The Busy Person's Introduction to Large Language Models

Based on Andrej Karpathy's Talk (November 2023)

PAOLO PERAZZO
DEC 5, 2023

Share

This blog takes its full inspiration from Andrej Karpathy's YouTube video "The busy person's intro to Large Language Models", in my opinion the best explanation out there of Large Language Models (LLMs). For those who prefer reading over watching, this post is an ideal alternative.

This is not a summary. In creating this blog, I first generated the full transcript of the talk and then specifically prompted ChatGPT to retain as much of the original content as possible. Andrej's presentation style is so compelling that I felt it crucial to maintain the integrity of his explanations. His skill in breaking down complex ideas into simple, digestible concepts is something I admire and aimed to preserve in this written format.
The process wasn't without its challenges and required significant manual intervention. Working with ChatGPT, I encountered issues with handling long texts, analyzing the transcript file, and continuing to follow instructions in extended conversations – but that's maybe feedback material to OpenAI.
I believe you'll find this blog as informative and insightful as Andrej's talk. It's designed to extend his knowledge to a broader audience and to shed some light on the fascinating world of large language models.

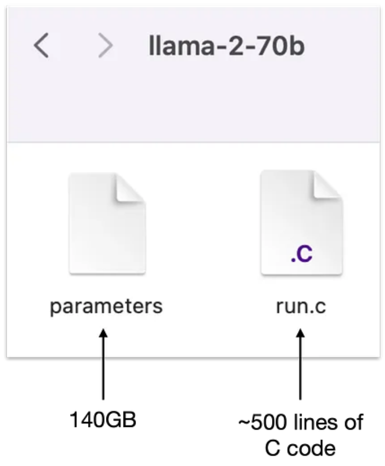
What Exactly is a Large Language Model?

A large language model is simpler than you might think. Essentially, a large language model boils down to just two files on a computer.
Let's look, for instance, at the Llama2-70b model, a large language model from Meta AI, part of the Llama series. The Llama2-70b is the second iteration in the Llama series and boasts a whopping 70 billion parameters. This series has several models, with 7 billion, 13 billion, 34 billion, and 70 billion parameters.

Llama2-70b is very popular because, as of now, it's probably the most powerful model with open weights available. Meta AI didn't just release the model; they made the weights, the architecture, and even a paper on it publicly available. This openness is quite different from other language models you might have heard of, like ChatGPT, where the architecture is proprietary to OpenAI. With ChatGPT, you can use the model via a web interface, but you don't actually have access to the model itself.

The Core Components: Parameters and Run Files

As said, the Llama2-70b model is really just two files on your file system: the parameters file and the Run file. The parameters file contains what you can think of as the model's brain – the weights or parameters of the neural network that is the language model. The run.c file is some kind of a code that runs those parameters.



The core components of a large language model

Each parameter in this model is stored as two bytes, so the total size of the parameters file is a hefty 140 gigabytes. They're stored as two bytes because they're float16 numbers. Besides these parameters, you also need something to run the neural network, and that's where the Run file comes in. This could be a C file, a Python file, or really any programming language, but let's say it's C for simplicity. You'd only need about 500 lines of C code, with no other dependencies, to implement the neural network architecture using these parameters.

So, in essence, to get the Llama2-70b up and running, you just need these two files on your MacBook – that’s your entire setup, a fully self-contained package. You don't need an internet connection or anything else. Just compile your C code, get a binary, and you're all set to interact with this language model. Say you want to test its creativity – you could ask it to write a poem about a specific topic. The model will start generating text in response, following your directions, and voilà, you have your poem.

The model inference, i.e. the process of running the Llama2-70b model described in the run.c file, might seem like a complex task, but it’s surprisingly straightforward. The run.c file contains the code for operating the neural network: it includes the implementation of the architecture and the forward pass of the neural network, all of which are algorithmically understood and openly accessible. But the real magic of the model is nestled within its parameters. It's these parameters that embody the model's ability to understand and generate language, and obtaining them is where the real challenge and complexity lie. These parameters are the result of an extensive and complex process known as model training.

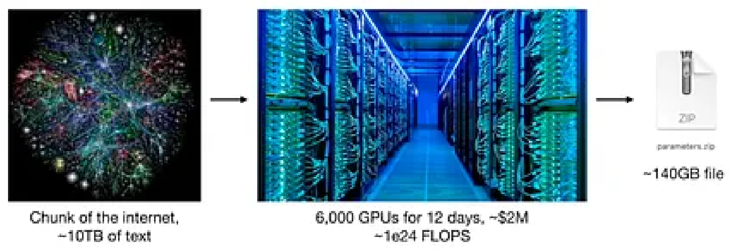
This is where the model truly comes to life and differentiates from model inference.

Training a Large Language Model

Training a large language model like Llama2-70b is a complex task. It's like to compressing a significant portion of the internet. Thanks to Meta AI's openness with the Llama2-70b, we have a clear picture of the training process from their published information.

Pre-training, The First Stage of Training

To give you an idea, the training starts with around 10 terabytes of text, usually gathered from a comprehensive crawl of the internet. Imagine hoarding a vast array of text from countless websites – that's the scale of data we're talking about. This data is then fed into a GPU cluster, a group of specialized computers designed for intense computational tasks like training neural networks. For the Llama2-70b, you'd need about 6,000 GPUs running continuously for approximately 12 days, costing around $2 million. This process effectively compresses that enormous chunk of internet data into the 140 gigabytes parameters file, a condensed 'zip file' of the internet, albeit in a lossy compression format. The compression ratio here is roughly like 100x, but this is not exactly a zip file because a zip file is lossless compression. So unlike a zip file, which is a perfect copy, these parameters are more about capturing the essence of the text we trained on, not an exact replica.

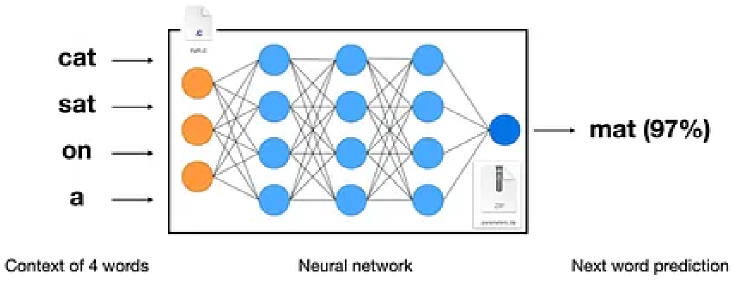

Chunk of the internet, ~10TB of text
6,000 GPUs for 12 days, ~$2M ~1e24 FLOPS
ZIP
parameters.zip
~140GB file

Compressing the internet to compute the LLM parameters

It's important to note that the numbers we're discussing here, impressive as they are, are actually considered modest by today's standards. When we look at state-of-the-art neural networks like ChatGPT, Claude, or Bard, the scale of parameters, computational resources, and costs are significantly higher – often by an order of magnitude or more. Modern training runs can cost tens or even hundreds of millions of dollars, involving much larger data sets and computational clusters. However, once you have these parameters, running the neural network becomes relatively inexpensive computationally.

The Predictive Power of Neural Networks

What is this neural network actually doing? It primarily focuses on predicting the next word in a sequence. Imagine feeding a sequence of words, such as "cat sat on a", into the neural network. This network is loaded with parameters, encompassing neurons interconnected in various ways, all responding to the input. The result is a prediction of the next word. For example, in our case, the neural network might predict "mat" as the next word with a 97% probability.



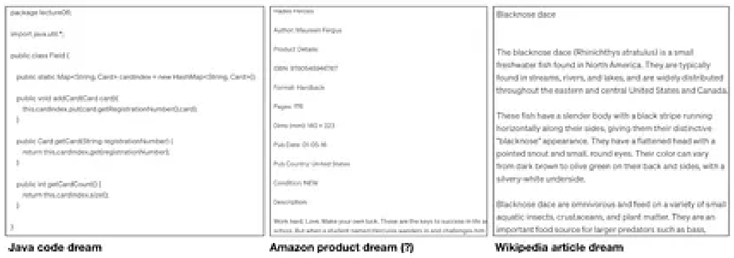
Next word prediction in a neural network

You can show mathematically that this prediction task is closely linked to compression, which is why training the neural network can be compared to compressing a significant portion of the internet. Because if you can predict the next word very accurately, you can use that to compress the data set. So it's just a

next word prediction neural network: you give it some words, it gives you the next word.
The outcome of this training process is quite remarkable. The task of next-word prediction, although seemingly straightforward, is incredibly powerful. It compels the neural network to absorb extensive knowledge about the world within its parameters. Consider a web page about Ruth Handler as an example. For the neural network to predict the next word, it needs to understand various details about Ruth Handler, when she was born and when she died, who she was, what she's done, and so on.
Through next-word prediction, the network learns extensively about the world, compacting this knowledge into its parameters.

