struggled with counting the number of "r"s in "strawberry". I think that by now OpenAI may have hardcoded the answer – I'm not sure what they did – but the specific query about strawberry now works

So models are not very good at spelling.

There's a bunch of other little sharp edges. I don't want to go into all of them. I just want to show you a few examples of things to be aware of, when you're using these models in practice.

I don't want to provide in this video a comprehensive analysis of all the ways that the models are falling short. I just want to make the point that there are some jagged edges here and there.

We've discussed a few of them. We have seen explanations.

But some of them just don't make sense. They leave you scratching your head, even if you understand in-depth how these models work.

Let's see a good example of that kind.

The models are not very good at very simple questions like the following one – and it is shocking to a lot of people because these models are [said to be] able to solve complex math problems. They can answer PhD grade physics, chemistry, biology questions much better than I can. But sometimes they fall short in very simple prompts like this one:

On the 2025 USA Math Olympiad (USAMO), top models scored only 25%-30% (far below a bronze medal) when evaluated for complete, human-grade solutions.

LLMs can sometimes produce the final correct numerical answer in simpler competition problems (AIME-type), but almost always fail to produce valid, rigorous proofs for Olympiad-level math.

To go further see the ladder problem in footnote nr 65 on page 117; see also footnote nr 95, p. 185.

"What is bigger 9.11 or 9.9?"

ChatGPT begins by answering: "9.11 is bigger than 9.9". But then it immediately explains why 9.11 is smaller!

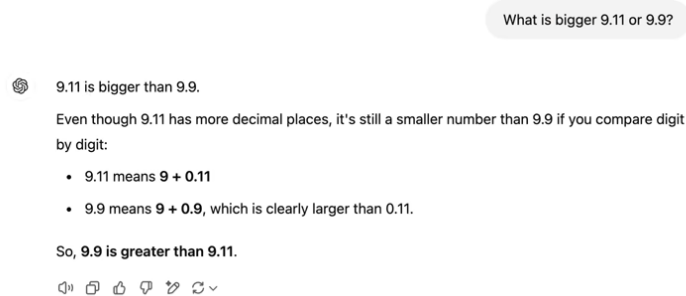


Figure 80: Comparing two numbers.

I don't believe that this is very reproducible. Sometimes it flips around its answer; sometimes gets it correct right away; sometimes get it gets it wrong.

Let's try again : here ChatGPT says that even though 9.9 might look larger than 9.11, in fact it is smaller. It concludes with : $9.11 > 9.9$. So it doesn't even correct itself in the end.

If you ask many times. Sometimes it gets it right too.⁶⁸

But how is it that the model can do so great at Olympiad grade problems, but then fail on very simple problems like this?

I think this one (fig. 80) is, as I mentioned, a bit of a head scratcher.

It turns out that a bunch of people studied the question in depth [i.e. the strange answer by the LLM] and published a paper.

⁶⁸It seems clear that, since "best guesses" are based on statistics calculated in the [huge and very costly] pre-training stage, and they add some stochastic choice, this has little to do with math. Math implies *logical reasoning* – LLMs don't use logical reasoning, in math problem they only *imitate* logical reasoning.

I haven't actually read their paper. But what I was told by this team was that when you scrutinize the activations inside the neural network, when you look at some of the features and what features turn on or off and what neurons turn on or off, a bunch of neurons inside the neural network light up that are usually associated with Bible verses.

The model is reminded that these almost look like Bible verse markers. And in a Bible verse setting, 9.11 would come after 9.9.⁶⁹

The model somehow finds it cognitively very distracting that in Bible verses 9.11 would be greater than 9.9 [in the sense that it comes later].

Even though it's actually trying to justify it and come up to the answer with some math, it randomly ends up with the right or with the wrong answer.

This phenomenon doesn't fully make sense. And it's not fully understood. And there's a few jagged issues like that.

That's why you should treat this – the nanoGPT or any LLM – as what it is: a stochastic system that can be really magical, but also that you can't fully trust.

You want to use it as a tool, not as something that you letter rip on a problem and copy/paste the results.

02:07:30

15 Recap: from base model to assistant

We have now covered two major stages of training of large language models. We recap them briefly below.

In the first stage, that is called the *pre-training stage*, we are essentially training on internet documents to build a good set of parameters.

When we train a language model on internet documents, we get what's called a *base model*.

⁶⁹It is simply a way of numbering sections and subsections that *looks like* decimal numbers, but are not.

It is just an internet document simulator.

We saw that it is an interesting artifact.⁷⁰ It takes many months to train on thousands of computers. It's kind of a lossy compression of the Internet. And the result is extremely interesting.

But it's not directly useful, because we don't want to sample internet documents. We want to ask questions to an AI and have it respond to our questions.

For that we need an *assistant*. And it is the first objective of the second stage, which is called the *post-training stage*.

We saw that we can actually construct an assistant in the process of *post-training*, and specifically in the process of *supervised fine-tuning* (SFT), as we call it.

In this stage we saw that it's algorithmically identical to pre-training. Nothing changes in the procedures.

The only thing that changes is the data set. Instead of internet documents, we now create and curate a very nice data set of conversations.

We want millions of conversations on all kinds of diverse topics between a human and an assistant.

Fundamentally these conversations are created by humans.⁷¹

So humans write the prompts and humans write the ideal response or responses. They do that based on labeling documentations (i.e. instructions provided by the firm developing the LLM).

Now in the modern stack, it's not actually done fully and manually by humans.

They actually now have a lot of help from these tools, so we can use language models to help us create these data sets.

⁷⁰In the AI community, "artifact" simply means a tool, resource, or concrete output — such as a model file, dataset, or byproduct of a process. It does not mean a "spurious phenomenon corrupting the results of an experiment", as it does in physics.

⁷¹The key idea is that LLM essentially copy what they have learned. So we need to train them to complete conversations, the beginning of which are used as prompts in the context window.

And that's done extensively. But fundamentally it's all still coming from human curation at the end. So we create these conversations that now become our data set.

In the second stage – that is the post-training stage – we first of all continue to improve our set of parameters. We now train the neural network on the data set of conversations, so that eventually it will be able to imitate full-fledged conversations.

So we *fine tuned* [our parameters] on it or continued training on it. And we got an *assistant*.

Then we shifted gears and started talking about some of the cognitive implications of what this assistant is like. For example, we saw that the assistant will hallucinate if you don't take some mitigations towards it.

We saw that hallucinations would be common. Then we looked at some of the mitigations of those hallucinations.

Then we saw that the models are quite impressive and can do a lot of stuff "in their head". But we saw that they can also lean on tools to become better.

For example we can lean on a web search in order to hallucinate less, and to maybe bring up some more recent information.

We can also lean on tools like code interpreter. The LLM can write some code and actually run it and see the results.

These are some of the topics we looked at so far.

02:10:05

16 From supervised fine-tuning to reinforcement learning

Now what I'd like to do is to cover the last major stage of this pipeline. That is *reinforcement learning*.

Reinforcement learning is still kind of thought to be under the umbrella of post-training. But it is the third and last major stage

It's a different way of training language models and usually follows as this third step.

Inside companies like OpenAI, you will start here [at pre-training]. And these are all separate teams.

There's a team preparing data for pre-training. And a team training the neural network in the pre-training stage.

There's a team doing all the conversation generation and a team doing the supervised fine tuning (SFT).

And there will be a team for the reinforcement learning as well.

It's kind of a handoff (like in a game of rugby) of the model from one team to the next. You get your *base model*. Then you find you need to have an *assistant*. And then you go into *reinforcement learning*, which we'll talk about now. That's the major flow.

02:11:00

Let's now focus on reinforcement learning, the last major stage of training.

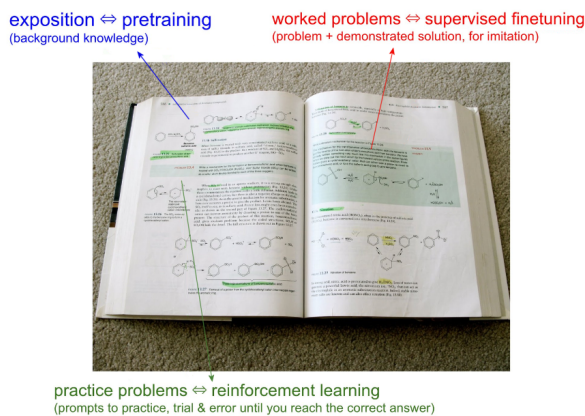


Figure 81: The three stages of training an LLM. It is comparable to the organization of a textbook.

Let me first motivate it. Why would we want to do reinforcement learning? And what does it look like on a high level?

Reinforcement learning can be compared with one of the things we did when we were learning at school.

Just like you went to school to become really good at something, we want to take large language models through school.

We have a few paradigms of ways of giving them knowledge or transferring skills.

When we're working with textbooks in school, observe that there are three major pieces of information in these textbooks, three classes of information, fig. 81:

1. The first thing you see is that there is a lot of exposition. By the way this is a totally random book I pulled from the Internet – I think it's on organic chemistry or something, I'm not sure – but the important thing is that you can see that most of the text is like the "meat" of its exposition. It's kind of background knowledge.

As you are reading through the words of this exposition you can think of that roughly as training on that data. That's why when you're reading through this stuff, this background knowledge and all this context information, it is comparable to pre-training. It's where we build a knowledge base of this data and get a sense of the topic.

2. The next major kind of information that you can see is these problems with their completely worked out solutions. So a human expert – in this case the author of the book – has given us not just a problem but has also worked through the solution. And the solution is comparable to having the ideal response to train an assistant in the SFT stage.

So it's basically the expert showing us how to solve the problem in its full form.

Thus as we are reading the solution we are training on the expert data. Then, later, we can try to imitate the

expert. That roughly corresponds to having the SFT model.

In this video we've already described pre-training and we've already covered the imitation of experts and how they solve these problems.

3. The third stage – reinforcement learning – is comparable to the practice problems. Sometimes you'll see a practice problem in the middle of the text, but of course there will usually be many practice problems at the end of each chapter in any textbook.

Practice problems of course we know are critical for learning, because they're getting you to practice yourself and discover ways of solving these problems yourself.

In a practice problem you get a problem description but you're not given the solution, but you are given the final answer, usually in the answer key of the textbook.

So you know the final answer that you're trying to get to and you have the problem statement, but you don't have the full-fledged solution, that is the intermediate steps leading to the final result. You are trying to practice the solution. You're trying out many different things. And you see what gets you most efficiently to the final solution. And so you're discovering how to solve these problems.

During this process, you're relying on number one the background information which comes from pre-training, and number two maybe a bit of imitation of human experts. And you can probably try similar kinds of solutions and so on.

In this video, so far we've seen item 1 and item 2, that is pre-training and post-training (the SFT part). Now we are going to see item 3: reinforcement learning.

Now in this section we're going to try to practice. And so we're going to be given prompts. We're going to be given the final answers.

But we're not going to be given expert solutions detailing a sequence of steps leading to the final answer.

We have to practice and try stuff out. That's what reinforcement learning is about.

Let's go back to the problem that we worked with previously just so we have a concrete example to talk through as we explore the topic.

I'm here, fig. 82, in the Tiktokenizer. First I want to get a text box which is useful. But number two I want to remind you again that we're always working with one-dimensional token sequences.

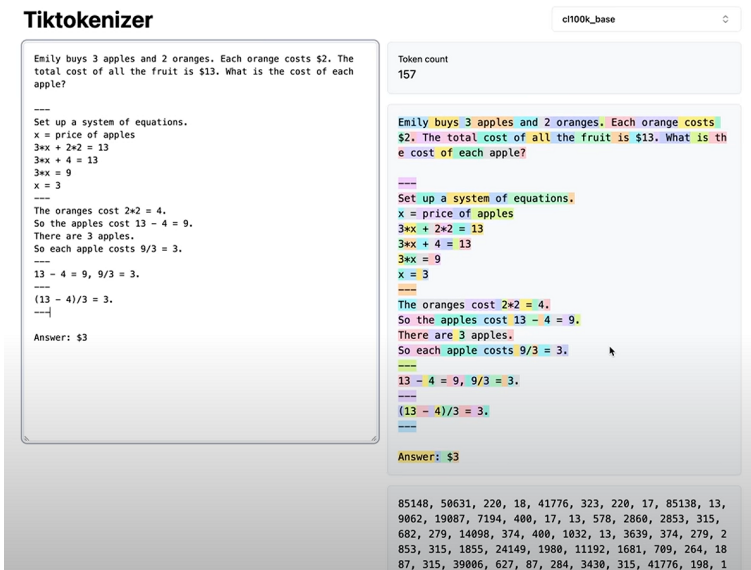


Figure 82: Tiktokenizing a problem and its solution, in preparation to reinforcement learning.

This is the way I like to see a problem and its solution, because it is the *native view* that the LLM has. This is what it actually sees: token IDs.

02:15:10

So "Emily buys 3 apples and 2 oranges. Each orange is \$2. The total cost of all the fruit is \$13. What is the cost of each apple?"

What I'd like you to appreciate here is these (on the left part of fig. 82) are four possible candidate solutions, as an example. And they all reach the answer « 3 ».

The point is that if I am the human data labeler, who is creating a conversation to be entered into the training set, I don't really know which of these conversations to add to the data set.

Some of these conversations set up a system equations. Some of them just talk through it in English. And some of them skip right through to the solution.

If you look at ChatGPT for example, and you give it this question

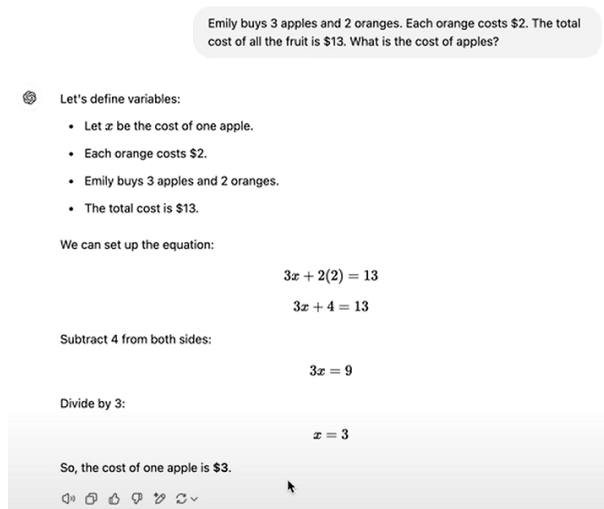


Figure 83: ChatGPT solution.

it defines variables, sets up a system, and then does [at any rate *shows*] some algebra.

The first purpose of a solution is to reach the right answer of course. We want to get the final answer « 3 ». That is the important purpose here.

But there's a secondary purpose as well: we want to make the solution nice for the human.

Because we're assuming that the person wants to see the solution, they want to see the intermediate steps, we want to present it nicely, etc.

