



# LA TECHNOLOGY TRANSFER presenta

## **Intelligenza Artificiale Generativa creare un Chatbot LLM per il Business**

di Ivan Reznikov

Online live streaming  
28-29 Aprile 2025

**Technology Transfer**

Piazza Cavour, 3 - 00193 Roma

**Tel.** 06-6832227 **Fax** 06-6871102

**e-mail:** [info@technologytransfer.it](mailto:info@technologytransfer.it) **Web:** [www.technologytransfer.it](http://www.technologytransfer.it)

**Tutti i diritti riservati.** E' vietata la riproduzione anche parziale di questo manuale e della documentazione associata, senza previa autorizzazione scritta dell'autore.

# Shameless Self-Promotion

- ✓PhD in Computational Sciences
- ✓12+ years of Python and Data Science experience
- ✓Worked for small/large companies, startups, government
- ✓Principal Data Scientist
- ✓Kaggle Competition Expert
- ✓TEDx Speaker (2017), GITEX/PyCON(2021, 2023),  
CodersHQ(2022, 2023), O'Reilly AI (2024)
- ✓Author of "LangChain for Life Sciences and Healthcare"

<https://www.linkedin.com/in/reznikovivan/>

<https://github.com/IvanReznikov>



# Shameless Self-Promotion (last slide, I promise)



LangChain @LangChainAI

...

If you're looking for an introduction to LangChain, we really like Ivan Reznikov's "LangChain 101 Course"

A fun weekend read if you've been meaning to get started!



O'REILLY®

## LangChain for Life Sciences and Healthcare

Innovation Through LLMs  
and Generative AI Agents



Ivan Reznikov

# Plan

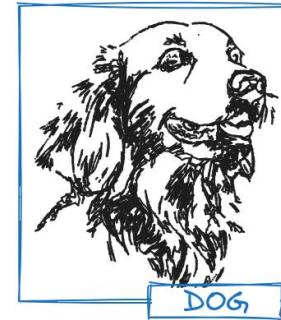
| Day 1   | Day 2                                |
|---|--------------------------------------|
| What is Generative AI?                            | Practice with No/Low Code frameworks |
| What are Large Language Models and how they work? | Finetuning Models                    |
| Practice with Tokenizers, LLMs, Embeddings        | Practice with Finetuning             |
| Basics of LangChain and RAG Pipelines             | Practice with Guardrails             |
| Practice with LangChain                           | Practice with Multi Agent            |

# **Generative AI Basics**

# Deep Learning Model Types



Classify



**Discriminative model:**  
classify as a dog or cat



Dog ...