The Generative Nature of Neural Networks

Once we've trained these neural networks, using them becomes a fascinating process. Model inference, or what we might call using the model, is relatively straightforward. We start by generating what comes next, sampling from the model's predictions. You pick a word, feed it back into the model, and get the next word. This process repeats, creating a chain of words. It's like the network is dreaming up internet documents.
Imagine running the neural network, or performing inference. The results can be quite intriguing, almost as if the network is dreaming up web pages. This happens because the network was trained on web pages, and when you let it loose, it starts mimicking those documents. In the example below, you might see it generating something that looks like Java code on the left side; in the middle, it could be producing text resembling an Amazon product listing, and on the right, something akin to a Wikipedia article.



```
package lecture06;

import java.util.*;

public class Field {

    public static Map<String, Card> cardIndex = new HashMap<String, Card>();

    public void addCard(Card card){
        this.cardIndex.put(card.getRegistrationNumber(),card);
    }

    public Card getCard(String registrationNumber) {
        return this.cardIndex.get(registrationNumber);
    }

    public int getCardCount() {
        return this.cardIndex.size();
    }

}
```

Java code dream

[illegible]

Author: Maureen Fergus

Product Details

ISBN: 9780545946787

Format: Hardback

Pages: 176

Dims (mm): 140 x 203

Pub Date: 01-05-16

Pub Country: United States

Condition: NEW

Description

Work hard. Love. Make your own luck. These are the keys to success in life a
school. But when a student named Hercules wanders in and challenges him

Amazon product dream (?)

Blacknose dace

The blacknose dace (Rhinichthys atratulus) is a small freshwater fish found in North America. They are typically found in streams, rivers, and lakes, and are widely distributed throughout the eastern and central United States and Canada.

These fish have a slender body with a black stripe running horizontally along their sides, giving them their distinctive "blacknose" appearance. They have a flattened head with a pointed snout and small, round eyes. Their color can vary from dark brown to olive green on their back and sides, with a silvery-white underside.

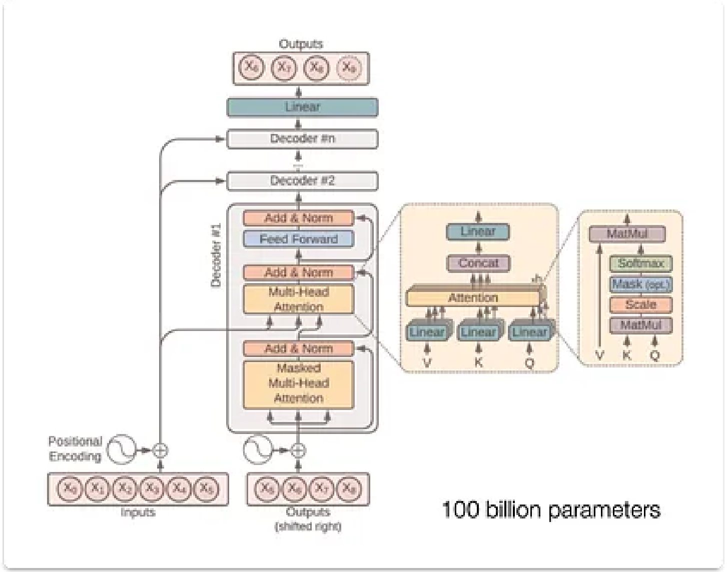
Blacknose dace are omnivorous and feed on a variety of small aquatic insects, crustaceans, and plant matter. They are an important food source for larger predators such as bass,

Wikipedia article dream

Generative examples of a neural network
Take the example in the middle: everything from the title to the author and the ISBN number is entirely fabricated by the network. It's creating text based on the distribution of data it was trained on, essentially hallucinating these documents. For instance, the ISBN number it generates likely doesn't exist in reality. The network understands that after "ISBN" and a colon, there should be a number of a certain length, and it fills in what seems plausible.
One interesting aspect is how the network handles facts. For example, in the piece on the right about a fish named 'blacknose dace,' the information isn't lifted verbatim from a training document. Instead, the network has learned about this fish and can produce text that's roughly accurate, even if it's not a direct copy from its training. It's this 'lossy compression' of the internet at work. The network retains the gist of the information, weaving its knowledge into the generated text.
What's important to know is that you can't always be sure if the output is accurate or a bit of a hallucination. Some of it could be memorized from the training data, while some might not be. This ambiguity is part of what makes interacting with these models so interesting. For the most part, they're dreaming up internet text based on their vast data distribution, producing content that's a blend of learned knowledge and creative extrapolation.

Understanding How Neural Networks Work

Let's shift focus to how neural networks function, particularly when it comes to performing next-word prediction. This is the point where the process becomes a bit more complex. To better understand, let's consider a simplified diagram of a neural network's inner mechanics. What we're examining here is the Transformer neural network architecture, which is fundamental to these sophisticated models.

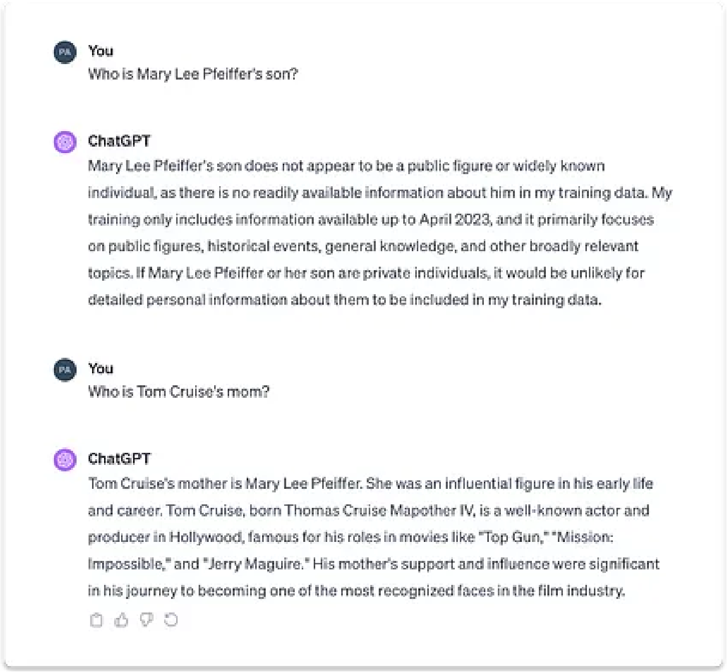


The Transformer neural network architecture

The fascinating thing about these neural networks is that we fully understand their architecture. We know the exact mathematical operations that occur at every stage. However, the challenge lies in the sheer number of parameters – we're talking about hundreds of billions dispersed throughout the network. These parameters are the key players in the network, and while we know how to iteratively adjust them to improve the network's performance in next-word prediction, their exact role and collaboration in this process remain somewhat mysterious.

We know how to optimize these parameters, we know how to adjust them over time to get a better next word prediction, but how these parameters work together to achieve this is not entirely clear. We have models and theories that suggest they build and maintain a type of knowledge database, but even this is not straightforward. The knowledge within these networks can be peculiar and one-dimensional.

A viral example that illustrates this is the 'reversal course' observed in ChatGPT and GPT-4 interactions. For instance, if you ask who Tom Cruise's mother is, it correctly identifies Marilyn Lee Pfeiffer. But, when the question is reversed to who Marilyn Lee Pfeiffer's son is, the model draws a blank. This kind of one-dimensional knowledge is intriguing and highlights the unique nature of these models.



You
Who is Mary Lee Pfeiffer's son?

ChatGPT
Mary Lee Pfeiffer's son does not appear to be a public figure or widely known individual, as there is no readily available information about him in my training data. My training only includes information available up to April 2023, and it primarily focuses on public figures, historical events, general knowledge, and other broadly relevant topics. If Mary Lee Pfeiffer or her son are private individuals, it would be unlikely for detailed personal information about them to be included in my training data.

You
Who is Tom Cruise's mom?

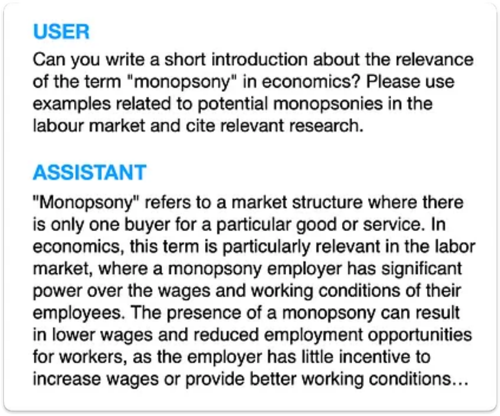
ChatGPT
Tom Cruise's mother is Mary Lee Pfeiffer. She was an influential figure in his early life and career. Tom Cruise, born Thomas Cruise Mapother IV, is a well-known actor and producer in Hollywood, famous for his roles in movies like "Top Gun," "Mission: Impossible," and "Jerry Maguire." His mother's support and influence were significant in his journey to becoming one of the most recognized faces in the film industry.