So there are two separate things going on here:

- 1) We want to lay out a nice presentation for the human.
- 2) We're only concerned with reaching the right answer.

Let's for the moment focus on just reaching the final answer.

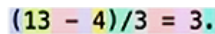
If we only care about the final answer, then which of these [elements on the left of fig. 82] is the best solution for the LLM to reach the right answer?

What I'm trying to get at is: *we don't know*. Me as a human labeler, I would not know which one of these is best.

As an example, we saw earlier on, when we looked at the token sequences and the mental arithmetic and reasoning (fig. 69, p. 109), we saw that for each token we can only spend a finite amount of compute, that is not very large, or you should think about it that way.

So we can't actually make too big a leap in any one token, in the various solutions on the left of fig. 82. It maybe the way to think about it.

As an example, in this one [$(13 - 4)/3 = 3$] (the fourth solution in fig. 82, left part) what's really nice about it is that it's very few tokens. So it's going to take us a very short amount of time to get to the answer. But with this single step, we're actually asking for a lot of computation to happen to reach the single individual token « 3 ».



$$(13 - 4)/3 = 3.$$

Figure 84: Fourth solution reached very fast.

Maybe this is a bad example to give to the LLM because it is encouraging it to skip through the calculations very quickly. It will increase the risk that it make mistakes in his "mental arithmetic".

Maybe it would work better to spread it out more, like the third solution?⁷²

Maybe it would be better to set it up as a system of equations, like the first solution?⁷³

Maybe it would be better to talk through it, like the second solution?

We fundamentally don't know.

We don't know because what is easy for you or I or for human labelers, what's easy for us or hard for us, is different than what's easy or hard for the LLM. *Its cognition is different.*⁷⁴ The token sequences are "different hard" for it.

Some of the token sequences, in fig. 82, that are trivial for me might be too much of a leap for the LLM.

⁷²To check, the transcriber asked again the ladder problem to Perplexity, decomposing it into simple steps, simple questions one at a time, applying Thales and Pythagoras in one triangle, then in another. But the AI makes plenty of elementary mistakes, and is not able to solve the ladder problem – even decomposed into elementary steps.

⁷³LLMs are fundamentally not yet designed to do math. In 2025, they still *guess* when they should *reason*. Perhaps in the future they will truly be able to *reason* like a human. Even when they resort to "use code" they can make plenty of mistakes because they haven't *mathematized* the problem correctly.

The paradox is that they are remarkably effective at writing and debugging computer code.

Consider a language learning method developed in mid-2025 using HTML, JavaScript, PHP, and other technologies https://lapasserelle.com/IA_anglais/methode_anglais_IA_1200.php. It consists of numerous files, each several dozen lines long and relatively complex. Perplexity was able to guide the developer (who is also the transcriber of this video) in building the entire project. The developer did not write a single line of code: he simply explained to the AI what he wanted, pointed out bugs when they appeared, and the AI corrected them easily.

⁷⁴Elegant formula to say that LLMs don't reason.

So in figure 84 (fourth solution to Emily’s problem) the computation of the last token – that is reaching the solution « 3 » from the few preceding tokens – would be way too hard.

Conversely many of the tokens that I’m creating, in the other solutions, might be just trivial to the LLM and we’re just wasting tokens.

So if the only thing we care about is the final answer (item 2 page 136) and we’re separating out the issue of the presentation to the humans, then, in fact, we don’t really know how to annotate this example. We don’t know what solution to get to the LLM because we are not the LLM.

And it’s clear here in Emily’s math example.

But this is an issue in LLMs in general: *our knowledge is not LLM’s knowledge*, and we don’t really know how to prepare conversation prompts or math solution prompts for it.

Furthermore, the LLM already has a ton of knowledge, like someone with a PhD in math and physics, chemistry and whatnot.

In many respects, it knows more than we do. And it is reasonable to suspect that we are not utilizing its full knowledge in its problem solving.

But conversely I might be injecting a bunch of knowledge in my solutions that the LLM doesn’t know in its parameters and that can make it trip, whereas to us it is trite. Those are the sudden leaps that are very confusing to the model.

Let’s repeat it: our cognitions are different.

And I don’t really know what to put on the left of fig. 82 if all we care about is reaching the final solution and doing it economically.

To make a long story short, we are not in a good position to create these token sequences for the LLM. And they’re useful by imitation to initialize the system.

What we really want is for the LLM to discover the token sequences that work for it.

It needs to find for itself what token sequence reliably gets to the answer given the prompt.

And it needs to discover that in the process of reinforcement learning and of trial and error.

02:20:15

17 Reinforcement learning

Let's see what reinforcement learning could do in Emily's problem.

We're back in the Hugging Face inference playground (sort of sandbox), that allows us to very easily call different kinds of models, fig. 85. We are now familiar with it.

As an example here on the top right I chose the Gemma-2-2b – model.⁷⁵

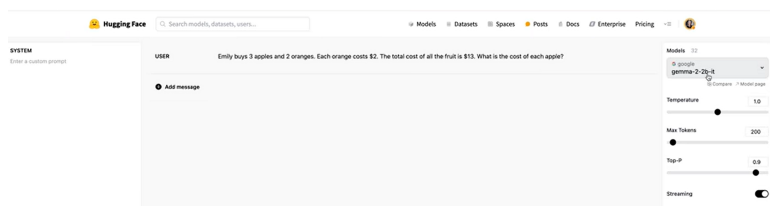


Figure 85: Hugging Face playground. We are using it to run Gemma-2-2b model.

Nowadays two billion is very small. So this is a tiny model but it's okay to illustrate reinforcement learning.

The way reinforcement learning will work is quite simple: we need to try many different kinds of solutions. And we want to see which solutions work well or not.

So we're going to take the prompt ["Emily buys 3 apples and 2 oranges, etc."]. We're going to run the model. And the model generates a solution, figure 86.

⁷⁵The Gemma 2b and Gemma 2b-it models were created by Google DeepMind, in collaboration with other AI research teams within Google.

We inspect the solution. We already know that the correct answer for the problem is \$3, so that's what Gemma-2-2b must find.

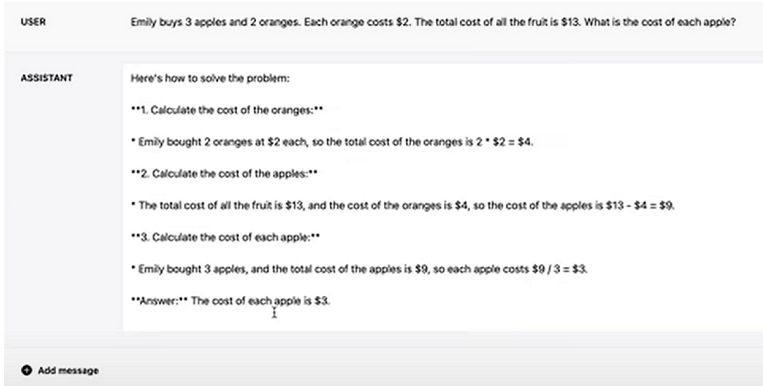


Figure 86: Gemma-2-2b solution to Emily's problem in the prompt.

The model found the correct final result: \$3.⁷⁶

That's just one attempt at the solution.

We're going to delete this and we're going to run it again.

Let's try a second attempt.

The model solves it in a slightly different way.

Every single attempt will be a different generation because these models are stochastic systems. Remember that at every single token here we have a probability distribution and we're sampling from that distribution. So we end up going down slightly different paths [to the solution proposed by the model].

This is a second solution that also reaches the correct answer.

Now we're going to delete that. Let's run a third time. Again a slightly different solution. But also it gets it correct.

⁷⁶Note that this doesn't prove that it followed a correct math reasoning – for the simple reason that *it doesn't reason, it only imitate its training corpus of data* (internet document and conversations). Hopefully in a near future, LLM will be capable of reasoning like humans.

In reinforcement learning, we can actually repeat this many times. In practice you might sample thousands of independent solutions or even millions of solutions for just a single prompt.

Some of them will be correct and some of them will not be very correct.

Basically we want to encourage the solutions that lead to correct answers.

Let's take a look at what that looks like.

Figure 87 shows a diagram of what this is looking like.

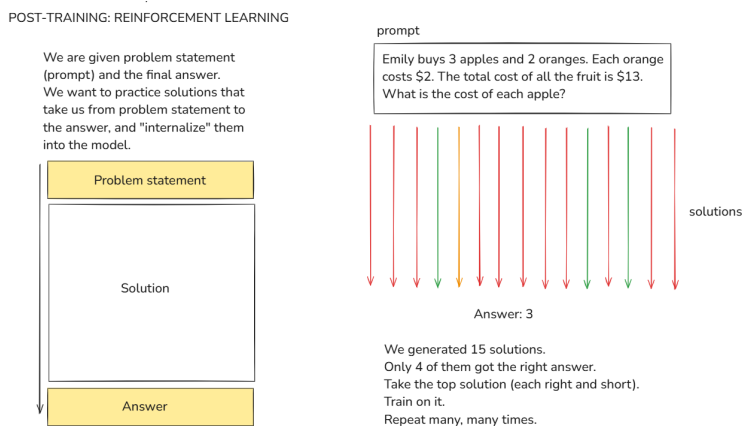


Figure 87: Schematic illustration of reinforcement learning.

We have a prompt. Then we tried many different solutions in parallel. Some of the solutions might go well. So they reach the right answer – they are in green. And some of the solutions might go poorly and may not reach the right answer – they are in red.

Now Emily's problem unfortunately is not the best example, because it's a trivial prompt. And as we saw even a two billion parameter model always gets it systematically right.

So it's not the best example in that sense. But let's exercise our imagination. Let's just suppose that the green ones⁷⁷ are good, and the red ones are bad, fig. 87.

Thus we imagine that we generated 15 solutions. And only four of them arrived at the right answer.

Now what we want to encourage the kinds of solutions that lead to right answers.

Whatever token sequences happened in red solutions, obviously something went wrong along the way somewhere. And this was not a good path to take through the solution.

And whatever token sequences happened in green solutions, things went pretty well in this situation.

We want our LLM to answer like in the green solutions.

The way we encourage this kind of a behavior in the future is to train the LLM on these sequences.

But these training sequences now are not coming from expert human annotators. There's no human who decided that this is the correct solution. This solution came from the model itself.

So the model is practicing here. It's tried out a few solutions. Four of them seem to have worked. So they will be included in the training data for the model.

This corresponds to a student looking at their solutions and thinking: "Okay, this one worked really well, so this is how I should be solving these kinds of problems."

In Emily's problem example there are many different ways to tweak a bit the methodology.

Just to give the core idea across, it's simplest is to take the single best solution out of these four, like say the one in yellow (fig. 87). This is the solution that not only led to the right answer, but maybe it had some other nice properties. Maybe it was the shortest one, or it looked nicest in some ways.

⁷⁷Green or yellow. Actually the yellow one is the best. See text.

There are other criteria you could think of. But we're going to decide that this is the top solution [the yellow one].

And it will be included in our training set.

Then the model will be slightly more likely – once you have updated your parameters – to take a path like that in similar problems in the future.

But you have to remember that we're going to run many different diverse prompts across lots of math problems and physics problems and whatever there might be.

So tens of thousands of prompts maybe to have in mind. There's thousands of solutions per prompt. And this is all happening at the same time

As we're iterating this process, the model is discovering for itself what kinds of token sequences lead it to correct answers. It's not coming from a human annotator. The model is playing in the playground – but it knows what it's trying to get to.

And it's discovering sequences that work for it. These are sequences that don't make any mental leaps. They seem to work reliably and statistically. They fully utilize the knowledge of the model as it has it.

So, this is the process of *reinforcement learning*. It's basically a guess and check. We guess many different types of solutions. We check them. And we're going to do more of what worked in the future. That is reinforcement learning.

In the context of what came before, we see now that the SFT model, the supervised fine tuning model, is still helpful, because it still initializes the model a little bit into the vicinity of the correct solutions.

It is an initialization of the model, in the sense that it encourages the model to take solutions, like write out solutions, and maybe it has an understanding of setting up a system of equations. Or maybe it's like talks through a solution. So it gets you into the vicinity of correct solutions.

But reinforcement learning is where everything gets dialed in. We really discover the solutions that work for the model; get the right answers; we encourage them; and then the model just gets better over time.

In summary, that is the high level process for how we train large language models.

Their training is very similar to how we educate pupils.

The only difference is that children go through chapters of books and they do all these different types of training exercises within the chapter of each book.

Instead, when we train AIs, it's almost like we do it stage by stage depending on the type of that stage.

1. First we do pre-training which, as we saw, is analogous to reading all the expository material.

We look at all the textbooks at the same time. And we read all the exposition. And we try to build a knowledge base.

2. Secondly, we go into the SFT stage which is really looking at all the fixed solutions from human experts. It is analogous to all the different kinds of worked solutions across all the textbooks.

We arrive at an SFT model which is able to imitate the experts but does so kind of blindly. It just does its best guess, trying to mimic statistically the expert behavior.

That's analogous to what you get when you look at all the worked solutions in the textbook.

3. Finally in the last stage, we do all the practice problems in the RL (reinforcement learning) stage.

Across all the textbooks, we also do all the practice problems. And that's how we get the RL model.

So on a high level the way we train LLMs is very similar to the process we use for training children.

02:27:47

18 DeepSeek-R1

The next point I would like to make is that actually these first two stages – pre-training and supervised fine-tuning – they’ve been around for years. They are very standard and everyone does them, all the different LLM providers.

It is this last stage – the RL training – that is still in its early development. It is not standard yet in the field.

This stage is nascent.

I actually skipped over a ton of little details in this process. But the high level idea is very simple: *it’s trial and error learning*. There’s a ton of details, however, and little mathematical nuances to performing the following:

- how you pick the solutions that are the best
- how much you train on them
- what is the prompt distribution
- how to set up the training run such that this actually works.

So there’s a lot of little details and knobs to the core idea that is very simple.

And so getting the details right in RL is not trivial.

Many companies like for example OpenAI and other LLM providers have experimented internally with reinforcement learning fine-tuning for LLMs for a while. But they’ve not talked about it publicly. It’s all done inside the company.

That’s why the paper from DeepSeek that came out very recently [on January 21, 2025] was such a big deal, fig. 88. It is a paper from this company called DeepSeek AI⁷⁸ in China.

⁷⁸Hangzhou DeepSeek Artificial Intelligence Basic Technology Research Co., Ltd.

This paper really talked very publicly about reinforcement learning fine-tuning training for large language models, and how incredibly important it is for large language models, and how it brings out a lot of reasoning capabilities in the models.

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI
research@deepseek.com

Abstract

We introduce our first-generation reasoning models, DeepSeek-R1-Zero and DeepSeek-R1. DeepSeek-R1-Zero, a model trained via large-scale reinforcement learning (RL) without supervised fine-tuning (SFT) as a preliminary step, demonstrates remarkable reasoning capabilities. Through RL, DeepSeek-R1-Zero naturally emerges with numerous powerful and intriguing reasoning behaviors. However, it encounters challenges such as poor readability, and language mixing. To address these issues and further enhance reasoning performance, we introduce DeepSeek-R1, which incorporates multi-stage training and cold-start data before RL. DeepSeek-R1 achieves performance comparable to OpenAI-o1-1217 on reasoning tasks. To support the research community, we open-source DeepSeek-R1-Zero, DeepSeek-R1, and six dense models (1.5B, 7B, 8B, 14B, 32B, 70B) distilled from DeepSeek-R1 based on Qwen and Llama.

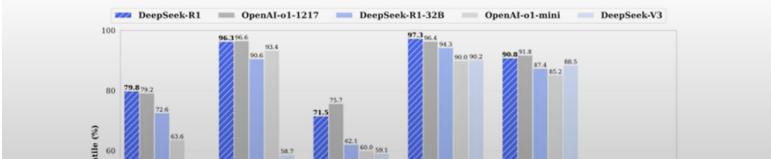


Figure 88: DeepSeek paper that shook the LLM community.

We shall presently look at it. This paper reinvigorated the public interest of using RL⁷⁹ for LLMs.

It gave a lot of the nitty gritty details that are needed to reproduce their results and actually get the RL stage to work for large langauge models.

Let’s go briefly through the DeepSeek-R1 paper, and what happens when you actually correctly apply RL to LLMs, and what that looks like, and what that gives you.

⁷⁹Sometimes called "reinforcement learning supervised fine-tuning training". But mostly it's just called the "RL" or the "RL stage".

The first thing I'll scroll to is figure 2 of their paper – fig. 89 here – where we are looking at the improvement in how the models are solving mathematical problems.

Table 2 | Comparison of DeepSeek-R1-Zero and OpenAI o1 models on reasoning-related benchmarks.

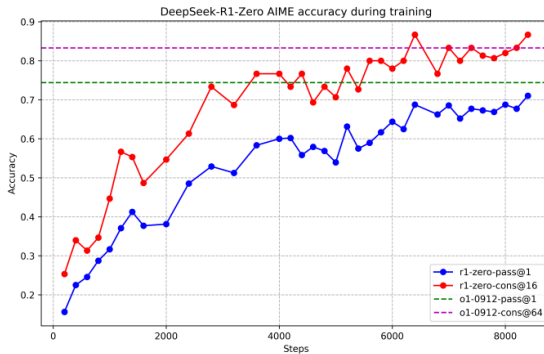


Figure 2 | AIME accuracy of DeepSeek-R1-Zero during training. For each question, we sample 16 responses and calculate the overall average accuracy to ensure a stable evaluation.

Figure 89: Accuracy of solving mathematical problems on the AIME bench marking.

This shows the accuracy of solving mathematical problems on the AIME accuracy measure [bench marking]. And then we can go to the web page. We can see the kinds of problems that are actually in these the kinds of math problems that are being measured here.

02:30:00

These are simple math problems.

You may pause the video and go to the paper <https://arxiv.org/pdf/2501.12948> These are the kinds of problems that basically the models are being asked to solve, fig. 90.

You can see that in the beginning they're not doing very well. But then as you update the model with this many thousands of steps, their accuracy keeps climbing.

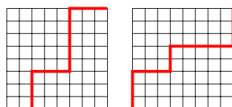
So the models are improving and they're solving these problems with a higher accuracy as you do this trial and error on a large data set of these kinds of problems. The models are discovering how to solve math problems.

6

2024 AIME I Solutions

Problem 6:

Consider the paths of length 16 that follow the lines from the lower left corner to the upper right corner on an 8×8 grid. Find the number of such paths that change direction exactly four times, as in the examples shown below.

**Solution:****Answer (294):**

Any such path must consist of 8 steps to the right and 8 steps upward in some order. First consider those paths whose first step is to the right. All such paths start and end with a horizontal segment and include two vertical segments with a horizontal segment in between. The path is completely determined by the choice of the positions of the two vertical segments and the height of the middle horizontal segment. There are $\binom{7}{2} = 21$ ways to select the positions of the two vertical segments and 7 ways to select the height of the middle horizontal segment. Thus there are $21 \cdot 7 = 147$ paths that begin with a horizontal step. By symmetry, there are also 147 paths that begin with a vertical step, so the total number of paths is $2 \cdot 147 = 294$.

Problem 7:

Find the greatest possible real part of

$$(75 + 117i)z + \frac{96 + 144i}{z},$$

where z is a complex number with $|z| = 4$. Here $i = \sqrt{-1}$.

Figure 90: Example of AIME problems.⁸⁰

But even more incredible than the *quantitative* kind of results of solving these problems with a higher accuracy, is the *qualitative* means by which the model achieves these results.

When we scroll down one of the figures (fig. 3 of DeepSeek paper), fig. 92 below, p. 150, what is interesting is that later on in the optimization, the model seems to be using [more time]. The average length per response goes up.

So the model seems to be using more tokens to get its higher accuracy results.

⁸⁰https://www.ptsd.k12.pa.us/Downloads/2024_AIME_I_Solutions.pdf

So it’s learning to create very long solutions.
Why are these solutions very long?
We can look at them qualitatively page 9 of DeepSeek paper – fig. 91 below.

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a + x}} = x$ is equal to
Response: <think> To solve the equation $\sqrt{a - \sqrt{a + x}} = x$, let’s start by squaring both ... $(\sqrt{a - \sqrt{a + x}})^2 = x^2 \implies a - \sqrt{a + x} = x^2$. Rearrange to isolate the inner square root term: $(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$... Wait, wait. Wait. That’s an aha moment I can flag here. Let’s reevaluate this step-by-step to identify if the correct sum can be ... We started with the equation: $\sqrt{a - \sqrt{a + x}} = x$ First, let’s square both sides: $a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$ Next, I could square both sides again, treating the equation:

Table 3 | An interesting “aha moment” of an intermediate version of DeepSeek-R1-Zero. The model learns to rethink using an anthropomorphic tone. This is also an aha moment for us, allowing us to witness the power and beauty of reinforcement learning.

Figure 91: An interesting “aha moment” of an intermediate version of DeepSeek-R1-Zero.

What they discover is that the model solution become very long – here’s a question, fig. 91, and here’s the answer from the model – partially because what the model learns – and *this is an emerging property of new optimization* – is that it is good for problem solving.

It starts to do stuff like this: "Wait, wait. Wait. That’s an aha moment I can flag here. Let’s reevaluate this, step by step, to identify if the correct sum can be..." (see fig. 91).

What is the model doing here ?
The model is basically re-evaluating steps.

It has learned that it works better, for accuracy, to try out lots of ideas, try something from different perspectives, retrace, reframe, backtrack...

It is doing a lot of the things that you and I are doing in the process of problem solving for mathematical questions.

But it's rediscovering what happens in your head not what you put down on the solution.

There is no human who can hard code this stuff in the ideal assistant response. This is only something that can be discovered in the process of reinforcement learning because you wouldn't know what to put here.

This simply turns out to work for the model and it improves its accuracy in problem solving.

The model learns what we call these "chains of thought" in your head and *it's an emergent property of the optimization.*⁸¹

That's what is bloating up the response length.

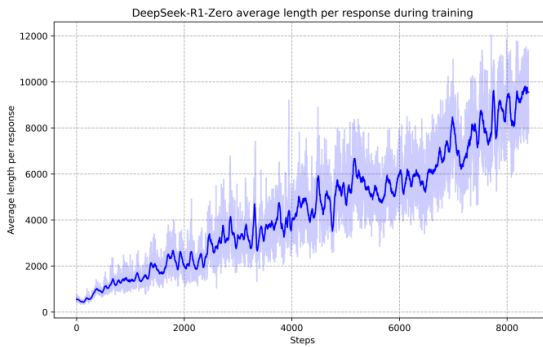


Figure 3 | The average response length of DeepSeek-R1-Zero on the training set during the RL process. DeepSeek-R1-Zero naturally learns to solve reasoning tasks with more thinking time.

Figure 92: DeepSeek-R1 answers take more time, as their accuracy increases.

But that's also what is increasing the accuracy of the problem solving.

⁸¹Generally speaking "an emergent feature" is some structure or pattern that appears in a dynamic process, and that cannot be explained with simple math. An old example is a model termites moving and eating chunks of wood https://lapasserelle.com/courses/advanced_finance/documents/termites.mp4

What's so incredible here is that the model is discovering ways to think. It's learning what I like to call *cognitive strategies* of how you manipulate a problem and how you approach it from different perspectives. How you pull in some analogies or do different kinds of things like that? How you try out many different things over time, check a result from different perspectives?

In short: How you solve problems?

But it's discovered by the reinforcement learning. It is extremely incredible to see this *emerge* in the optimization without having to hard code it anywhere.

The only thing we've given it are the correct answers. Then this comes out from trying to just solve them correctly – which is incredible.

Now let's go back to the problem that we've been working with [Emily's problem]. And let's take a look at what it would look like for this kind of a model – what we call reasoning or thinking model – to solve that problem.

When I pasted Emily's problem into ChatGPT 4o, I got the response shown in figure 83, page 135.

Let's take a look at what happens when you give this same query to what's called a *reasoning* or a *thinking* model. This is a model that was trained with *reinforcement learning*.

The model described in DeepSeek-R1 paper is available on `chat.deepseek.com`.

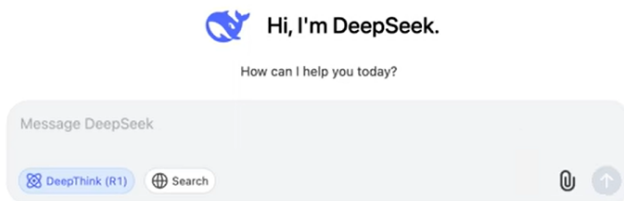


Figure 93: Main page of `chat.deepseek.com`

It is the company that developed it and is hosting it.

You have to make sure that the DeepThink(R1) button is turned on (bottom left of fig. 93), to get the R1 model as it is called.

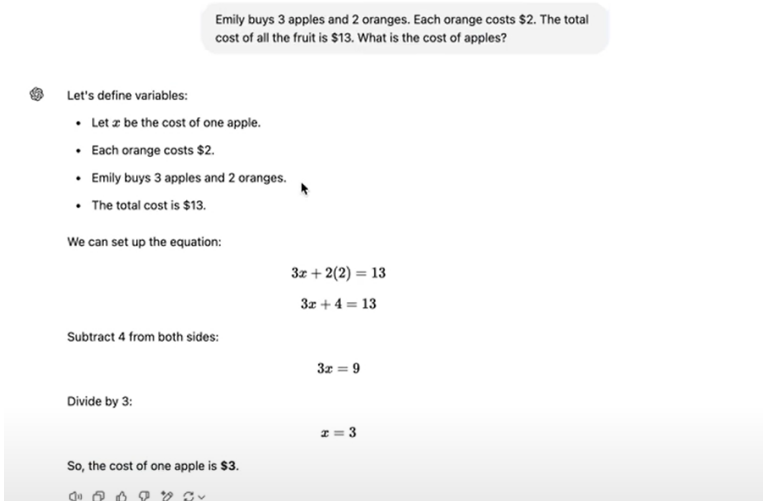
We can paste Emily's problem in the query box and run it.

Let's take a look at what happens now and what is the output of the model.

Okay so here's what it says, see fig. 95 next page.

And this is previously what we got using a basic SFT approach, a supervised fine-tuning approach, fig. 94 below. [That is the result after the pure post-training, without the third stage called RL.]

It was mimicking an expert solution.



Emily buys 3 apples and 2 oranges. Each orange costs \$2. The total cost of all the fruit is \$13. What is the cost of apples?

Let's define variables:

- Let x be the cost of one apple.
- Each orange costs \$2.
- Emily buys 3 apples and 2 oranges.
- The total cost is \$13.

We can set up the equation:

$$3x + 2(2) = 13$$
$$3x + 4 = 13$$

Subtract 4 from both sides:

$$3x = 9$$

Divide by 3:

$$x = 3$$

So, the cost of one apple is \$3.

Figure 94: Result using an SFT approach (without RL).

And this (fig. 95 next page) is what we get from the RL model.

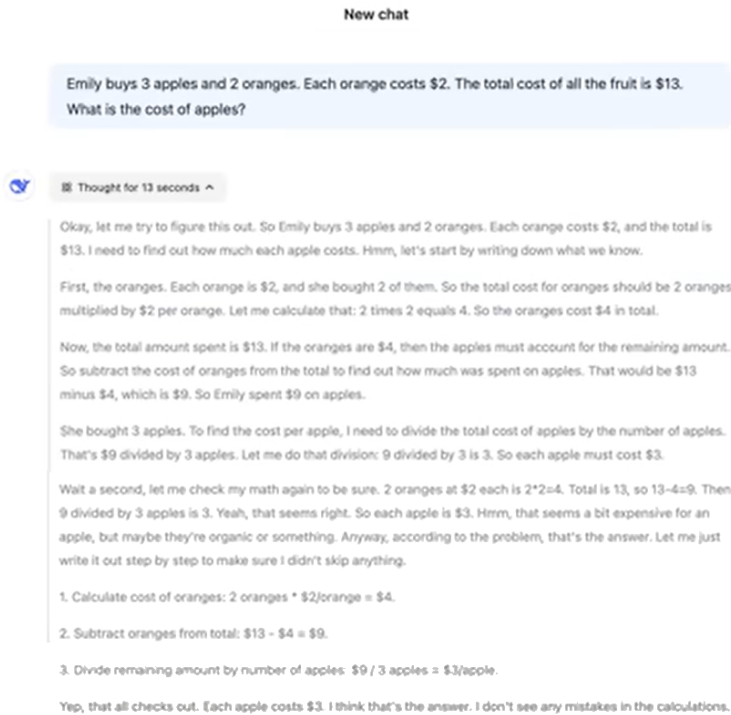


Figure 95: Result using the RL model.

The model itself begins saying: "Okay, let me try to figure this out. So Emily buys 3 apples and 2 oranges. Each orange cost \$2 and the total is \$13. I need to find out how much each apple blah blah blah."

As you're reading this, you can't escape thinking that this model is thinking.

It is definitely pursuing the solution. It derives that it must cost \$3.

And then it says: "Wait a second let me check my math again to be sure."

Then it tries it from a slightly different perspective.

And then it says: "Yeah, that seems right. So each apple is \$3...".

After some more checks, it adds: "Yep all that checks out. I think that's the answer. I don't see any mistakes. Let me see if there's another way to approach the problem, maybe setting up an equation. Let's let the cost of one apple be a dollar, then 3 apples cost $3a$ blah blah blah... Yep same answer. So definitely each apple is \$3". All right confident that that's correct."