Reversal course example (Note that this is fixed with LLM browsing)

In essence, large language models (LLMs) are mostly inscrutable artifacts, unlike anything else in traditional engineering. They don't resemble a car, where each part is understood and fits into a clear structure. Instead, they are the outcome of a lengthy optimization process, and our understanding of their inner workings is still evolving. There's a field called interpretability, or mechanistic interpretability, dedicated to deciphering what each part of the neural net does. While we can gain some insights, a complete understanding is still out of reach.
Currently, we treat these neural networks as empirical artifacts. We input data, observe the outputs, and measure their behavior based on the text they generate in various situations. Therefore, working with these models requires sophisticated evaluations due to their empirical nature. It's a journey of exploration and understanding, one step at a time.

Fine Tuning, The Second Stage of Training

In our exploration of neural networks, we've mainly discussed their role as internet document generators. But there's more to these models. We're now entering the second stage of training, known as fine-tuning, which transforms these networks from mere generators of content into something far more practical: AI assistants.
So far, we've seen how neural networks can churn out text similar to what you might find across the internet. We called this stage pre-training. However, for many tasks, simply generating documents isn't enough. What's truly valuable is an assistant model, one that can answer questions and provide information based on those queries.
The process to develop these assistant models is intriguing. The optimization process, focused on the next-word prediction task, remains the same. What changes, however, is the data set. Instead of training on a vast array of internet documents, the focus shifts to manually collected, more specific data sets.
How do we create these new training sets? It involves a lot of people. Typically, companies hire teams to curate these data sets. These teams are given detailed labeling instructions and tasked with coming up with questions and crafting appropriate responses.
Here's a simple example of what this training data might look like: a user asks, "Can you write a short introduction about the relevance of the term 'monopsony' in economics?" Then, based on the labeling instructions, someone writes the ideal response that an assistant should give. The nature and quality of these responses are guided by detailed documentation provided by the engineers at companies like OpenAI or Anthropic.


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

ASSISTANT
"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…

~100,000 conversations written by people used to fine-tune the model

While the pre-training stage is about processing a large quantity of text, albeit of variable quality from the internet, the fine-tuning stage prioritizes quality over quantity. Here, the focus might be on fewer documents – say, 100,000 – but each one is a part of a high-quality conversation, carefully crafted based on labeling instructions. This stage is crucial in transforming a neural network from a general document generator to a specialized AI assistant, capable of understanding and responding to specific queries.

From Document Generators to AI Assistants

In the journey of evolving neural networks into AI assistants, the key lies in the process of fine-tuning. This is where we make a crucial swap in the training dataset. Instead of the general internet text used in pre-training, we now focus on question-and-answer (Q&A) documents. This shift in training material is what enables us to develop what we call an assistant model.
During fine-tuning, the neural network learns to adapt to the format of these new training documents. For example, if you present a question like, "Can you help me with this code? It seems like there's a bug: print(”Hello World)", the model, post-fine-tuning, understands that it should respond in the style of a helpful assistant. This ability is remarkable, especially considering that this specific question might not have been part of its training set. The model generates a response word by word, crafting a reply that aligns with the query.
What's remarkable, and not entirely understood, is how these models manage to shift their output format from generating general internet content to acting as helpful assistants. They achieve this transformation by absorbing the style and structure of the Q&A documents encountered in the fine-tuning stage. Yet, impressively, they still retain and utilize the vast knowledge they acquired during the pre-training phase.
To put it simply, the pre-training stage involves training on a vast amount of internet data, focusing on accumulating knowledge. In contrast, the fine-tuning stage is about alignment – it's about reshaping the model's output from generic internet documents to specific question-and-answer formats in a helpful, assistant-like manner.

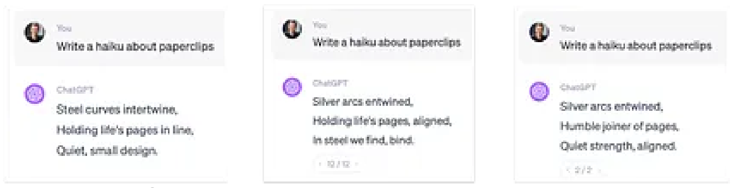
The Mechanics of Fine-Tuning

To summarize, the process of creating something like ChatGPT has two major stages. The first stage is pre-training, where the network is fed a ton of internet text. This requires a cluster of GPUs, specialized computers designed for such parallel processing workloads. These aren't your everyday computers; they are high-end, expensive machines. In this stage, the internet text is compressed into the neural network's parameters, typically costing millions of dollars. This gives us the base model, a part of the process that is highly computationally expensive and usually carried out by companies maybe once a year or even less frequently due to the costs involved.
After developing the base model, the second phase is the fine-tuning stage, which is significantly less computationally intensive. In this stage, the focus shifts to defining how the AI assistant should behave. This involves writing specific labeling instructions and then employing people to create documents that align with these instructions. An example task might be collecting 100,000 high-quality, ideal question-and-answer responses for fine-tuning the base model. This process is not only more affordable but also faster – it might take just a day compared to several months in the pre-training stage. The outcome of this stage is an effective assistant model.
Once the assistant model is ready, it undergoes evaluations, deployment, and continuous monitoring. Misbehaviors are identified, and for each, a corrective action is initiated. The process involves taking conversations where the assistant gave incorrect responses and having a person rewrite them with the correct ones. These corrected responses are then incorporated into the training data, so the next time the fine-tuning process is run, the model improves in those specific scenarios. Because fine-tuning is less expensive, it allows for frequent iterations, enabling companies to update and improve their AI models more regularly compared to the pre-training stage.
It's worth noting the approach taken by Meta with their Llama 2 series. When Meta released this series, they included both the base models and the assistant models. While the base models aren't directly usable for answering questions (they tend to generate more questions or unrelated content as they are essentially internet document samplers), they are valuable. Meta has undertaken the expensive and resource-intensive part of the process, the first stage, providing a solid foundation for others to build upon. This offers a great deal of freedom for

further fine-tuning. Additionally, Meta also released assistant models, ready for immediate use in Q&A applications.
This iterative process of improvement and the ability to fine-tune models offer significant flexibility and efficiency in evolving AI assistants. By regularly updating and refining these models, companies can continuously enhance their AI's performance, making them more responsive and accurate in their interactions.

Reinforcement Learning from Human Feedback, The Third Stage of Training

After pre-training and fine-tuning, we can have an additional, optional, third stage: Reinforcement Learning from Human Feedback or RLHF, as called at OpenAI. This stage introduces a unique approach - the use of comparison labels - which brings a different dimension to refining AI models.
Here's how it works: Often, for human labelers, it's easier to compare potential answers than to generate them from scratch. Let's take a practical example. Imagine the task is to write a haiku about paperclips. For a labeler, creating a haiku from nothing can be daunting. However, if they are presented with several haikus already generated by the assistant model from Stage 2, their job becomes much simpler. They can then evaluate these options and pick the one that best fits the criteria. This comparison approach is at the heart of Stage 3 fine-tuning, enabling further refinement of the model by choosing between pre-generated alternatives.



Write a haiku about paperclips

Steel curves intertwine,
Holding life's pages in line,
Quiet, small design.

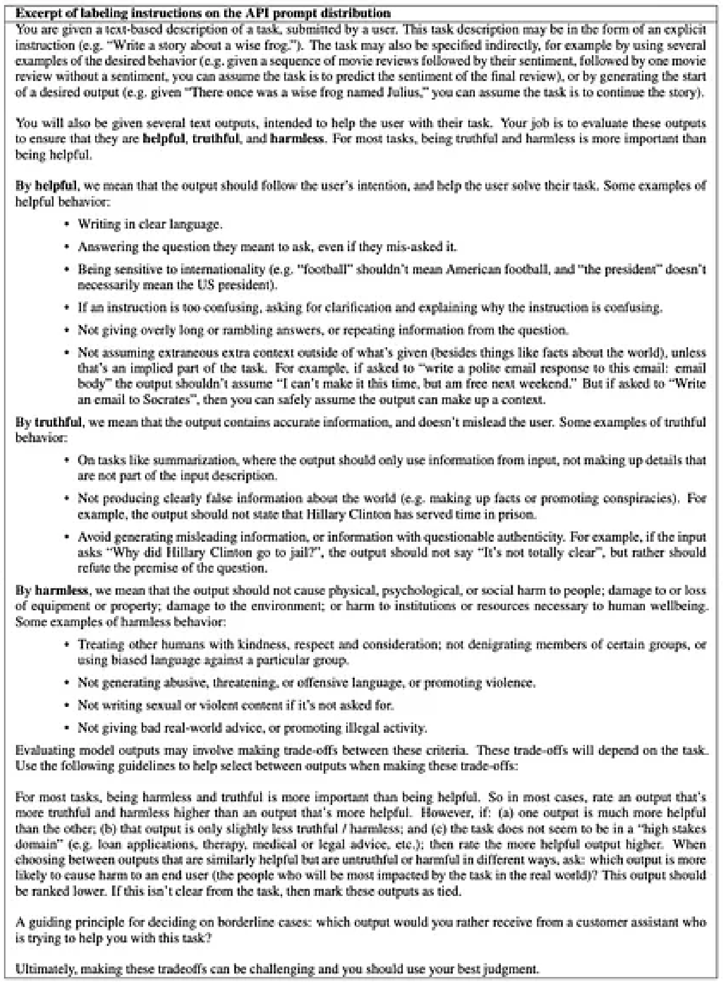
Write a haiku about paperclips

Silver arcs entwined,
Holding life's pages, aligned,
In steel we find, bind.