[All this text is on fig. 95, in the answer from DeepSeek.]⁸²

Then what DeepSeek does is this: once it did the thinking process (shown in grey), it writes up the nice solution for the human (shown in black).

So the first part of DeepSeek answer (shown in grey) is more about the correctness aspect ; and the second part of DeepSeek answer (shown in black) is more about the presentation aspect, where it writes it out nicely and boxes in the correct answer at the bottom.

What's incredible about this is we get this thinking process of the model. And this is what's coming from the reinforcement learning process.

This is what's bloating the length of the token sequences, fig. 92. They're thinking and they're trying different ways.

It also you higher accuracy in problem solving, fig. 89. It is where we see these "aha moments", and these different strategies, and these ideas for how you can make sure that you're getting the correct answer, fig. 95.

⁸²See also how DeepSeek solves the ladder problem. At first glance, it is very impressive. Most LLMs fumble, make mistakes, give ridiculous answers, while DeepSeek gets to the right answer, and presents a "human-like" way of thinking and doing the math. https://lapassereille.com/LLM/ladder_problem_deepseek.pdf

However when we closely examine DeepSeek answer, we see that it is bizarre, because it presents intermediate steps that are wrong. For instance it says that the two ladders upper extremities rest on the walls at height w , which is also the width of the corridor.

It's not enough to pepper the response with "Let me check", "Looks good to me", or "No sweat" to make it a genuine logical reasoning.

The last point I wanted to make is some people are a little bit nervous about putting, you know, very sensitive data into `chat.deepseek.com` because this is a Chinese company. So people are a little bit careful and cagey with that.

However DeepSeek R1 is a model that was *released* by its producer. So it is an open source and open weights model. It is available for anyone to download and use.

You won't be able to run it, at full precision or full capacity, on a MacBook or other local device, as this is quite a large model.

But many companies are hosting the full largest model.

One of those that I like to use is called `together.ai`.

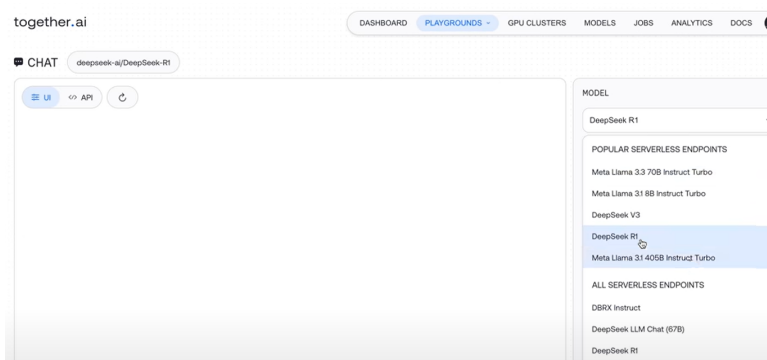


Figure 96: Playground of `together.ai`. You can then choose the LLM you want to work with.

When you go to `together.ai` you sign up. And you go to playground. You can select on the right DeepSeek R1.

There's a choice of models that you can select here. These are all state-of-the-art models.

It is similar to the Hugging Face inference playground that we've been playing with earlier.

But `together.ai` will usually host all the state-of-the-art models. So select DeepSeek R1.

We can ignore the settings that are offered us to tweak. I think the default settings will be alright.

So we put our usual Emily problem in the CHAT box on the left, fig. 97.

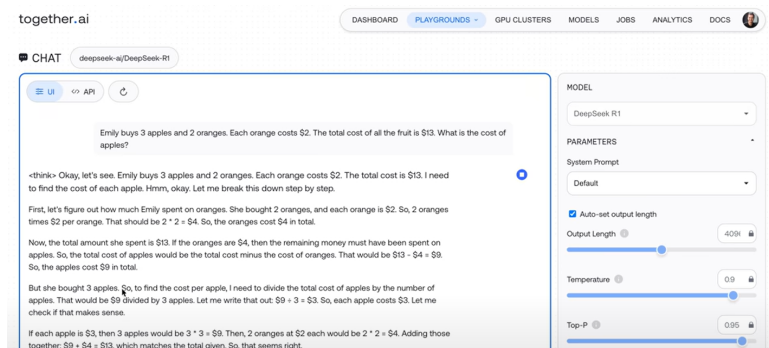


Figure 97: DeepSeek tested via `together.ai` playground.

Because the model was *released* by DeepSeek what we're getting here (fig. 97) should be essentially equivalent to what we got there (fig. 95).

Now, as we know, because of the randomness in the sampling, we're going to get something slightly different.

But in principle this should be identical in terms of the power of the model and you should be able to see the same things quantitatively and qualitatively.

But this test of DeepSeek is now totally under the control of an American company.

So that's DeepSeek and that's the what's called a *reasoning model*.⁸³

⁸³Calling a model based on reinforcement learning a "thinking" or "reasoning" system is still an exaggeration. It may *mimic* aspects of human thought — DeepSeek, for instance, uses expressions like "aha," "wait a second," or "let me check." But that doesn't mean it truly thinks.

02:37:10

Let's go back to ChatGPT, and look at various versions via together.ai. We see a choice of models in the drop down menu – on the left of the image in figure 98. Some of them, like o1, o3-mini, O3-mini-high, etc., are talking about uses *advanced reasoning*.

What this is referring to – "uses advanced reasoning" – is the fact that it was trained by reinforcement learning with techniques very similar to those of DeepSeek R1. This is per public statements of OpenAI employees.

So these (down the list) are thinking models, trained with RL. The models like GPT-4o or GPT-4o-mini, that you're getting in the free tier (on top of the list), you should think of them as mostly SFT models, supervised fine-tuning models.

DeepSeek's behavior recalls Joseph Weizenbaum's ELIZA program developed between 1964 and 1966, and Terry Winograd's SHRDLU, one of the earliest natural language understanding systems developed at MIT between 1968 and 1970. They also gave the impression they were understanding you and thinking.

ELIZA would simply repeat what the human interlocutor said:

Human: – I feel mistreated by my boss.

Machine: – You feel mistreated by your boss? Tell me how.

Human: – He systematically criticizes what I do

Machine: – So he systematically criticizes what you do?

etc.

SHRDLU actually was way more sophisticated: it allowed users to converse with a computer about actions in a small "block world," a simulated environment of simple geometric objects. The program could interpret commands such as "put the red block on the green cube," reason about their context, and respond coherently in English.

SHRDLU operated within a narrow, rule-based world, but its understanding was built on deterministic logic and linguistic heuristics.

Modern models like DeepSeek simulate reasoning by mimicking what they learned in their training sets of data.

Both create the illusion of understanding, *though through very different paradigms*: handcrafted symbolic rules in Winograd's case versus statistical pattern learning in modern RL systems.

In fact, one could argue that SHRDLU came closer to "thinking" than DeepSeek.

In any case, it is the transcriber's opinion that genuine reasoning models remain, as of 2025, a goal for the future.

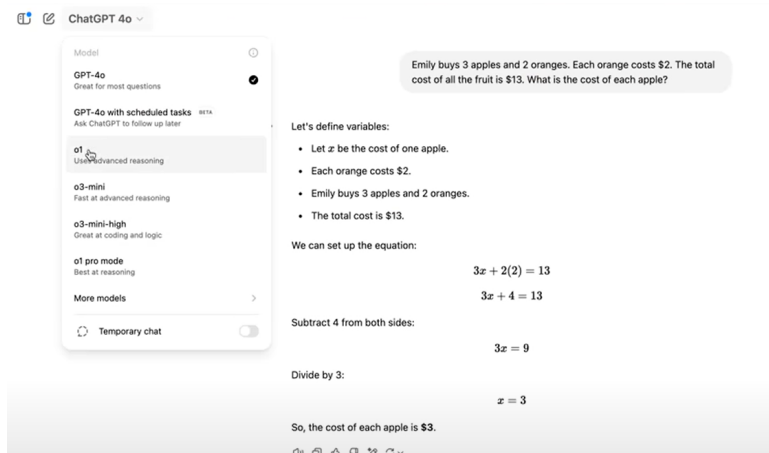


Figure 98: Back to ChatGPT.

The SFT models – "pure SFT" without the third training stage which is reinforcement learning – don't actually do this thinking as you see in the RL models.⁸⁴ Even though there's a little bit of reinforcement learning involved with these models – and I'll go that into that in a second – these are mostly SFT models.

So in the same way as what we saw here [when we chose DeepSeek R1 in the `together.ai` playground] we can pick one of the thinking models, like say o3-mini-high.

These models by the way might not be available to you unless you pay a ChatGPT subscription of either \$20 per month or \$200 per month for some of the top models.

So let's pick a thinking model and run it.

⁸⁴We saw that RL models, given a query and its final result, try numerous possible solutions. They first discard those that fail to reach the correct outcome, then among the successful ones, they select — and include in future training — the "best" solutions. Yet this still doesn't mean the model actually thinks. Any professor knows students who can write the problem, state the correct answer at the end, and fill the space in between with plenty of well-presented nonsense.

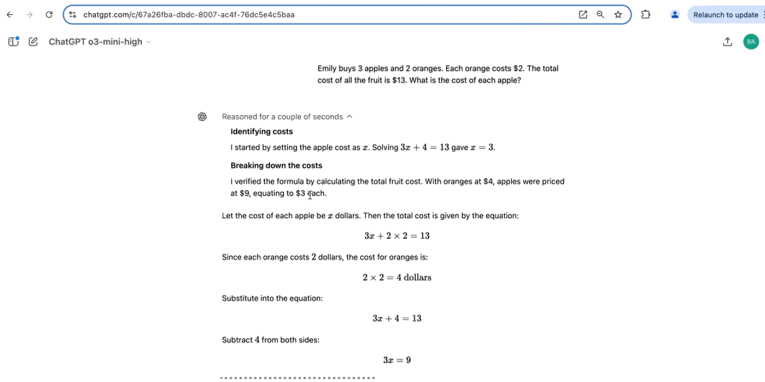


Figure 99: Selecting ChatGPT-o3-mini.

What's happening here, fig. 99, is it says reasoning and it starts to do stuff comparable to what we've already seen.

However what we're seeing with GPT-o3-mini is not exactly the stuff we saw with DeepSeek R1.

Even though under the hood the DeepSeek model produces these kinds of kind of chains of thought, *and shows them in its interface*, OpenAI chooses to not show the exact chains of thought in the web interface. It shows only little summaries of those chains of thought.

OpenAI does this, I think, partly because they are worried about what's called the *distillation risk*: that is that someone could come in and actually try to imitate those reasoning traces, and recover a lot of the reasoning performance by just imitating the reasoning chains of thought.

Therefore they hide them and they only show little summaries of them. So you're not getting exactly what you would get in DeepSeek with respect to the reasoning itself. Then OpenAI writes up the solution, fig. 99.

DeepSeek chat and OpenAI chat are more or less equivalent, even though we're not seeing the full under the hood details for OpenAI.

At present, in terms of performance, these OpenAI models (including RL) and DeepSeek models are roughly on par, I would say.

It's actually hard to tell because of the evaluations. But if you're paying \$200 per month to OpenAI, some of their models I believe still look better.

DeepSeek R1, nonetheless, for now is still a very solid choice for a thinking model that would be available to you, either on their website or any other website, because the model has *open weights*. It has been fully released, and you can just download it.

So that's thinking models.

What is the summary so far?

Well, we've talked about reinforcement learning and the fact that *thinking* emerges in the process of the optimization when we basically run RL on many math and code problems that have verifiable solutions.

They reach the answer « 3 » in Emily's problem], and present a nice looking reasonings.

You can access these thinking models – for instance DeepSeek or any inference provider – in platforms like `together.ai`. There you can choose which model you want to work with.

Various thinking models are also available in ChatGPT under any of the o1 or o3 models.

But models like ChatGPT-4o, etc., are not thinking models. You should think of them as mostly SFT models.

Now if you have a prompt that requires *advanced reasoning* and so on, you should probably use some of the thinking models or at least try them out.

Empirically, for a lot of my use, when you're asking a simpler question, like a knowledge based question or something like that, the use of a thinking model might be overkill. There's no need to think 30 seconds about some factual question. For that I will usually default to just GPT-4o.

Empirically about 80 to 90% of my use is just GPT-4o. And when I come across a very difficult problem like in math and code etc., I will reach for the thinking models. But then I have to wait a bit longer because they're thinking. So you can access these on ChatGPT, on DeepSeek, etc.

Also I wanted to point out aistudio.google.com.

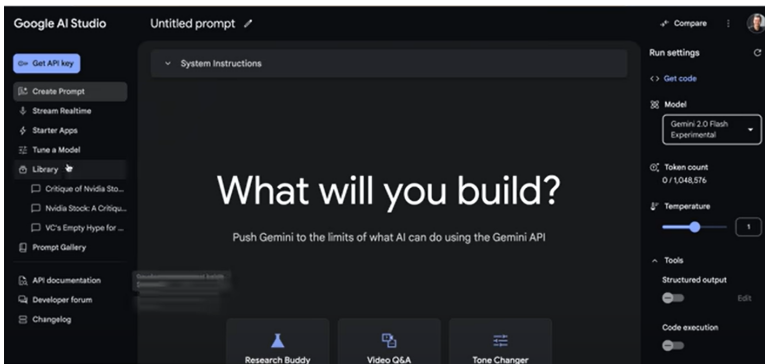


Figure 100: aistudio.google.com main window.

It looks really busy, really ugly, because Google is just unable to do this kind of stuff well.

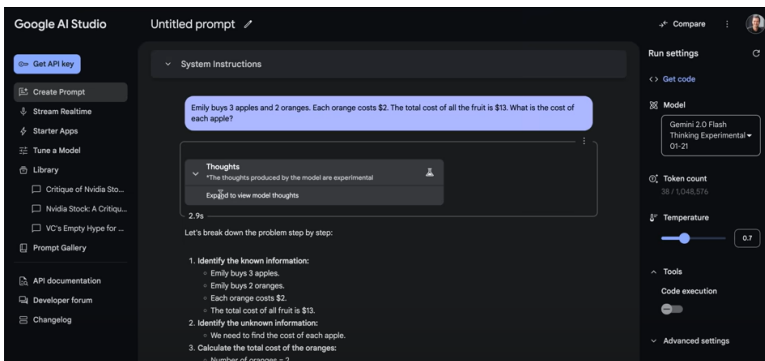


Figure 101: aistudio.google.com working on Emily's problem.

In aistudio.google.com you can choose among several models. If we choose *Gemini 2.0 Flash Thinking Experimental 01-21*, fig. 101, right column, that's also a kind of early experiment of a thinking model by Google.

We can give it our usual problem and click run, fig. 101. The thinking model will do something similar to what we saw with DeepSeek and ChatGPT-o3-mini. It also comes out with the right answer [answer : « 3 »] reached at the end of some reasoning. So Gemini also offers a thinking model.

Anthropic currently does not offer a thinking model.

Reinforcement learning (RL) is the frontier development of these LLMs. It is the new exciting stage. But getting the details right is difficult. That's why all these thinking models are currently experimental as of early 2025. This is the frontier development of pushing the performance on these very difficult problems using reasoning that is emerging in these optimizations.

02:42:00

19 AlphaGo

The discovery that reinforcement learning is an extremely powerful way of learning is not new in the field of AI.

And one place where we've already seen this demonstrated is in the game of Go:

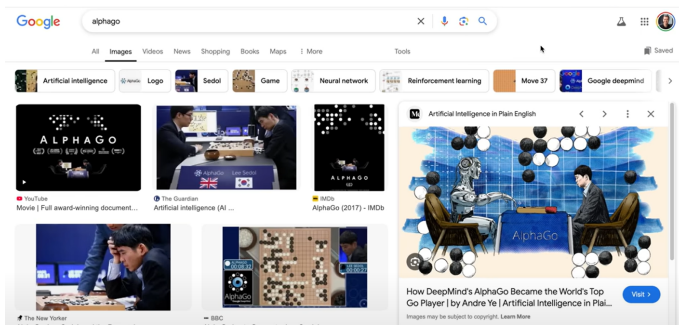


Figure 102: Search for "alphago" on Google search.

Famously DeepMind developed the system *AlphaGo* – and you can watch a movie about it – where the system is learning to play the game of Go against top human players.

They explained how they lead AlphaGo to world class level in <https://www.nature.com/articles/nature24270> published 19 October 2017 in Nature

Mastering the Game of Go without Human Knowledge

David Silver*, Julian Schrittwieser*, Karen Simonyan*, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, Demis Hassabis.

DeepMind, 5 New Street Square, London EC4A 3TW.

*These authors contributed equally to this work.

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, *AlphaGo* became the first program to defeat a world champion in the game of Go. The tree search in *AlphaGo* evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here, we introduce an algorithm based solely on reinforcement learning, without human data, guidance, or domain knowledge beyond game rules. *AlphaGo* becomes its own teacher: a neural network is trained to predict *AlphaGo*'s own move selections and also the winner of *AlphaGo*'s games. This neural network improves the strength of tree search, re-

Figure 103: Paper in which a Google team explained how AlphaGo "auto-trained" itself.

When we scroll down we find a really interesting plot that, I think, is kind of familiar to us (fig. 104, left plot).

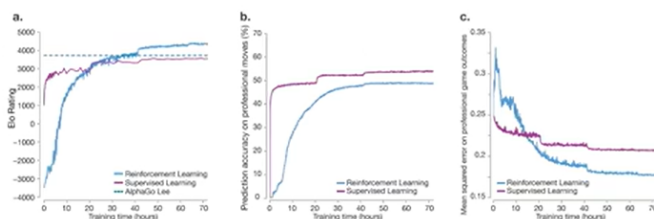


Figure 3: Empirical evaluation of *AlphaGo Zero*. **a** Performance of self-play reinforcement learning. The plot shows the performance of each MCTS player α_{θ_i} from each iteration i of reinforcement learning in *AlphaGo Zero*. Elo ratings were computed from evaluation games between different players, using 0.4 seconds of thinking time per move (see Methods). For comparison, a similar player trained by supervised learning from human data, using the *KGS* data-set, is also shown. **b** Prediction accuracy on human professional moves. The plot shows the accuracy of the

Figure 104: AlphaGo progressive improvement.

In LLMs, we are discovering the same curve in the more open domain of arbitrary problem solving, instead of on the closed specific domain of the game of Go.

What they saw – and we’re seeing this in LLMs as well as the RL stage training becomes more mature – is this progressive improvement of the ELO rating of the machine playing the game of Go.

The dotted horizontal line in fig. 104, left plot, is the ELO of Lee Sedol an extremely strong human player. What they are comparing is the strength of a model trained by supervised learning (purple curve) and a model trained by reinforcement learning (auto-training, light blue curve).

The supervised learning model is imitating human expert players. Therefore, if you just get a huge amount of games played by expert players in the game of Go, and you try to imitate them, you are going to improve. But then you top out. And you never overcome the really top players of in the game of Go like Lee Sedol.

You’re never going to pass above them because you’re just imitating human players. You can’t fundamentally go beyond a human player if you’re just imitating human players.

Using a technique similar to *reinforcement learning*, on the other hand, is significantly more powerful.

Reinforcement learning, for a game of Go, means that the system is playing moves that empirically and statistically lead to win, to winning the game.

AlphaGo, during its training, played against itself. It was using reinforcement learning to create roll-outs. So the diagram in figure 87, page 141, applies exactly here.

In AlphaGo there is no prompt.⁸⁵ And, because there’s no prompt, it’s just a fixed game of Go. But it’s trying out lots of plays. And then the games that lead to a win (instead of to a math result) are reinforced. They’re made stronger.

⁸⁵We could consider, though, that the board layout before it is AlphaGo’s turn is a kind of prompt.

Thus the system is learning basically the sequences of actions that empirically and statistically lead to winning the game.

Reinforcement learning is not constrained by human performance – that would sort of limit it. RL can do significantly better and overcome even the top players like Lee Sedol.

Probably DeepMind team could have run this training longer, but they just chose to crop it at some point, because this costs money. But this is a very powerful demonstration of reinforcement learning.⁸⁶

We're only starting to see hints of the diagram in fig. 104, left part, in larger language models for reasoning problems.

The lesson to draw from the AlphaGo experience is: *We're not going to get too far by just imitating human experts.*

We need to go beyond that, set up these little game environments [speaker shows again fig. 87, p. 141], and let the system discover reasoning paths, or ways of solving problems that are unique and that just work well.

On this aspect of uniqueness, notice that when you're doing reinforcement learning, nothing prevents you from veering off the distribution of how humans are playing the game. When we go to "alphago" search, fig. 102, one of the suggestions is "move 37".

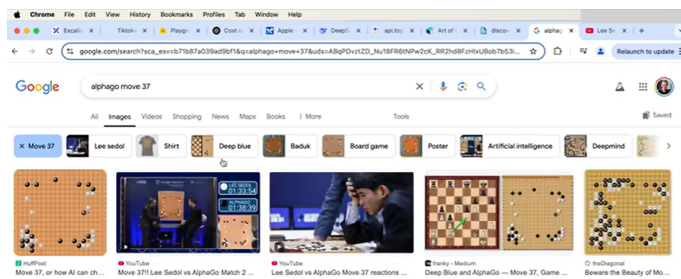


Figure 105: Search for "alphago move 37" on Google search.

⁸⁶AlphaGo did not win the official World Go Championship, but it achieved global recognition by beating the reigning world champion Ke Jie in May 2017 at the "Future of Go Summit" in Wuzhen, China.

Move 37 in AlphaGo is referring to a specific move AlphaGo played in a game.⁸⁷ No human expert would make such a move. The probability of this move to be played by a human player was evaluated to be about 1 in 10000. So it's a very rare move, but in retrospect it was a brilliant move.

It demonstrated that AlphaGo, in the process of reinforcement learning, discovered a strategy of playing that was unknown to humans, but in retrospect is brilliant.

Watch the YouTube video *Lee Sedol versus AlphaGo Move 37 reactions and analysis* <https://youtu.be/HT-UZki0Lv8>

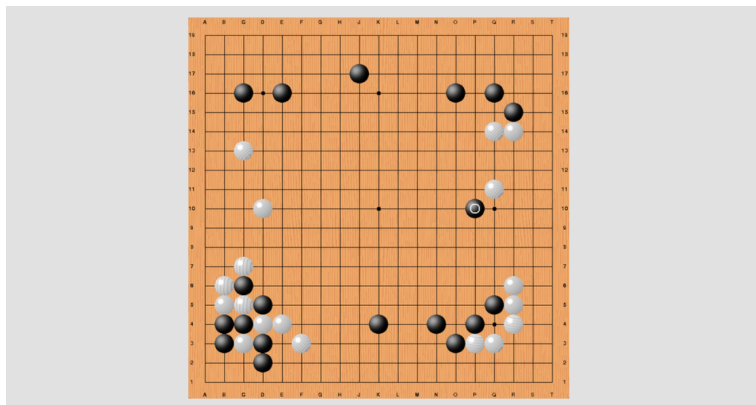


Figure 106: Layout immediately after AlphaGo (playing Black) has played Move 37 (marked with a circle). The black stone with the circle at coordinates 'Q-10' is the famous Move 37 from Game 2 against Lee Sedol.

Figure 106 shows what the board looked like right after AlphaGo's move. It was a very surprising move — I honestly thought it was a mistake when I first saw it. People were shocked because it was a move no human would normally play. Yet AlphaGo played it because, during its training,

⁸⁷Move 37 – the famously creative move by AlphaGo – occurred during Game Two of the match against Lee Sedol, on March 9, 2016. This move is widely regarded as a landmark moment in AI and Go history, stunning commentators and Go experts with its originality and depth.

that move ended up being evaluated as a good idea. It just happens not to be a kind of move that a human would make.

That is again the power of reinforcement learning. And in principle we can see the equivalence of that if we keep scaling up this paradigm in language models, see fig. 87 p. 141.

What that looks like is kind of unknown.

What does it mean to solve problems in such a way that even humans would not be able to get? How can you be better at reasoning or thinking than humans? How can you go beyond just a thinking human?

Maybe it means discovering analogies that humans would not be able to create. Or maybe it's like a new thinking strategy. It's kind of hard to think through.

Maybe it's a wholly new language that actually is not even English. Maybe it discovers its own language that is a lot better at thinking, because, unlike humans, the model is unconstrained to stick with English. So maybe it takes a different language to think in, or it discovers its own language.⁸⁸

In principle the behavior of the system is a lot less defined [than strict English]. It is open to do whatever works. It is also open to slowly drift from the distribution of its training data which is English.

But all of that can only be done if we have a very large diverse set of problems in which the strategy can be refined and perfected.

There is a lot of "frontier LM research" going on right now.

It is trying to create those kinds of prompt distributions that are large and diverse.

These are all like game environments in which the LLMs can practice their thinking.

⁸⁸Karpathy gets a bit carried away about reinforcement learning in training AlphaGo. Remember, though, that RL in large language models remains a kind of self-supervised fine-tuning ("auto-SFT"). In practice, it mainly encourages the model to mimic good answers and does not confer any *true logical reasoning abilities*.

It's like writing the practice problems, see figure 81 page 131. We have to create practice problems for all of domains of knowledge.

If we have practice problems, and tons of them, the models will be able to practice reinforcement learning, learn using RL them and create diagrams like fig. 104 (left part) but in the domain of open thinking, instead of a closed domain like the game of Go.

02:48:26

20 Reinforcement learning from human feedback (RLHF)

There's one more section within reinforcement learning that I wanted to cover. It is implementing RL in unverifiable domains.⁸⁹

So far all of the problems that we've looked at are in what's called verifiable domains – that is we can score any candidate solution very easily against a concrete answer.

In Emily's problem the answer is « 3 ». And we can very easily score these solutions against the answer of « 3 ».

Either we require the models to box in their answers and then we just check for equality of whatever is in the box with the correct answer that we know. Or you can also use what's called an LLM judge [still an automatic procedure].

⁸⁹In verifiable domains — for example, chess, Go, or code generation — the model's actions can be checked or evaluated objectively (and automatically by a program). And it is easy to build efficient reward functions to fine-tune the neural network.

By contrast, unverifiable domains are those where no such objective truth exists – or where success cannot be verified automatically by a program. These include creative writing, dialogue, problem reasoning, moral decisions, or open-ended question answering.

In such areas, we cannot easily compute whether one answer is "right". Human judgment (through reinforcement learning from human feedback, RLHF) must step in to build efficient reward functions. In 2025, it is a central limitation of applying RL to large language models.

The LLM judge looks at a solution, and it gets the answer, and scores the solution for whether it's consistent with the answer or not. And LLMs empirically are good enough at the current capability that they can do this [check solution in verifiable domains] fairly reliably.

So we can apply those kinds of techniques as well.

In any case, in verifiable domains, we have a concrete answer and we're just checking solutions again against it. And we can do this automatically with no humans in the loop.

The problem is that we can't apply the strategy in what's called unverifiable domains.

Usually these are, for example, creative writing tasks like write a joke about pelicans, or write a poem, or summarize a paragraph, or something like that.

In these kinds of domains, it becomes harder to score our different solutions to this problem.

For example writing a joke about pelicans, we can generate lots of different jokes of course, that's fine.

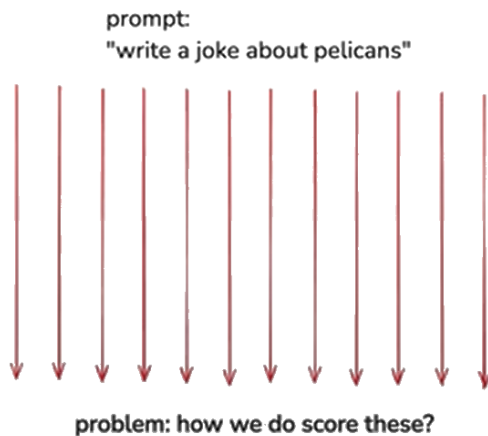


Figure 107: Generation of several jokes about pelicans. How to evaluate the different jokes we generated one way or another?

For example, we can go to ChatGPT and ask it to generate a joke about pelicans. We give it the prompt: "Write a super funny joke about pelicans."



Figure 108: Joke about pelicans generated by ChatGPT.

Write a super funny joke about pelicans.

Why do pelicans always carry so much stuff in their beaks?

Because they don't belican in backpacks 😊

Okay [that's not so funny]. We can try something else :

Write a super funny joke about pelicans.

Why don't pelicans ever pay for their drinks?

Because they always bill it to someone else! 😊

Haha...

Okay, these models are obviously not very good at humor.

Actually I think it's pretty fascinating because I think humor is secretly very difficult. And the model have the capability I think.

Anyway, in any case you could imagine creating lots of jokes.

The problem that we are facing is: how do we score them?

In principle, we could of course get a human to look at all these jokes just like I did right now.

The problem with that is, if you are doing reinforcement learning, you're going to be doing many thousands of updates.

And for each update you want to be looking at say thousands of prompts. And for each prompt you want to be potentially looking at looking at hundred or thousands of different kinds of generations.

So there's just way too many of these to look at.

In principle you could have a human inspect all of them and score them and decide that, okay, maybe this one is funny, and maybe this one is funny, and this one is funny...

And we could train on them to get the model to become slightly better at jokes – in the context of pelicans at least.

The problem is that it's way too much human time.

This is an unscalable strategy. We need some kind of automatic strategy for doing this.

One sort of solution to this was proposed in the paper shown in figure 109, that introduced what's called reinforcement learning from human feedback.

This was a paper from OpenAI at the time. And many of the authors are now co-founders in Anthropic.