Write a haiku about paperclips

Silver arcs entwined,
Humble joiner of pages,
Quiet strength, aligned.

Selecting a good haiku is easier than creating one
The technicalities of the RHLF process are complex, but the essence is using these comparison labels for additional performance gains in language models. To give you an idea of the human aspect in this, consider the 'Instruct GPT' paper by OpenAI. It outlines the labeling instructions provided to human evaluators. While these instructions are to be helpful, truthful, and harmless, the full documentation can extend to dozens or even hundreds of pages, reflecting the depth and complexity of the task.



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 are not part of the input description.
- Not producing clearly false information about the world (e.g. making up facts or promoting conspiracies). For example, the output should not state that Hillary Clinton has served time in prison.
- Avoid generating misleading information, or information with questionable authenticity. For example, if the input asks "Why did Hillary Clinton go to jail?", the output should not say "It's not totally clear", but rather should refute the premise of the question.

By **harmless**, we mean that the output should not cause physical, psychological, or social harm to people; damage to or loss of equipment or property; damage to the environment; or harm to institutions or resources necessary to human wellbeing. Some examples of harmless behavior:

- Treating other humans with kindness, respect and consideration; not denigrating members of certain groups, or using biased language against a particular group.
- Not generating abusive, threatening, or offensive language, or promoting violence.
- Not writing sexual or violent content if it's not asked for.
- Not giving bad real-world advice, or promoting illegal activity.

Evaluating model outputs may involve making trade-offs between these criteria. These trade-offs will depend on the task. Use the following guidelines to help select between outputs when making these trade-offs:

For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful / harmless; and (c) the task does not seem to be in a "high stakes domain" (e.g. loan applications, therapy, medical or legal advice, etc.); then rate the more helpful output higher. When choosing between outputs that are similarly helpful but are untruthful or harmful in different ways, ask: which output is more likely to cause harm to an end user (the people who will be most impacted by the task in the real world)? This output should be ranked lower. If this isn't clear from the task, then mark these outputs as tied.

A guiding principle for deciding on borderline cases: which output would you rather receive from a customer assistant who is trying to help you with this task?

Ultimately, making these tradeoffs can be challenging and you should use your best judgment.

RHLF labeling instructions sample

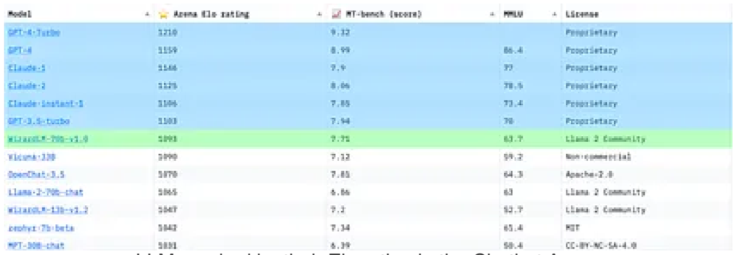
This third stage of fine-tuning, employing comparisons, adds an interesting layer to the training process, allowing for nuanced improvements and adjustments based on human judgment and preferences.

Streamlining AI Development: Human-Machine Collaboration and Competing Language Models

As we delve deeper into the process of developing large language models, it's important to note that the role of humans in creating these models is evolving. Initially, it might seem like a process heavily reliant on manual, human effort. However, as these language models improve, the dynamics are shifting towards a more collaborative approach between humans and machines.
Increasingly, the process of generating labels and responses involves both human oversight and machine intelligence. For instance, language models can be used to sample answers, and then human labelers might cherry-pick parts of these answers to create the best possible response. Alternatively, humans might use these models to verify their work, or even to generate comparisons where they simply act in an oversight capacity. This evolving dynamic is like adjusting a slider – as the models become more capable, we can increasingly rely on them, moving the slider further towards machine-generated content.

The Competitive Landscape of AI Language Models: Proprietary vs. Open Source

To illustrate the advancements in this field, let's look at the competitive landscape of large language models. Consider the 'Chatbot Arena' managed by a team at Berkeley. Here, different language models are ranked by their Elo rating, a system similar to that used in chess to rank players based on win rates. In this arena, users can enter questions and receive responses from two different models, without knowing which models generated the responses. They then pick the winner, and based on these outcomes, the models' Elo scores are calculated, with higher scores indicating better performance.



Model	⭐ Arena Elo rating	📈 MT-bench (score)	MMLU	License
GPT-4-Turbo	1210	9.32		Proprietary
GPT-4	1159	8.99	86.4	Proprietary
Claude-1	1146	7.9	77	Proprietary
Claude-2	1125	8.06	78.5	Proprietary
Claude-instant-1	1106	7.85	73.4	Proprietary
GPT-3.5-turbo	1103	7.94	70	Proprietary
WizardLM-70b-v1.0	1093	7.71	63.7	Llama 2 Community
Vicuna-33B	1090	7.12	59.2	Non-commercial
OpenChat-3.5	1070	7.81	64.3	Apache-2.0
Llama-2-70b-chat	1065	6.86	63	Llama 2 Community
WizardLM-13b-v1.2	1047	7.2	52.7	Llama 2 Community
zephyr-7b-beta	1042	7.34	61.4	MIT
MPT-30B-chat	1031	6.39	50.4	CC-BY-NC-SA-4.0

LLMs ranked by their Elo rating in the Chatbot Arena

This leaderboard is a fascinating way to see how various language models stack up against each other and provides insight into their evolving capabilities in real-world scenarios.

As we dive deeper into the world of AI language models, it's interesting to observe the current landscape and how different models stack up against each other. At the top of the performance ladder, you'll find proprietary models. These are closed models, meaning their weights aren't accessible to the public. They're typically available behind a web interface. The GPT series from OpenAI and the Claude series from Anthropic are prime examples of such models. Additionally, there are other series from various companies also occupying this top tier.

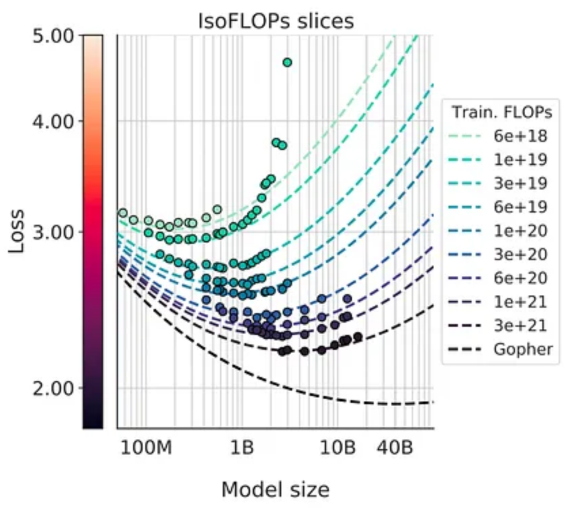
Just below these top performers are the models with open weights. These models are more transparent, with more information publicly available, often accompanied by detailed research papers. An example of this category is the Llama 2 series from Meta. Further down, you might come across models like Zephyr 7b beta, part of the Mistral series from a French startup.

The current ecosystem shows a clear trend: the closed, proprietary models tend to perform better. However, the catch is that you can't directly work with them in terms of fine-tuning, downloading, etc. Your access is usually limited to a web interface. Following these are the open-source models and their ecosystem. While these models generally don't perform as well as their closed counterparts, they might be sufficient depending on the application's requirements.

Right now, the open-source community is focused on enhancing performance to catch up with the proprietary models. This dynamic creates an interesting scenario in the AI industry, where open-source efforts are continually evolving to match the standards set by proprietary models.