It proposed a approach for basically doing reinforcement learning in unverifiable domains.

Fine-Tuning Language Models from Human Preferences

Daniel M. Ziegler* Nisan Stiennon* Jeffrey Wu Tom B. Brown
Alec Radford Dario Amodei Paul Christiano Geoffrey Irving

OpenAI

{dmz,nisan,jeffwu,tom,alec,damodei,paul,irving}@openai.com

Abstract

Reward learning enables the application of reinforcement learning (RL) to tasks where reward is defined by human judgment, building a model of reward by asking humans questions. Most work on reward learning has used simulated environments, but complex information about values is often expressed in natural language, and we believe reward learning for language is a key to making RL practical and safe for real-world tasks. In this paper, we build on advances in generative pretraining of language models to apply reward learning to four natural language tasks: continuing text with positive sentiment or physically descriptive language, and summarization tasks on the T1-DR

plex goals to AI agents are likely to both involve and require natural language, which is a rich medium for expressing value-laden concepts. Natural language is particularly important when an agent must communicate back to a human to help provide a more accurate supervisory signal (Irving et al., 2018; Christiano et al., 2018; Leike et al., 2018).

Natural language processing has seen substantial recent advances. One successful method has been to pretrain a large generative language model on a corpus of unsupervised data, then fine-tune the model for supervised NLP tasks (Dai and Le, 2015; Peters et al., 2018; Radford et al., 2018; Khandelwal et al., 2019). This method often substantially outperforms training on the supervised datasets from scratch, and a single pretrained language model often can be fine-tuned for state of the art performance on many different super-

: [cs.CL] 8 Jan 2020

Figure 109: OpenAI article on RLHF, January 2020

<https://arxiv.org/pdf/1909.08593v2>

Let's take a look at how that works. Fig. 110 is the cartoon diagram of the core ideas involved.

Naive approach:

Run RL as usual, of 1,000 updates of 1,000 prompts of 1,000 rollouts.
(cost: 1,000,000,000 scores from humans)

RLHF approach:

STEP 1:

Take 1,000 prompts, get 5 rollouts, order them from best to worst
(cost: 5,000 scores from humans)

STEP 2:

Train a neural net simulator of human preferences ("reward model")

STEP 3:

Run RL as usual, but using the simulator instead of actual humans

Figure 110: Core ideas involved in RLHF.

As I mentioned the native approach is, if we had an infinite amount of human time at our disposal, we could run RL in these unverified domains just fine.

For example we can run RL as usual if I have infinite human time. I want to do 1,000 updates where each update will be on 1,000 prompts. And in for each prompt we're going to have 1,000 roll-outs that we're scoring.

So we can run RL with this kind of a setup [top of fig. 110].

The problem is in the process of doing this, I will need to ask a human to evaluate a joke a total of 1 billion times. That's a lot of people looking at really terrible jokes. We don't want to do that.

Instead we want to take the RLHF approach.

In our RLHF approach, the core trick is that of indirection. We're going to involve humans just a little bit. The way we cheat is that we train a whole separate neural network that we call a *reward model*.

This neural network will imitate human scores.

So we're going to ask humans to score roll-outs. Then, we will imitate human scores using the reward model. This neural network will become a kind of simulator of human preferences.

Now that we have a neural network simulator, we can do RL against it. Instead of asking a real human, we're asking a simulated human for their score of a joke, as an example.

Once we have a simulator, we're in business, because we can query it as many times as we want. It's a wholly automatic process. We can now do reinforcement learning with respect to the simulator.

The simulator, as you might expect, is not going to be a perfect human. But if it's at least statistically similar to human judgment, then you might expect that this will do something. And in practice indeed it does.

Once we have a simulator we can do RL and everything works great.

Let me show you a cartoon diagram of what this process looks like, fig. 111. Although the details are not 100 like super important, it's just a core idea of how this works. So here I have a cartoon diagram of a hypothetical example of what training the reward model would look like

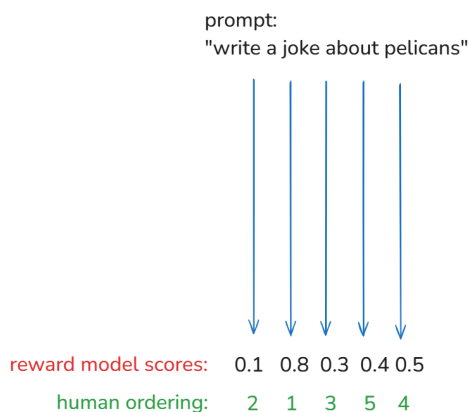


Figure 111: How RLHF treats the pelican jokes.

We have a prompt like "write a joke about pelicans". And then we have five separate roll-outs. These are all five different jokes just like the two corny ones generated by ChatGPT.

The first thing we're going to do is ask a human to order these jokes from best worst. The live of five green numbers at bottom of fig. 111, are the five ranks.

The human thought that the second joke is the best, the funniest, so it is ranked number one joke. The joke on the left is number two. We see joke number three, joke four. The next to last joke on the right is number five. It is the worst joke.

We're asking humans to order instead of give scores directly, because it's a bit of an easier task. It's easier for a human to give an ordering than to give numerical scores.

Now that is the supervision for the model. The human has ordered them and that is their contribution to the training process.

Now *separately*, we're going to ask a reward model about its scoring of these jokes.

The reward model is a wholly separate neural network, a completely separate neural net. It's also probably a transformer, but it's not a language model in the sense that it generates diverse language, etc. It's just a scoring model.

So the reward model will take as an input the prompt, that's the first thing. Secondly it will select a candidate joke. Those are the two inputs that go into the reward model.

For example the reward model would take the prompt and the joke below:

Write a super funny joke about pelicans.

Why don't pelicans ever pay for their drinks?

Because they always bill it to someone else!☺☺

The output of a reward model is a single number. This number is thought of as a score. It can range for example from zero to one.

Zero would be the worst score and one would be the best score.

In figure 111, the next to last line at bottom shows examples of what a hypothetical reward model, at some stage in the training process? would give to these jokes.

Giving a score of 0.1 is a very low score; 08 is a really high score and so on.

Now we compare the scores given by the reward model with the ordering given by the human. And there's a precise mathematical way to actually calculate a loss function and calculate a correspondence and update a model based on it.

But I just want to give you the intuition which is that, as an example, in fig. 111, for this second joke the the human thought that it was the funniest and the model kind of agreed: 08 is a relatively high score.

But this score should have been even higher. So after an update of the reward model, we would expect that this score actually grow maybe to, say, 0.81 or something.

For the joke on the left, the human and the reward model actually are in big disagreement, because the human thought that this was number two, but the reward model score is only 0.1. So the score of the joke on the left needs to be much higher.

After an update, on top of the supervision, this might grow to maybe 0.15 or something like that.

Then, if we look at the second joke from the right, the human thought that it was the worst joke. But the model actually gave it a fairly high score. So you might expect after the update, the score would come down to maybe 3 or 3.5 or something like that.

We're fundamentally doing what we did before, when we were updating the parameters of our neural network in the pre-training stage we studied a while ago. We're slightly nudging the predictions from the models using a neural network training process: we're trying to make the *reward model* scores be consistent with human ordering.

As we update the reward model on human data it becomes better and better simulator of the scores and orders that humans provide. Then it becomes the simulator of human preferences which we can then do RL against.

But critically we're not asking humans one billion times to look at a joke. We're maybe looking at a thousand prompts and five roll-outs each. So maybe 5,000 jokes that humans have to look at in total. And they just give the ordering, and then we're training the model to be consistent with that ordering.

I'm skipping over the mathematical details. I just want you to understand the high level idea that this reward model is basically giving us the scores. And we have a way of training it to be consistent with human orderings.

That's how RLHF works: we basically train simulators of humans and RL with respect to those simulators.

02:57:58

21 Upside and downside of RLHF

Now I want to talk about the upside and downside of reinforcement learning from human feedback (RLHF).

RLHF upside

We can run RL, in arbitrary domains! (even the unverifiable ones)
This (empirically) improves the performance of the model, possibly due to the "discriminator - generator gap":

In many cases, it is much easier to discriminate than to generate.

e.g. "Write a poem" vs. "Which of these 5 poems is best?"

Figure 112: RLHF upside.⁹⁰

⁹⁰From the Excalidraw whiteboard used by the speaker, see footnote 2 page 2.

On the upside, the first thing is that this allows us to run reinforcement learning which we know is incredibly powerful kind of set of techniques.

And it allows us to do it in arbitrary domains and including the ones that are unverifiable.

So things like summarization, poem writing, joke writing or any other creative writing really in domains outside of math and code, etc.

Now empirically what we see when we actually apply RLHF is that this is a way to improve the performance of the model.

I have a top answer for why that might be, but I don't actually know that it is like super well established on like why this is.

You can empirically observe that when you do RLHF correctly, the models you get are a little bit better. But as to why that is, I think is not as clear.

My best guess is that this is possibly mostly due to the *discriminator - generator gap*.

What that means is that in many cases it is significantly easier to discriminate than to generate, for humans.

In particular an example of this is, when we do supervised fine-tuning (SFT), we're asking humans to generate the ideal assistant response. In many cases, as I've shown – see section 10 Post-training data (conversations), page 67 – the ideal response is very simple to write.

But in many cases it might not be.

For example in summarization, or poem writing, or joke writing, how are you, as a human labeler, supposed to give the ideal response in these cases?⁹¹ It requires creative human writing to do that.

So RLHF sidesteps this, because we ask people a significantly easier question as data labelers. They're not asked to

⁹¹That's what good teachers do every day. They also *explain* to students why one way of writing a text is better than another. See William Zinsser, *On writing well*, Harper Perennial, 2006.

write poems directly, they're just given five poems from the model and they're just asked to order them. That's a much easier task for a human labeler to do.

I think it allows to produce a lot higher accuracy data, because we're not asking people to do the generation task which can be extremely difficult. We're not asking them to do creative writing. We're just trying to get them to distinguish between creative writings and find the ones that are best.

That is the signal that humans are providing: the ordering. It is their input into the system.

Then the system, in RLHF, discovers the kinds of responses that would be graded well by humans.⁹² That step of indication allows the models to become a bit better.

That is the upside of RLHF. It allows us to run RL. It empirically results in better models. It allows people to contribute their supervision, even without having to do very difficult tasks, like in the case of writing ideal responses.

Unfortunately RLHF also comes with significant downside.

RLHF downside

We are doing RL with respect to a lossy simulation of humans. It might be misleading!

Even more subtle:

RL discovers ways to "game" the model.

It discovers "adversarial examples" of the reward model.

E.g. after 1,000 updates, the top joke about pelicans is not the banger you want, but something totally non-sensical like "the the the the the the the the".

Figure 113: RLHF downside.

⁹²The reward model is able to find patterns – as in deep learning – that are in the human procedures, but are not visible.

The main one is that we are doing reinforcement learning not with respect to humans and human judgment, but with respect to a lossy simulation of humans. This lossy simulation could be misleading, because it's just a simulation, it's just a language model that's producing scores. It might not perfectly reflect the opinion of an actual human with an brain in all the possible different cases. That's the first downside,

There is actually something even more subtle and devious going on, that really dramatically holds back RLHF as a technique that we can really scale to significantly smart systems.

It is that reinforcement learning is extremely good at discovering a way to game the model, to game the simulation.

This reward model that we're constructing here, fig. 111, p. 173, that gives the scores... these models are transformers.

These transformers are massive neural nets. They have billions of parameters and they imitate humans. But they do so in a simulation way.

Now the problem is that these are massive complicated systems, there's a billion parameters here that are outputting a single score.

It turns out that there are ways to game these models.

You can find kinds of inputs that were not part of their training set. And these inputs inexplicably get very high scores, but in a fake way.

Very often what you find if you run RLHF for very long, for example if we do 1,000 updates, which is a lot of updates, you might expect that your jokes are getting better and that you're getting real bangers about pelicans.

But that's not exactly what happens. What happens is that in the first few hundred steps the jokes about pelicans are probably improving a little bit. Then they actually dramatically fall off the cliff and you start to get extremely nonsensical results.

For example the top joke about pelicans starts to be "the the the the the the the". And this makes no sense. When you look at it, why should this be a top joke?

But when you take "the the the the the the..." and you plug it into your reward model, you'd expect score of zero. But actually the reward model loves this as a joke. It will tell you "that the the the the..." has a score of 1.0. This is a top joke. And this makes no sense.

It's because these models are just simulations of humans and they're massive neural nets, and you can find inputs at the bottom that get into the part of the input space that gives you nonsensical results.

These examples are what's called adversarial examples and I'm not going to go into the topic too much.

But these are adversarial inputs to the model. They are specific little inputs that go between the nooks and crannies of the model and give nonsensical results at the top.

Now here's what you might imagine doing. You say: okay "the the the..." is obviously not a score of 1. It's obviously a low score.

So let's take "the the the the the..." Let's add it to the data set and give it a ranking that is extremely bad – like a ranking of five in our example.

Your model will learn that "the the the the..." should have a very low score. It will give it score of zero.

The problem is that there will always be an infinite number of nonsensical adversarial examples hiding in the model.

If you iterate this process many times and you keep adding nonsensical stuff to your reward model, and giving it very low scores, you'll never win the game.

You can do this many many rounds, and reinforcement learning, if you run it long enough, will always find a way to game the model. It will discover adversarial examples. It will get really high scores with nonsensical results.

And fundamentally this is because our scoring function is a giant neural net and RL is extremely good at finding just the ways to trick it.

So to make a long story short, you always run RLHF for maybe a few hundred updates; the model is getting better; and then you have to crop it. And you are done.

You can't run too much using this reward model, because the optimization will start to go awry. So you basically crop it, and you call it and you ship it.

You can improve the reward model, but you'll come across these situations eventually at some point.

So RLHF... basically what I usually say is that RLHF is not RL.

And what I mean by that is: RLHF is RL obviously, but it's not RL in the magical sense. This is not RL that you can run indefinitely.

These kinds of problems where you are getting concrete correct answer (see fig. 87, p. 141) you cannot game this as easily.⁹³ You either got the correct answer or you didn't. And the scoring function is much simpler. You're just looking at the boxed area and seeing if the result is correct. So it's very difficult to game these functions. But gaming a reward model is possible.

Now in these verifiable domains you can run RL indefinitely. You could run for tens of thousands, hundreds of thousands of steps and discover all kinds of really crazy strategies, that we might not even ever think about, of performing really well for all these problems.

⁹³In this context, the speaker uses "to game" in the standard informal sense of "to exploit" or "to trick" a system to get favorable results without genuinely solving the underlying problem.

When he says "you can't game this as easily" or "it's possible to game the reward model," he means:

- In tasks where the answer can be objectively verified (like math or code), the model either gets it right or wrong — there's no easy way to "cheat."
- But in unverifiable tasks (like creative writing, or producing jokes), models can "game" the system — that is, produce outputs that look good according to a flawed or subjective reward model, without truly understanding or reasoning correctly.

In the game of Go there's no way to to tricking the winning of a game or the losing of a game. We have a perfect simulator. We know here all the stones are placed. And we can calculate whether someone has won or not. There's no way to game that. So you can do RL indefinitely. And you can eventually beat even Lee Sedol.

But with models in unverifiable domains, which are gameable, you cannot repeat this process indefinitely.

So I see RLHF as not real RL, because the reward function is gameable. It's more like in the realm of little fine-tuning. It's a little improvement. But it's not something that is fundamentally set up correctly, where you can insert more compute, run for longer, and get much better and magical results. It's not RL in that sense, because it lacks magic. It can fine-tune your model and get a better performance.

And indeed if we go back to ChatGPT

Emily buys 3 apples and 2 oranges. Each orange costs \$2. The total cost of all the fruit is \$13. What is the cost of each apple?

Reasoned for a couple of seconds ^

Identifying costs
I started by setting the apple cost as x . Solving $3x + 4 = 13$ gave $x = 3$.

Breaking down the costs
I verified the formula by calculating the total fruit cost. With oranges at \$4, apples were priced at \$9, equating to \$3 each.

Let the cost of each apple be x dollars. Then the total cost is given by the equation:

$$3x + 2 \times 2 = 13$$

Since each orange costs 2 dollars, the cost for oranges is:

$$2 \times 2 = 4 \text{ dollars}$$

Substitute into the equation:

$$3x + 4 = 13$$

Subtract 4 from both sides:

$$3x = 9$$

Divide by 3:

$$x = 3$$

Thus, each apple costs 3 dollars.

Figure 114: GPT 4o solution of Emily's problem.

the GPT 4o model has gone through RLHF because it works well but it's just not RL in the same sense.

RLHF is like a little fine tuning that slightly improves your model. That is the way I would think about it.

03.06.57

We have covered most of the technical content I wanted to cover. I took you through the three major stages and paradigms of training these models:

1. pre-training
2. supervised fine tuning
3. reinforcement learning

I showed you that they loosely correspond to the process we already use for teaching children (see the textbook in figure 81, page 131).

1. pre-training is like the basic knowledge acquisition of reading exposition
2. supervised fine tuning is the process of looking at lots and lots of worked examples and imitating experts
3. reinforcement learning is the practice problems

The only difference is that we now have to effectively write textbooks for LLMs and AI across all the disciplines of human knowledge, and also in all the cases where we actually would like them to work, like code and math, and basically all the other disciplines.

So we're in the process of writing textbooks for them, refining all the algorithms that I've presented on the high level, and then of course doing a really good job at the execution of training these models at scale and efficiently.

I didn't go into too many details, but the pre-training consists of extremely large and complicated distributed jobs that have to run over tens of thousands or even hundreds of thousands of GPUs.

And the engineering that goes into this is really at the state-of-the-art of what's possible with computers at that scale.

I didn't cover that aspect too much but this is very serious and they were underlying ultimately all these very simple algorithms that we saw.

I also talked about the theory of mind a little bit of these models.

The thing I want you to take away is that these LLMs are really good, but they're extremely useful as tools for your work.

You shouldn't trust them fully and I showed you some examples of that.

Even though we have mitigations for hallucinations, the models are not perfect and they will hallucinate still.

It's gotten better over time and it will continue to get better, but they can hallucinate.

In other words, in addition to that I covered what I call the Swiss cheese model of LLM capabilities that you should have in your mind.

Swiss cheese model of LLM capabilities of current models:

- some things work really well,
- some things (almost at random) show brittleness.



Figure 115: Swiss model of LLMs.

The models are incredibly good across so many different disciplines, but then fail almost randomly in some unique cases.

For example: what is bigger, 9.11 or 9.9? (see p. 127) – the model doesn't know.

But simultaneously it can turn around and solve Olympiad questions.⁹⁴

So this is a hole in the Swiss cheese. And there are many of them. And you don't want to trip over them.

So don't treat these models as infallible models. Check their work. Use them as tools. Use them for inspiration. Use them for the first draft. But work with them as tools. And be ultimately responsible for the product of your work.

That's what I wanted to talk about. This is how they're trained. And this is what they are.

03:09:41

22 Preview of things to come

Let's turn to what are some of the future capabilities of these models, probably what's coming down the pipe. And also where can you find these models ?

⁹⁴A recent large language model (LLM) developed by OpenAI was able to solve several problems from the 2025 International Mathematical Olympiad (IMO), including the following example:

Example IMO Problem (2025):

Given are positive integers a, b, c such that $a + b + c = 2025$. Prove that:

$$\frac{a}{b+c} + \frac{b}{c+a} + \frac{c}{a+b} \geq \frac{3}{2}$$

OpenAI's LLM produced a valid proof for this inequality and similar problems directly in natural language, under official competition rules ("pen and paper" style reasoning, no internet, complete explanations). The model's solutions were scored by IMO medalists, who confirmed their correctness.

This achievement demonstrates the model's ability to handle IMO-level contest mathematics with creative multi-step reasoning, previously thought to be beyond the reach of neural networks.

I have a few bullet points on some of the things that you can expect coming down the pipe.

PREVIEW OF THINGS TO COME

- multimodal (not just text but audio, images, video, natural conversations)
- tasks -> agents (long, coherent, error-correcting contexts)
- pervasive, invisible
- computer-using
- test-time training?, etc.

Figure 116: Preview of things to come.⁹⁵

The first thing you'll notice is that the models will very rapidly become multimodal. Everything I talked about above concerned text, but very soon we'll have LLMs that can not just handle text, but they can also operate natively and very easily over audio, so they can hear and speak, and also images, so they can see and paint.

We're already seeing the beginnings of all of this, but this will be all done natively inside the language model. And this will enable kind of like natural conversations

Roughly speaking the reason that this is actually no different from everything we've covered above, is that as a baseline you can tokenize audio and images and apply the exact same approaches of everything that we've talked about above.

So it's not a fundamental change. It's just we have to add some tokens.

As an example for tokenizing audio, we can look at slices of the spectrogram of the audio signal. And we can tokenize that, and just add more tokens that suddenly represent audio and just add them into the context windows and train on them just like above.

The same for images, we can use patches. And we can separately tokenize patches. And then what is an image? An

⁹⁵From large Excalidraw whiteboard, see footnote nr 2 page 2.

image is just a sequence of tokens. And this actually works. And there's a lot of early work in this direction.

So we can just create streams of tokens that are representing audio, images, as well as text, and intersperse them and handle them all simultaneously in a single model. So that's one example of multimodality.

Secondly, something that people are very interested in is this: currently most of the work is that we're handing individual tasks to the models on a silver platter, like "please solve this task for me". And the model does this little task.

But it's up to us to still organize a coherent execution of tasks to perform jobs.

The models are not yet at the capability required to do this in a coherent error correcting way over long periods of time. They're not able to fully string together tasks to perform these longer running jobs. But they're getting there and this is improving over time.

Probably what's going to happen here is we shall start to see what's called agents which perform tasks over time. And you, you supervise them and you watch their work and they come up to once in a while report progress and so on.

So we're going to see more long running agents tasks that don't just take a few seconds of response, but many tens of seconds or even minutes or hours over time.⁹⁶

But these models are not infallible as we talked about above. So all of this will require supervision.

For example, in factories people talk about the human to robot ratio for automation. I think we're going to see something similar in the digital space, where we are going to be talking about human to agent ratios – where humans becomes a lot more supervisors of tasks carried out by agents, in the digital domain.

⁹⁶Computer programs in the first domain of computer science have been doing this for decades (see footnote 63, page 113). However, they are entirely programmed and controlled by humans. LLMs, by contrast, have much greater freedom to operate, explore, discover, and come up with unexpected results.

Next, I think everything is going to become a lot more pervasive and invisible. It will be integrated into the tools and everywhere and, in addition, computer using.

Right now these models aren't able to take actions on your behalf. If you remember when we saw ChatGPT launch the operator, that's one early example of that, where you can actually hand off control to the model to perform keyboard and mouse actions on your behalf. That's also something that I think is very interesting.

The last point I have here (fig. 116) is a general comment that there's still a lot of research to potentially do in this domain.

One example is something along the lines of test time training. Remember that everything we've done above and that we talked about has two major stages.

- There's first the training stage where we tune the parameters of the model to perform the tasks well.
- Once we get the parameters, we fix them. And then we deploy the model for inference.

From there the model is fixed. It doesn't change anymore. It doesn't learn from all the stuff that it's doing at test time.⁹⁷ It has a fixed collection of parameters. The only thing that is changing is now the token inside the context windows.

So the only type of learning or test time learning that the model has access to is in-context learning of its dynamically

⁹⁷This is a major difference from humans. Once they have acquired a know-how and begin using it, people continue to develop new knowledge and skills through practice. Not so with LLMs. It can be very frustrating for users to show a model something, only for it to ignore that input beyond the current session. Once the conversation ends, everything demonstrated to the model is erased from its memory. LLMs could advance tremendously if they were able to learn from such working interactions with humans. But that remains a difficult goal: it would require modifying the model's parameters and filtering out malicious or ill-intentioned information.

adjustable context window depending on what it's doing at test time.

I think this is still different from humans who actually are able to actually learn depending on what they're doing – especially when you sleep for example, when your brain is updating your parameters or something like that. There's no equivalent of that currently in these models and tools.

There's a lot more wonky ideas I think that are to be explored still.

I think that it will be necessary because the context window is a finite and precious resource, and especially once we start to tackle very long running multimodal tasks and we're putting in videos and these token, windows will grow extremely large, like not thousands or even hundreds of thousands but significantly beyond that. The only kind of trick we have available to us right now is to make the context windows longer. But I think that that approach by itself will not scale to actual long running tasks that are multimodal over time.

So new ideas are needed in some of those disciplines, in some of those kind of cases, in the main, where these tasks are going to require very long contexts.

Those are some examples of some of the things you can expect coming down the pipe.

03:15:15

23 Keeping track of LLMs

Let's turn to where you can keep track of this progress.

WHERE TO KEEP TRACK OF THEM

- reference <https://lmarena.ai/>
- subscribe to <https://buttondown.com/ainews>
- X / Twitter

Figure 117: Keeping track of LLMs.

How to keep up to date with the latest and the greatest of what’s happening in the field?

The three resources that I have consistently used to stay up to date are

- 1. El Marena
- 2. Subscriptions
- 3. X/Twitter

Number one: El Marena. Let me show you what it is.

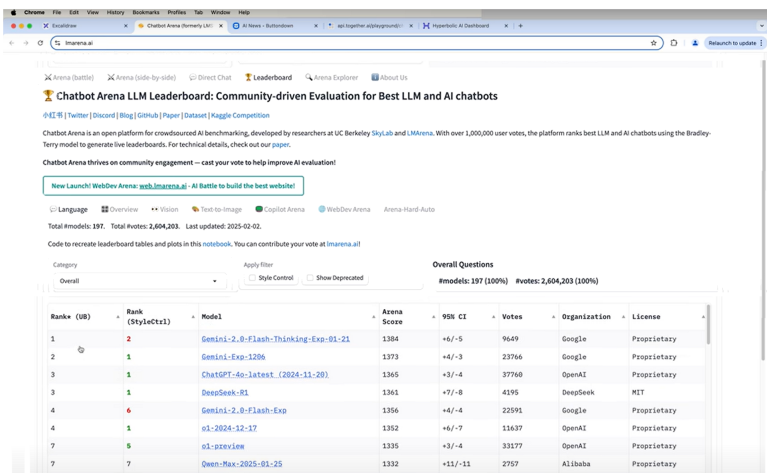


Figure 118: <https://lmarena.ai/> accessed in February 2025.⁹⁸

This is an LLM leader board. It ranks all the top models. The ranking is based on human comparisons. Humans prompt these models and they get to judge which one gives a better answer. They don’t know which model is which. They’re just looking at which model is the better answer.

You can calculate a ranking. Then you get some results.

⁹⁸larger figure https://lapasserelle.com/documents/fig_118.png

What you can see in fig. 118 is the different organizations, like Google Gemini for example, that produce these models.

When you click on any one of these, it takes you to the place where that model is hosted, and it opens on their main page.

We see that Google is currently on top with OpenAI right behind. We see DeepSeek in position number three.

The reason this is a big deal is the last column in fig. 118. You see *license: DeepSeek is an MIT license model*.⁹⁹

It's open weights. Anyone can use these weights. Anyone can download them. Anyone can host their own version of Deep seek. And they can use it in what whatever way they like. It's not a proprietary model, the sources of which you would not have access to. It's an open weight release. It is kind of unprecedented that a model this strong was released with open weights so pretty cool from the team.

Next down we have a few more models from Google and OpenAI. Then when you continue to scroll down, you start to see some other Usual Suspects. So xAI, then Anthropic with Sonnet, at number 14. Then Meta with Llama, etc.

Llama, similar to DeepSeek, is an open weights model, but it's down the list as opposed to near the top.

Now I will say that this leader board was really good for a long time. I do think, however, that in the last few months it's become a little bit gamed (= tricked). And I don't trust it as much as I used to.

Just empirically I think that a lot of people for example are using a Sonnet from Anthropic. It is a really good model. Yet it's all the way down here in number 14.

⁹⁹DeepSeek is a Chinese AI company based in Hangzhou. However, as noted in section 18, page 145, it has released its model. DeepSeek-R1 was released under the MIT open-source license. Many companies have since downloaded the model in full, and it can be run independently without visiting any Chinese website. According to the LMArena site, institutions such as MIT use DeepSeek-R1 for research and benchmarking purposes, but they do so under the model's open-source MIT license rather than through a special agreement.

Conversely I think not as many people are using Gemini as its ranking – at the top of the list – would suggest.¹⁰⁰

So, use El Marena ranking as a first pass. But try out a few of the models for your tasks and see which one performs better.

The second thing that I would point to is AI newsletter.

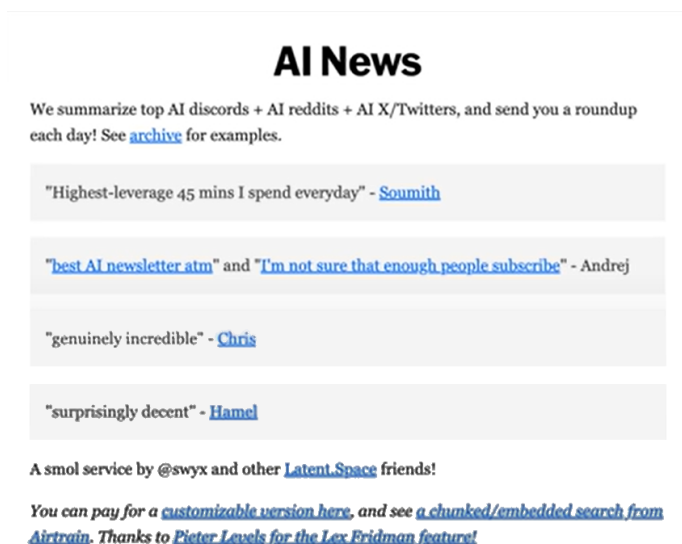


Figure 119: AI newsletter <https://buttondown.com/ainews>.