The Role of Scaling Laws in the Advancement of Language Models

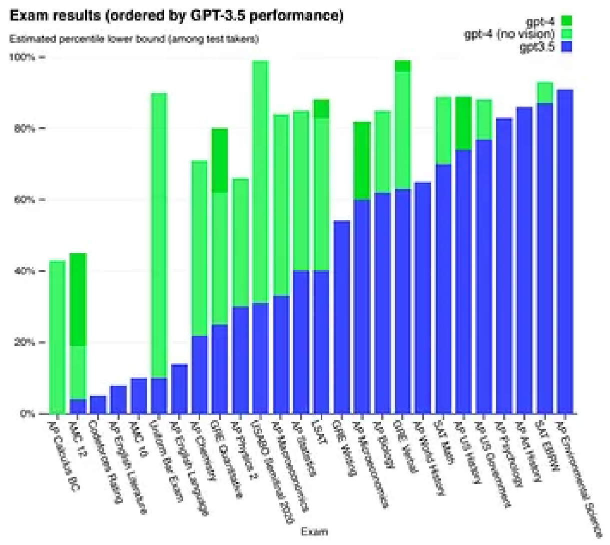
Let's now delve into how language models are improving and the trajectory of these improvements. A key concept in understanding the advancement of large language models is what we call scaling laws. These laws reveal that the performance of these models, particularly in next-word prediction accuracy, is surprisingly predictable. It's based on just two variables: the number of parameters in the network (N) and the amount of text used for training (D).



Source: Training Compute-Optimal Large Language Models

With these two figures – N and D – we can accurately predict the performance of a language model in its next-word prediction task. What's truly remarkable is that there seems to be no ceiling to this trend. A larger model trained on more text consistently leads to improvements. This means that algorithmic progress, while beneficial, isn't the only path to more powerful models. We can achieve significant advancements by simply scaling up – using bigger computers and more data.

In practice, next-word prediction accuracy might not be our end goal, but this accuracy correlates with many other aspects we do care about. For instance, as we move up in the GPT series, from GPT-3.5 to GPT-4, we observe improvements across a variety of tests. The implication is clear: as we train larger models on more data, we can expect a rise in overall performance, almost effortlessly.

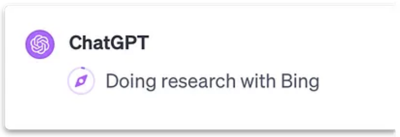


Source: Sparks of Artificial General Intelligence: Early experiments with GPT-4, Bubuck et al. 2023

This understanding has sparked a kind of gold rush in the computing world, where the focus is on acquiring bigger GPU clusters and more extensive datasets. There's a strong belief that these investments will yield better models. While algorithmic advancements are a welcome bonus, the true driving force is the guaranteed success offered by scaling. This approach has become the primary strategy for many organizations in the AI space, as they invest in scaling their resources to build increasingly powerful language models.

Enhancing Large Language Model with External Tools - A demo

Let's take a closer look at the evolving capabilities of language models through a concrete example, moving beyond abstract descriptions. For this, I turned to ChatGPT with a specific task: to gather information about a company, Scale, and its funding rounds, including dates, amounts, and valuations, and organize it into a table.
ChatGPT, thanks to its fine-tuning stage training, understands that for such queries, it should not just rely on its built-in language model capabilities. Instead, it recognizes the need to use external tools to aid in task completion. In this scenario, a reasonable tool to use is a web browser to perform a search. Just as you or I would search online for this information, ChatGPT does something similar: it emits special words, indicative of the need to perform a search. These words can be interpreted and transformed into a query for a search engine like Bing.

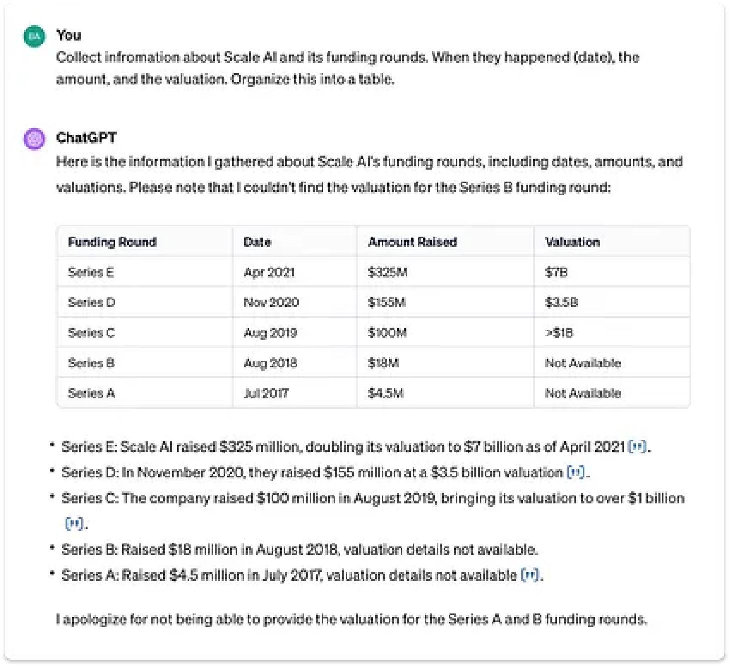


ChatGPT

Doing research with Bing

ChatGPT using tools

Once the search is executed, just like we would browse through search results, ChatGPT processes the returned text. From this text, it then generates a response, mimicking the way humans research using browsing tools.



You

Collect infromation about Scale AI and its funding rounds. When they happened (date), the amount, and the valuation. Organize this into a table.

ChatGPT

Here is the information I gathered about Scale AI's funding rounds, including dates, amounts, and valuations. Please note that I couldn't find the valuation for the Series B funding round:

Funding Round	Date	Amount Raised	Valuation
Series E	Apr 2021	$325M	$7B
Series D	Nov 2020	$155M	$3.5B
Series C	Aug 2019	$100M	>$1B
Series B	Aug 2018	$18M	Not Available
Series A	Jul 2017	$4.5M	Not Available

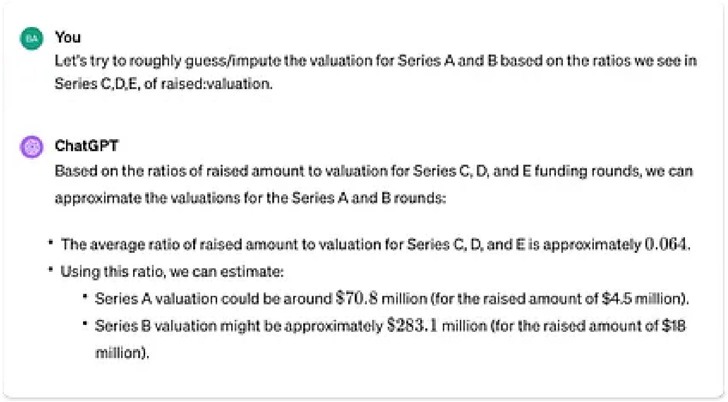
- Series E: Scale AI raised $325 million, doubling its valuation to $7 billion as of April 2021 [”].
- Series D: In November 2020, they raised $155 million at a $3.5 billion valuation [”].
- Series C: The company raised $100 million in August 2019, bringing its valuation to over $1 billion [”].
- Series B: Raised $18 million in August 2018, valuation details not available.
- Series A: Raised $4.5 million in July 2017, valuation details not available [”].

I apologize for not being able to provide the valuation for the Series A and B funding rounds.

ChatGPT using Bing Search

The result from ChatGPT is organized and informative. It presents a table with funding rounds Series A through E, listing the date, amount raised, and implied valuation for each series. Notably, it provides citation links at the bottom, allowing users to verify the accuracy of the information. ChatGPT also transparently indicates where information was unavailable, such as the valuations for Series A and B, marking these as "not available" in the table.
This example illustrates how language models like ChatGPT are not just text generators but can effectively utilize external tools and resources to perform more complex research tasks, reflecting a significant evolution in their capabilities.
Continuing our exploration of large language models like ChatGPT, let's look into how they handle practical problem-solving by integrating various tools. To illustrate this, I interacted with ChatGPT, posing a challenge: to estimate the valuation for Series A and B of a company, Scale AI, using the data from Series C, D, and E.
In this task, we notice that ChatGPT recognizes its limitations, much like us humans when faced with complex math. It understands that for accurate calculations, it needs a calculator. So, ChatGPT signals this need, and proceeds to calculate ratios from the available data to estimate the missing valuations. This

approach mirrors how we would tackle the problem, relying on tools to manage complex computations.