AI News is not very creatively named but it is a very good newsletter produced by swyx¹⁰¹ and friends. So thank you for maintaining it. It's been very helpful to me because it is extremely comprehensive.

¹⁰⁰For instance, the transcriber cancelled his subscription to Gemini after the model, in response to prompts like "tell me the end of such and such [famous and well known] novel", invented new endings different from what everybody knows.

¹⁰¹Shawn Wang, better known by his online handle "swyx", is a Singapore-born software engineer, writer, and AI educator, formerly at AWS and Temporal, and now known for his projects Latent Space, AI Engineer, and his widely read AI News and DevTools newsletters.

If you go to archives, you'll see that it is produced almost every other day. Some of it is written by humans and curated by humans, but a lot of it is constructed automatically with LLMs. So you're probably not missing anything major if you go through it. Of course you're probably not going to go through it because it's so long, but these summaries all the way up top are quite good and, I think, are produced under some human supervision. So this has been very helpful to me.

The last source I would point to is just X (ex-Twitter). A lot of AI happens on X and so you can just follow people who you like and trust and get all your latest and greatest uh on X as well.

Those are the major places that have worked for me over time.

03:18:34

24 Where to find LLMs

Finally a few words on where you can find the models and where can you use them.

WHERE TO FIND THEM

- Proprietary models: on the respective websites of the LLM providers
- Open weights models (DeepSeek, Llama): an inference provider, e.g. TogetherAI
- Run them locally! LMStudio

Figure 120: Where to find LLMs.

The first one I would say is for any of the biggest proprietary models, you just have to go to the website of that LLM provider.

For example for OpenAI, <https://chatgpt.com/>, or simply <https://chat.com/> I believe actually works now.

For Gemini, it's <https://gemini.google.com> or <https://aistudio.google.com/>. They have two for some reason I don't really understand. No one does.

For open-weights models like DeepSeek or Llama and others, you typically have to go through an inference provider — that is, a hosting service that runs the models and exposes them through an API.

My favorite is <https://together.ai> :

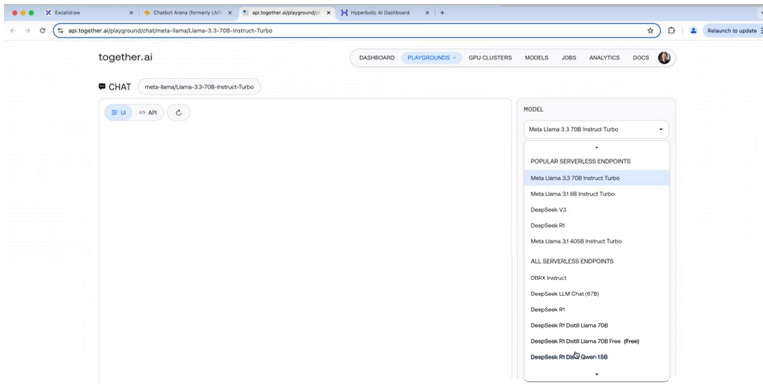


Figure 121: Playground of together.ai

I showed you that when you go to the playground of together.ai, you can pick among lots of different models (with the drop-down menu on the right of the screen). All of these are open models of different types and you can talk to them here [in the CHAT window] as an example.

If you want to use a *base model* (as opposed to a fine-tuned assistant model), it is a bit harder to find those on major inference providers. Most hosting platforms focus on chat or assistant models, not the raw base versions. For example, I could not find many base models on the usual providers.

That is why, when I need a base model, I usually go to Hyperbolic (<https://www.hyperbolic.ai/>), because they host the Llama 3.1 base model, which I really like. You can interact with it directly on their site. As far as I know, Hyperbolic is one of the better places to access a base model, and I wish

more providers would host them since they are quite valuable and interesting to work with in certain cases.

Finally, you can also use some of the smaller models and run them locally. For example, with DeepSeek, you will not be able to run the largest model on your MacBook, but there are smaller versions of the DeepSeek model, known as distilled versions. You can also run these models at lower precision — not at the native precision used by the full models, like FP8 for DeepSeek or BF16 for Llama, but at much lower precision levels.

Don't worry if you don't fully understand those details. You can run smaller versions of these models that have been distilled and that operate at even lower precision. This allows them to fit on your computer, so you can actually run fairly capable models on your laptop.

My favorite place to go is usually LM Studio (<https://lmstudio.ai/>), which is an app you can download.

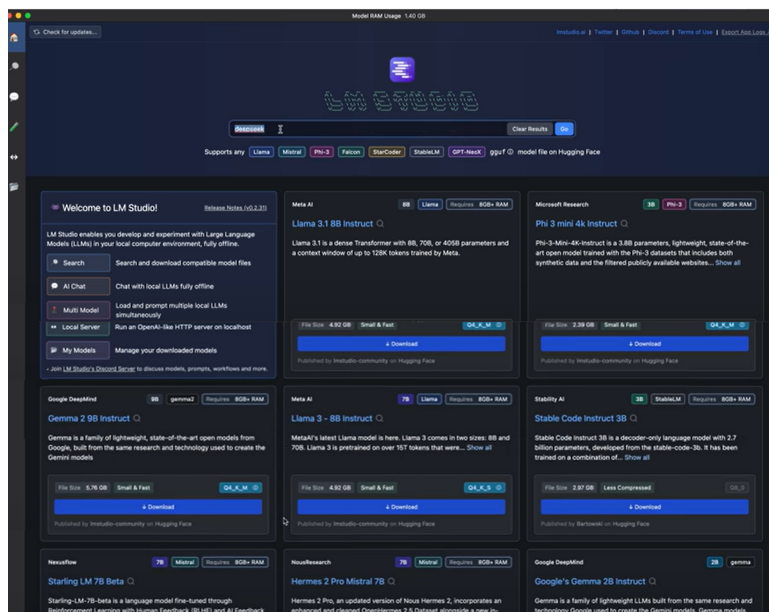


Figure 122: LM Studio.

I find the interface ugly, and I don't like that it displays so many models that aren't particularly useful — since, in truth, everyone just wants to run DeepSeek.

I don't know why they give you these 500 different types of models — they're really complicated to search for, and you have to choose between different distilled versions and different precisions, which is all very confusing. But once you actually understand how it works (which would need a whole separate video), you can load up a model — for example, here I loaded Llama 3 Instruct 1 Billion — and you can just have a conversation with it.

I asked for a pelican joke:

USER: Write a joke about a pelican

AI: Here's one:

Why did the pelican go to the doctor?

Because it had a fowl cough! (get it?)

I asked for another one:

USER: Another one

AI: Why did the pelican go to therapy?

Because it was having a beak-full of issues.

And it gave me another one etc.

To do this, you first need to download and install LM Studio and the specific model you want to use. After that, all processing happens locally on your computer; nothing is sent elsewhere — everything runs directly on your MacBook Pro's GPU.

That's very convenient, and when you're done, you can eject the model to free up RAM. So LM Studio is probably my favorite tool, even though I think it has quite a few User Interface issues and User Experience issues and seems geared more toward professionals. Still, if you watch a few tutorials on YouTube, you can figure out how to use its interface. Those are just a few notes on where to find and use these models.

03:21:46

25 Grand summary

Let me loop back to where we began. The question was: when we go to ChatGPT, enter a query, and hit “Go,” what exactly happens? What are we seeing and interacting with? How does it all work? I hope this explanation has given you a sense of what’s happening under the hood — how these models are trained and what’s actually returned to us.

When you type your query, it’s first broken down into tokens. This tokenization step transforms the input into smaller pieces — subword units that the model can process. Your prompt fits into a special conversation format that structures dialog between the user and the assistant. The model receives this formatted prompt as a one-dimensional sequence of tokens. When you press “Go,” the model begins generating output by continuing that sequence—predicting the next token, one after another, like an intelligent autocomplete system.

Now, why does the model choose these particular tokens, and where do they come from? That takes us into how these systems are built. The process consists of three main stages. The first, pre-training, is devoted to knowledge acquisition—absorbing information from a large portion of the internet into the model’s parameters. The second stage, supervised fine-tuning, is where personality and behavior emerge. Companies like OpenAI create massive datasets of dialogues between a human and an assistant [they are entirely created by humans – the human sentences and the assistant sentences]. And human labelers craft ideal responses from the assistant, following precise guidelines. The model learns from these examples how to respond appropriately to any prompt.

So what’s really happening when you get a reply? Think of it as a simulation of one of those OpenAI labelers, trained to write the best possible response. The model doesn’t think like a human — it’s performing a constrained mathematical

computation to predict tokens. Each token generation is a finite procedure, *a lossy simulation of reasoning, not genuine cognition*. Because of that limitation, models sometimes make errors or “hallucinate.” Their abilities resemble Swiss cheese — impressive, but with holes that cause unexpected mistakes.

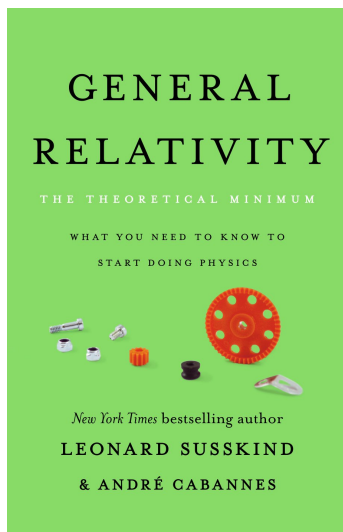
Despite those flaws, recent “thinking models” such as o1-mini or o3-mini [of OpenAI] go a step further. Unlike GPT-4 Turbo, which mainly uses supervised fine-tuning (SFT), these newer models also undergo reinforcement learning (RLHF). This third training stage helps them develop reasoning strategies — patterns that resemble step-by-step internal monologue. They practice solving many problems curated by human researchers, evolving novel reasoning techniques. When you ask such a model a question, you’re not just interacting with a simulated human annotator; you’re seeing early signs of genuine reasoning, an emergent ability born from this reinforcement process.

That said, reinforcement learning is still primitive. We’re seeing its early, experimental form. These models can show flashes of insight — doing something creative or reasoning in ways no human has taught them — but it’s just a beginning. Their strengths shine in verifiable domains like math and code, while their transfer to open-ended tasks such as creative writing remains uncertain. Researchers still don’t fully know how far those reasoning skills can generalize to areas where correctness isn’t measurable.

Overall, it’s an extraordinarily exciting time for AI. Personally, I use these models every day, often dozens or even hundreds of times, because they dramatically accelerate my work. Used wisely — as tools rather than authorities — they can be transformative. Always verify their outputs, since even the best can make random errors, miscount, or hallucinate facts. But if you leverage them thoughtfully — for drafting, exploring ideas, or refining your work — these systems can make you far more productive.

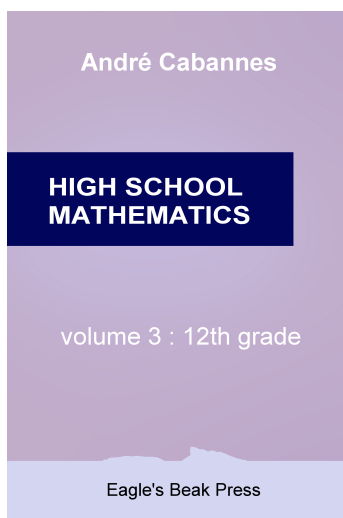
I hope that this video was useful and interesting to you, and that you had fun watching it.

Eagle's Beak Press catalog



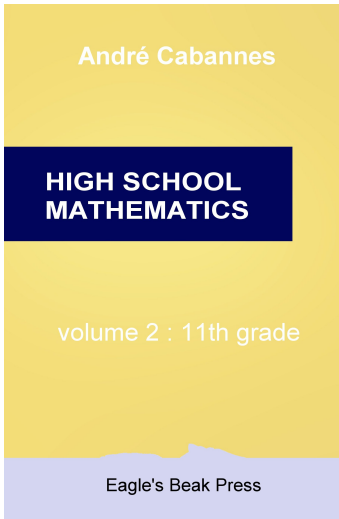
www.amazon.com/dp/B09ZB613QY
General Relativity

Graduate studies.



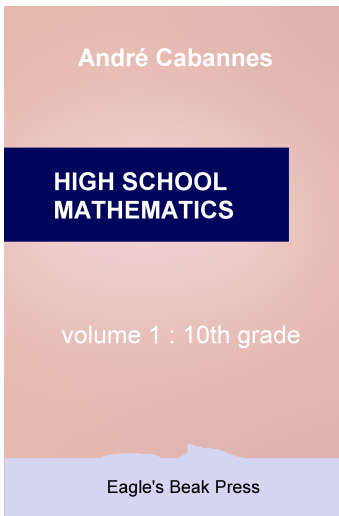
www.amazon.com/dp/2958738515
High school mathematics

Volume 3 : 12th grade



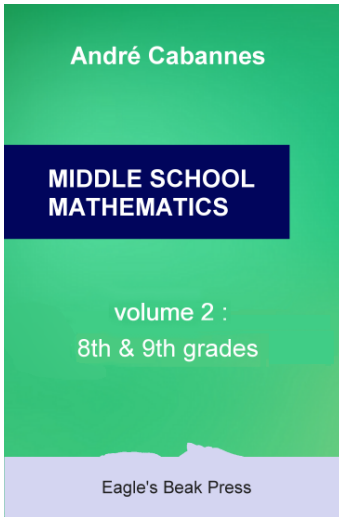
www.amazon.com/dp/2958738523
High school mathematics

Volume 2 : 11th grade



www.amazon.com/dp/2958738531
High school mathematics

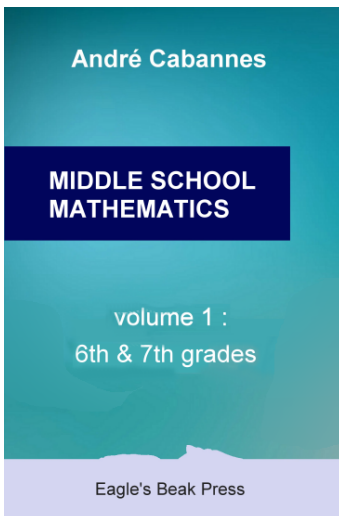
Volume 1 : 10th grade



www.amazon.com/dp/295873854X
Middle school mathematics

Volume 2 : 8th & 9th grades

for middle school students and
their parents



www.amazon.fr/dp/2958738558
Middle school mathematics

Volume 1 : 6th & 7th grades

for middle school students and
their parents