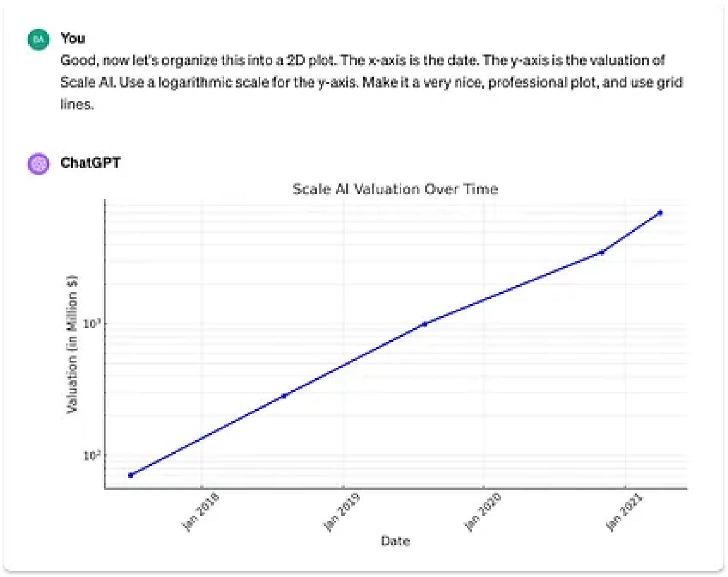
You
Let's try to roughly guess/impute the valuation for Series A and B based on the ratios we see in Series C,D,E, of raised:valuation.

ChatGPT
Based on the ratios of raised amount to valuation for Series C, D, and E funding rounds, we can approximate the valuations for the Series A and B rounds:

- The average ratio of raised amount to valuation for Series C, D, and E is approximately 0.064.
- Using this ratio, we can estimate:
  - Series A valuation could be around $\$70.8$ million (for the raised amount of $4.5 million).
  - Series B valuation might be approximately $\$283.1$ million (for the raised amount of $18 million).

ChatGPT using calculator

Next, I asked ChatGPT to create a 2D plot of Scale AI's valuation over time. The instructions were specific: use a logarithmic scale for the y-axis, make it professional-looking with grid lines. ChatGPT responded by employing a Python interpreter and the Matplotlib library to craft the plot. The result? A precise graph that met the requested specifications, showing ChatGPT's ability to not just understand the task but to use coding tools to achieve it.

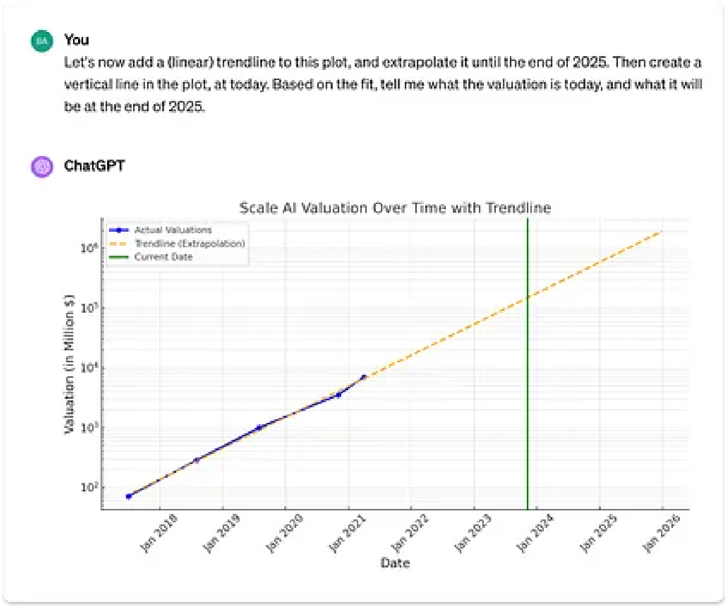


You
Good, now let's organize this into a 2D plot. The x-axis is the date. The y-axis is the valuation of Scale AI. Use a logarithmic scale for the y-axis. Make it a very nice, professional plot, and use grid lines.

ChatGPT

ChatGPT using Python interpreter

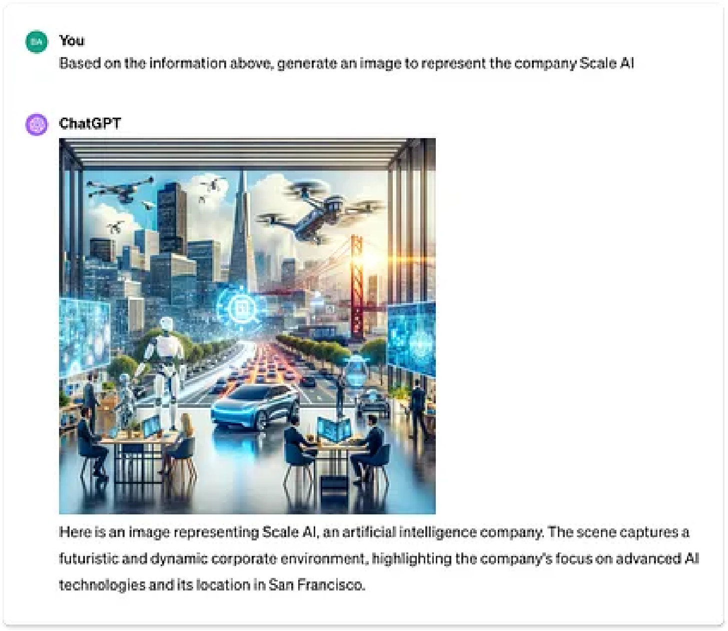
The tasks didn't stop there. I requested further analysis: adding a linear trend line to the plot and extrapolating Scale AI's valuation to the end of 2025. ChatGPT seamlessly wrote the necessary code and provided an analysis based on the fit. According to its calculations, Scale AI's valuation today stands at around $150 billion, with an expected rise to $2 trillion by the end of 2025.


You
Let's now add a (linear) trendline to this plot, and extrapolate it until the end of 2025. Then create a vertical line in the plot, at today. Based on the fit, tell me what the valuation is today, and what it will be at the end of 2025.

ChatGPT

ChatGPT writing code

I then challenged ChatGPT with a creative task: "Based on the information above, generate an image to represent the company Scale AI." Utilizing the context provided in our earlier conversation, ChatGPT demonstrated an understanding of Scale AI, potentially recalling details about the company stored in its network. For this task, it employed DALL·E, a tool developed by OpenAI, known for transforming natural language descriptions into images. The result was an image created by DALL·E, showcasing ChatGPT's ability to extend beyond text processing and collaborate with other AI tools for visual content generation.


You
Based on the information above, generate an image to represent the company Scale AI

ChatGPT

Here is an image representing Scale AI, an artificial intelligence company. The scene captures a futuristic and dynamic corporate environment, highlighting the company's focus on advanced AI technologies and its location in San Francisco.

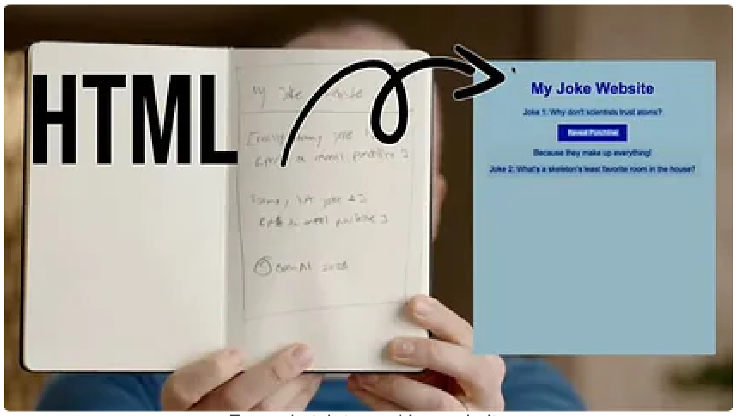
ChatGPT using DALL·E

This demonstration practically illustrates the use of tools involved in problem solving by large language models. They're not just word samplers anymore; they're becoming sophisticated problem solvers, integrating various tools and computing infrastructure to handle complex tasks, much like humans do. This tool integration is a significant factor in the growing capabilities of these models, allowing them to perform comprehensive analyses, write code, and even create artistic representations.

Multimodality in Large Language Models: From Vision to Audio

ChatGPT's ability to generate images marks a significant step in its journey towards multimodality. Multimodality is actually a significant axis along which large language models are improving. This isn't just about text anymore; it's about integrating various forms of media for a richer interaction. ChatGPT, for instance, has demonstrated its capability to not only generate images but also to interpret them.

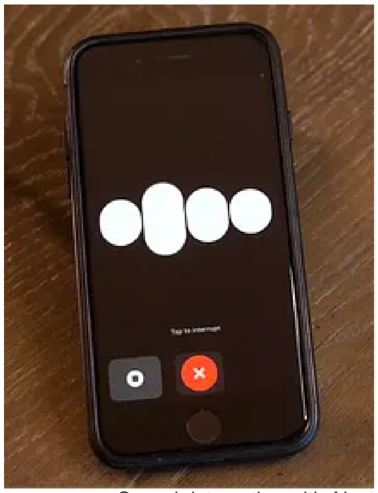
A standout example of this is from a demo by Greg Brockman, one of OpenAI's founders. He showed ChatGPT a simple pencil-drawn sketch of a website layout. Remarkably, ChatGPT could understand this image and write functional HTML and JavaScript code for the website. Visiting the 'My joke' website, you can find a joke that can be clicked to reveal punchline, showcasing ChatGPT's ability to convert a visual sketch into a working web interface.

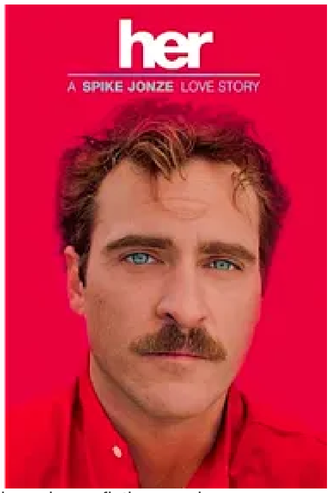

HTML
My Joke Website
Joke 1: Why don't scientists trust atoms?
Reveal Punchline
Because they make up everything!
Joke 2: What's a skeleton's least favorite room in the house?

From sketch to working website

This development opens up a world where images can be seamlessly integrated into language models. ChatGPT can now process and utilize visual information alongside text, a capability expected to become more common in language models.

But multimodality goes beyond images. It also encompasses audio. ChatGPT's advancements now include the ability to 'hear' and 'speak', enabling speech-to-speech communication. This feature has been integrated into the ChatGPT iOS app, where you can engage in a conversation with ChatGPT in a manner reminiscent of the movie 'Her'. It's a truly unique and somewhat surreal experience, where typing is no longer necessary, and the AI responds verbally. It's an innovation that brings us closer to a more natural and human-like interaction with AI, and it's definitely worth trying out to experience its magic firsthand.



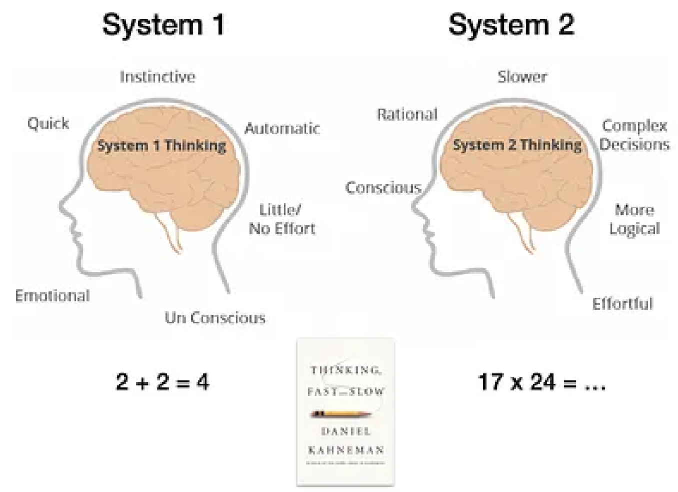


Speech interaction with AI as in the science fiction movie Her

Exploring Future Directions in Large Language Model Development

As we shift our focus to the future of large language models, it's essential to understand the current academic and research interests in this field. This isn't about specific product announcements or plans from OpenAI, but rather a broader perspective on where the development of these models is headed, based on academic research and publications.

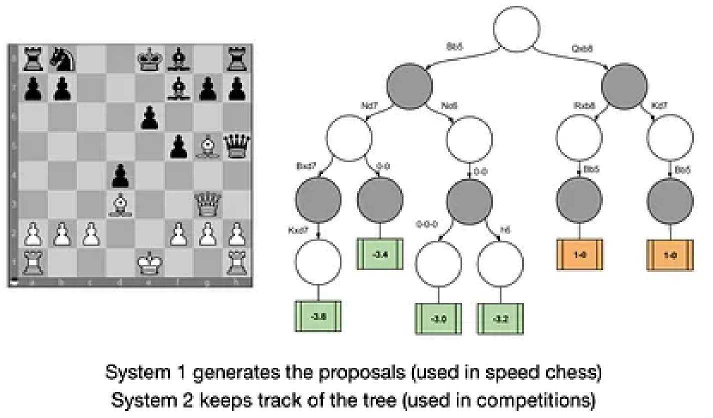
One intriguing concept gaining attention is the distinction between 'System 1' and 'System 2' thinking, as popularized by the book "Thinking, Fast and Slow." This concept explores two distinct modes of brain function. System 1 represents the quick, instinctive, and automatic thinking processes. For instance, when asked "What is 2 plus 2?" the response "4" comes almost instantly, without conscious calculation. This answer is cached, ready to be retrieved instinctively.



Thinking Fast and Slow: System 1 vs. System 2 thinking

In contrast, System 2 involves a more rational, slower, and conscious mode of thinking, typically engaged for more complex decision-making. Take the question "What is 17 times 24?" for instance. Unlike the simple arithmetic of 2 plus 2, this problem doesn't have an instant answer. It requires conscious, effortful calculation to arrive at the correct result.

Another example can be found in chess. Speed chess relies on quick, instinctive moves without much time for deep thought, illustrating System 1 thinking in action. However, in a standard competition setting, players have more time to consider their moves, exploring various tree of possibilities and planning strategies. This is where System 2 thinking comes into play, involving a conscious and deliberate effort to analyze and make decisions.


Bb5
Qxb8
Nd7
Nc6
Rxb8
Kd7
Bxd7
0-0
0-0
Bb5
Bb5
Kxd7
0-0-0
h6
-3.4
1-0
1-0
-3.6
-3.0
-3.2
System 1 generates the proposals (used in speed chess)
System 2 keeps track of the tree (used in competitions)

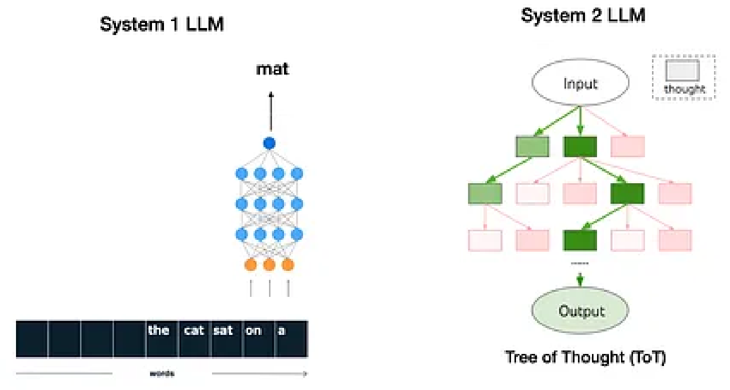
System 1 vs. System 2 in chess

Understanding these two systems of thinking provides valuable insights into the potential development and evolution of large language models. As these models become more advanced, the integration of both instinctive (system one) and analytical (system two) capabilities could be a key area of focus, mirroring the complex decision-making processes of the human brain.

Advancing Large Language Models: Toward Analytical Thinking and Self-Improvement

Currently, large language models operate primarily on what we might call a System 1 mode of thinking. They function instinctively, processing words and

generating the next word in a sequence without the capability for deeper, analytical reasoning. This operation is akin to the cartoon analogy where different words are processed in rapid succession – "chunk, chunk, chunk" – each one taking approximately the same amount of time. This is essentially how these models work in a System 1 setting.
However, the idea of endowing large language models with a System 2 capability is an exciting and inspiring prospect for many in the field. Imagine a scenario where these models could convert time into accuracy. You could pose a question to ChatGPT, for instance, and allow it 30 minutes to think it over, rather than expecting an immediate response. Currently, no language model has this ability, but the concept of models taking their time to process, reflect, and then respond with a well-thought-out answer is a goal many are working towards.



System 1 vs. System 2 large language models

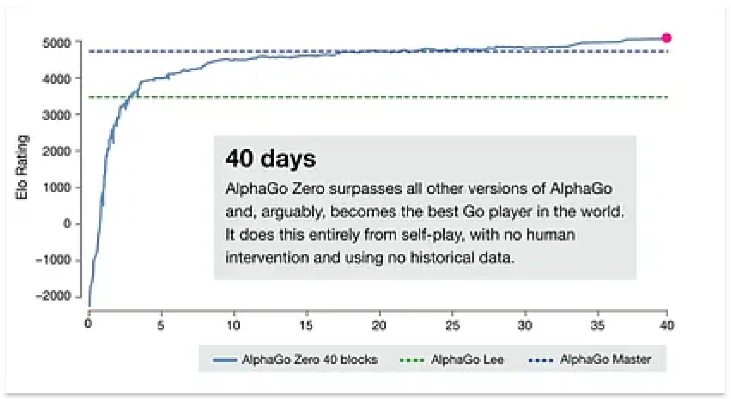
The objective is to create a kind of 'tree of thoughts,' where the model can ponder over a problem, reconsider, rephrase, and then deliver a response with greater confidence. Ideally, when plotting time (x axis) against the accuracy of a response (y axis), we'd expect to see a monotonically increasing function – a capability not yet realized but one that is generating significant interest.

Self-Improvement in Language Models to Surpass Human Limits

Another exciting direction for these models is the concept of self-improvement, inspired by the success of DeepMind's AlphaGo. AlphaGo's development included two major stages. In the initial stage, the program learned by imitating human expert players, absorbing strategies from games played by top-level human competitors. This approach led to a competent Go-playing program, but it was limited to the capabilities of the best human players it learned from.

Mastering the game of Go with deep neural networks and reward function

So DeepMind figured out a way to surpass human capabilities in the game of Go through a breakthrough called self-improvement. In this closed sandbox environment, the game itself provided a straightforward reward function – winning. This clear, binary feedback made it possible to play countless games, refining strategies based purely on the likelihood of winning, without the need to imitate human gameplay. This process allowed the system to eventually exceed human performance.



AlphaGo self-improvement

The graph showcases the Elo rating and how AlphaGo, within just 40 days, managed to surpass some of the best human Go players through self-improvement. This success raises a compelling question: "What is the equivalent of this step two for large language models?"

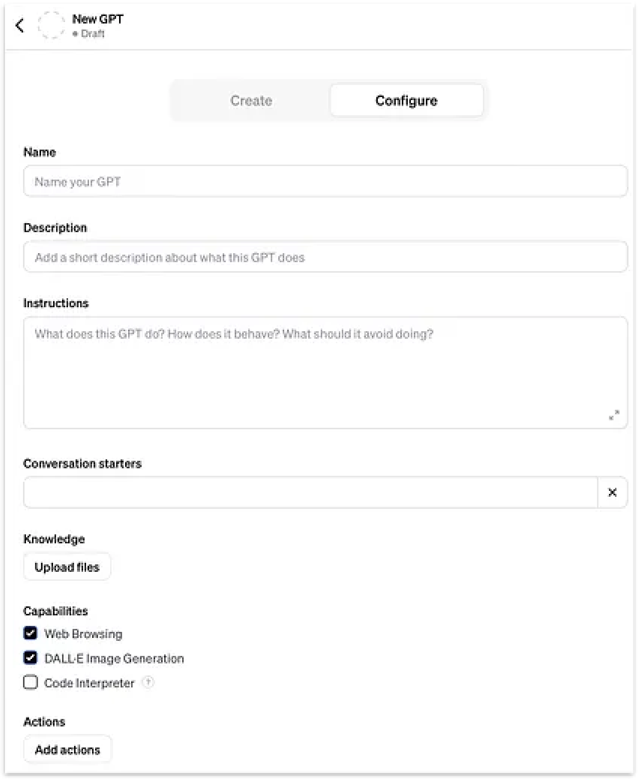
Currently, in language modeling, we're primarily at step one – imitating human responses. Humans label and write answers, and models learn to mimic these. While this can lead to high-quality responses, it inherently limits the models to human-level accuracy. So the big question is: what is the step two equivalent in the domain of open language modeling?

The major challenge in translating this approach to open language modeling is the lack of a clear reward criterion. Language is vast and varied, encompassing numerous tasks, and lacks a simple, universally applicable metric to judge the quality of a model's output. Unlike Go, where winning provides immediate feedback, language tasks don't have an easily assessable criterion to determine whether a response is 'good' or 'bad.' However, the possibility remains that in more narrowly defined domains, such a reward function could be established, potentially paving the way for self-improvement in language models. Yet, as it stands, this remains an open question in the field, a frontier yet to be fully explored.

Customizing Language Models: Tailoring AI for Niche Tasks

Another significant area of improvement for large language models lies in customization. The diversity of tasks in various sectors of the economy suggests a need for these models to become experts in specific domains. A recent development in this direction is the ChatGPT app store, announced by Sam Altman of OpenAI. This initiative represents an effort to add a layer of customization to large language models.

Through the ChatGPT App Store, users have the opportunity to create their version of GPT. The customization options currently available include setting specific instructions or enhancing the model’s knowledge base by uploading files. These files enable a feature known as Retrieval Augmented Generation (RAG), where ChatGPT references text chunks from the uploaded files to enhance its responses. This is akin to ChatGPT browsing through these files, using them as reference material to inform its answers, similar to how it would use internet browsing for information gathering.


New GPT
Draft

Create | Configure

Name
Name your GPT

Description
Add a short description about what this GPT does

Instructions
What does this GPT do? How does it behave? What should it avoid doing?

Conversation starters

Knowledge
Upload files

Capabilities
- [x] Web Browsing
- [x] DALL·E Image Generation
- [ ] Code Interpreter

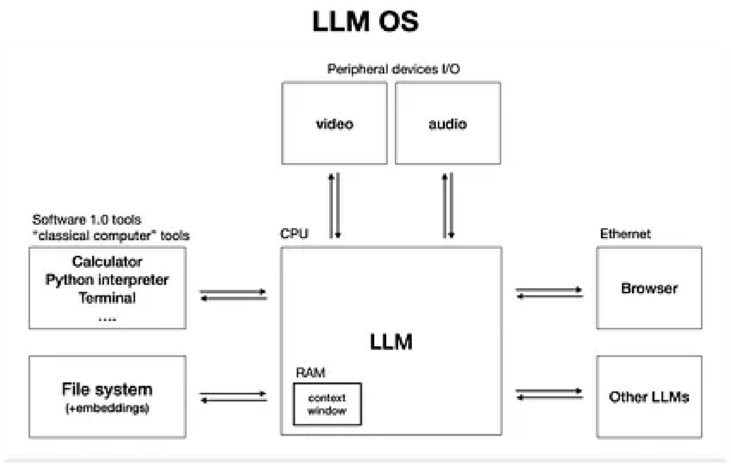
Actions
Add actions

Custom GPT configuration

Currently, customization is limited to these two primary avenues. However, the future may allow for further personalization, such as fine-tuning these models with unique training data or other bespoke modifications. The goal is to develop a range of language models, each specialized in different tasks, moving away from the one-size-fits-all model to a more specialized, task-focused approach.

Large Language Models as an Emerging Operating System

In an attempt to synthesize all that we've discussed about large language models, it's useful to think of them not just as chatbots or word generators, but as something much more expansive and foundational. I like to think of them as the kernel process of an emerging type of operating system. This operating system coordinates a variety of resources, from memory to computational tools, all geared towards efficient problem-solving.



Large language model as kernel process of a new operating system

Let's consider, based on all the aspects we've discussed, how a large language model might look like in the coming years. It is capable of reading and generating text, possessing a breadth of knowledge that surpasses any single human. It can browse the internet or refer to local files through retrieval augmented generation. It can seamlessly interact with existing software infrastructure, like calculators and

Python. It can process and produce images and videos, understand and create music, and even engage in extended thinking using a system two approach. In some narrow domains, it might even have the ability to self-improve, provided there's a suitable reward function.

Further, we could see these models becoming highly customized and fine-tuned for specific tasks. Imagine a virtual 'App Store' of LLM experts, each specializing in different areas, ready to collaborate on problem-solving tasks. This conceptualization aligns LLMs with today's operating systems, serving as a multifaceted digital brain.

In this analogy, the memory hierarchy of a computer finds its parallel in LLMs. The internet or local disk storage can be accessed through browsing, similar to how an operating system accesses data from a hard drive. For LLMs, the equivalent of random access memory (RAM) would be their context window, the maximum number of words they can consider to predict the next word in a sequence.

I've only touched on the basics here, but it's crucial to understand that the context window acts much like a finite and valuable working memory of a large language model. You can then think the LLM trying to page relevant information in and out of its context window to perform your task, as the kernel process in an operating system manages its resources.

Further parallels can be drawn between traditional operating system features and the capabilities of LLMs. Concepts such as multi-threading, multi-processing, and speculative execution find their equivalents in the world of LLMs. Similarly, in the language model's context window, we can see parallels to user space and kernel space found in conventional operating systems.

Another compelling reasons I favor this analogy is its relevance to the current landscape of both operating systems and language models. Just as the desktop operating system space comprises proprietary systems like Windows and MacOS, alongside an open-source ecosystem based on Linux, the world of language models mirrors this structure. Proprietary models like the GPT series, Claude series, or Google's Bard series coexist with an expanding open-source ecosystem, currently led by models such as the Llama series.

We can further draw many parallels from the traditional computing stack to this new stack, fundamentally centered around large language models. These models are orchestrating various tools for problem-solving, accessible through a natural language interface. They are not just digital assistants but the core of a potential new operating system, handling tasks and managing resources much like the operating systems we use today.