Generate



**Generative model:**  
generate an image of a  
dog



# Deep Learning Model Types

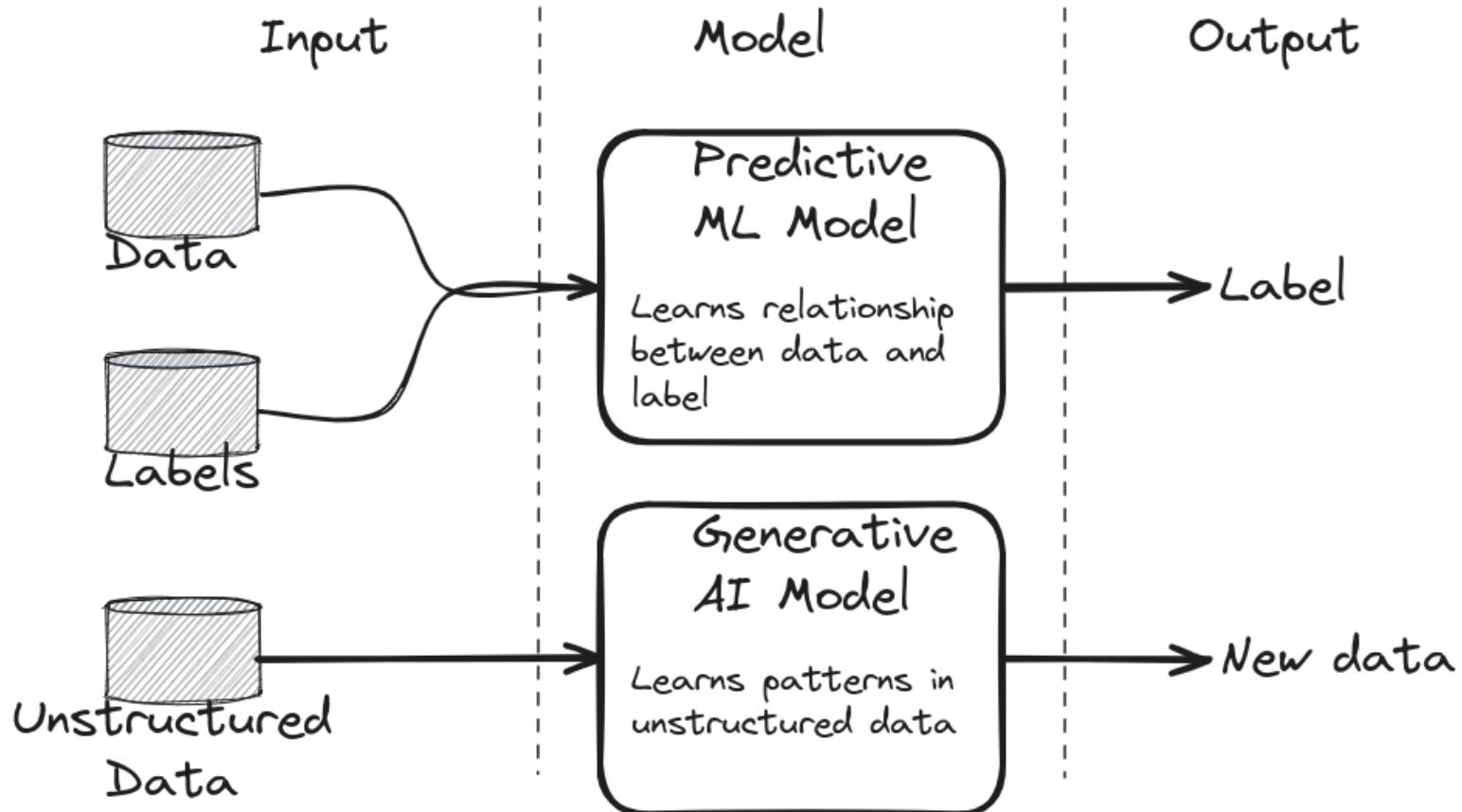
## Discriminative AI

- Used to classify or predict
- Typically trained on a dataset of labeled data
- Learns the relationship between the features of the data points and the labels

## Generative AI

- Generates new data that is similar to data it was trained on
- Understands distribution of data and how likely a given example is
- Predict next word in a sequence

# Deep Learning Model Types



# Applications?

Business / Tech

## Fake Biden robocall linked to Texas-based companies, New Hampshire attorney general announces

By David Wright, Yahya Abou-Ghazala and Brian Fung, CNN

② 6 minute read · Updated 3:58 PM EST, Tue February 6, 2024



TECHNOLOGY EXECUTIVE COUNCIL

## Generative AI financial scammers are getting very good at duping work email

PUBLISHED WED, FEB 14 2024 11:54 AM EST UPDATED 3 HOURS AGO



Ellen Sheng  
@ELLENSHENG

WATCH LIVE

### KEY POINTS

- Even companies that ban employees from using generative artificial intelligence are falling prey to financial scams that deploy the technology and amplify traditional phishing techniques used by hackers.
- Armed with tools like ChatGPT or its dark web equivalent FraudGPT, criminals can easily create realistic videos of profit and loss statements, fake IDs, false identities or even convincing deepfakes of company executives using their voice and image.
- A recent scam that cost a Hong Kong-based company over \$25 million shows how convincing the crimes have become and how difficult it is to detect them.

# How Large?



10 TBs of Data from web

| Dataset       | Sampling prop. | Epochs | Disk size |
|---------------|----------------|--------|-----------|
| CommonCrawl   | 67.0%          | 1.10   | 3.3 TB    |
| C4            | 15.0%          | 1.06   | 783 GB    |
| Github        | 4.5%           | 0.64   | 328 GB    |
| Wikipedia     | 4.5%           | 2.45   | 83 GB     |
| Books         | 4.5%           | 2.23   | 85 GB     |
| ArXiv         | 2.5%           | 1.06   | 92 GB     |
| StackExchange | 2.0%           | 1.03   | 78 GB     |

# How Large??

| Dataset       | Sampling prop. | Epochs | Disk size |
|---------------|----------------|--------|-----------|
| CommonCrawl   | 67.0%          | 1.10   | 3.3 TB    |
| C4            | 15.0%          | 1.06   | 783 GB    |
| Github        | 4.5%           | 0.64   | 328 GB    |
| Wikipedia     | 4.5%           | 2.45   | 83 GB     |
| Books         | 4.5%           | 2.23   | 85 GB     |
| ArXiv         | 2.5%           | 1.06   | 92 GB     |
| StackExchange | 2.0%           | 1.03   | 78 GB     |

LLAMA Model



> 5000 GPUs  
weeks, months  
~ \$10M

# How Much did DeepSeek Spend??

China's DeepSeek became the biggest topic in tech this week, with many in the industry and on Wall Street focused on a single number: \$6 million.

In DeepSeek's [paper](#) about its newest artificial intelligence model, the company said that its total training costs amounted to \$5.576 million, based on the rental price of [Nvidia's](#)  graphics processing units. DeepSeek included a clear caveat, saying that the number included only the model's "official training" and excluded the costs tied to "prior research and ablation experiments on architectures, algorithms, or data."

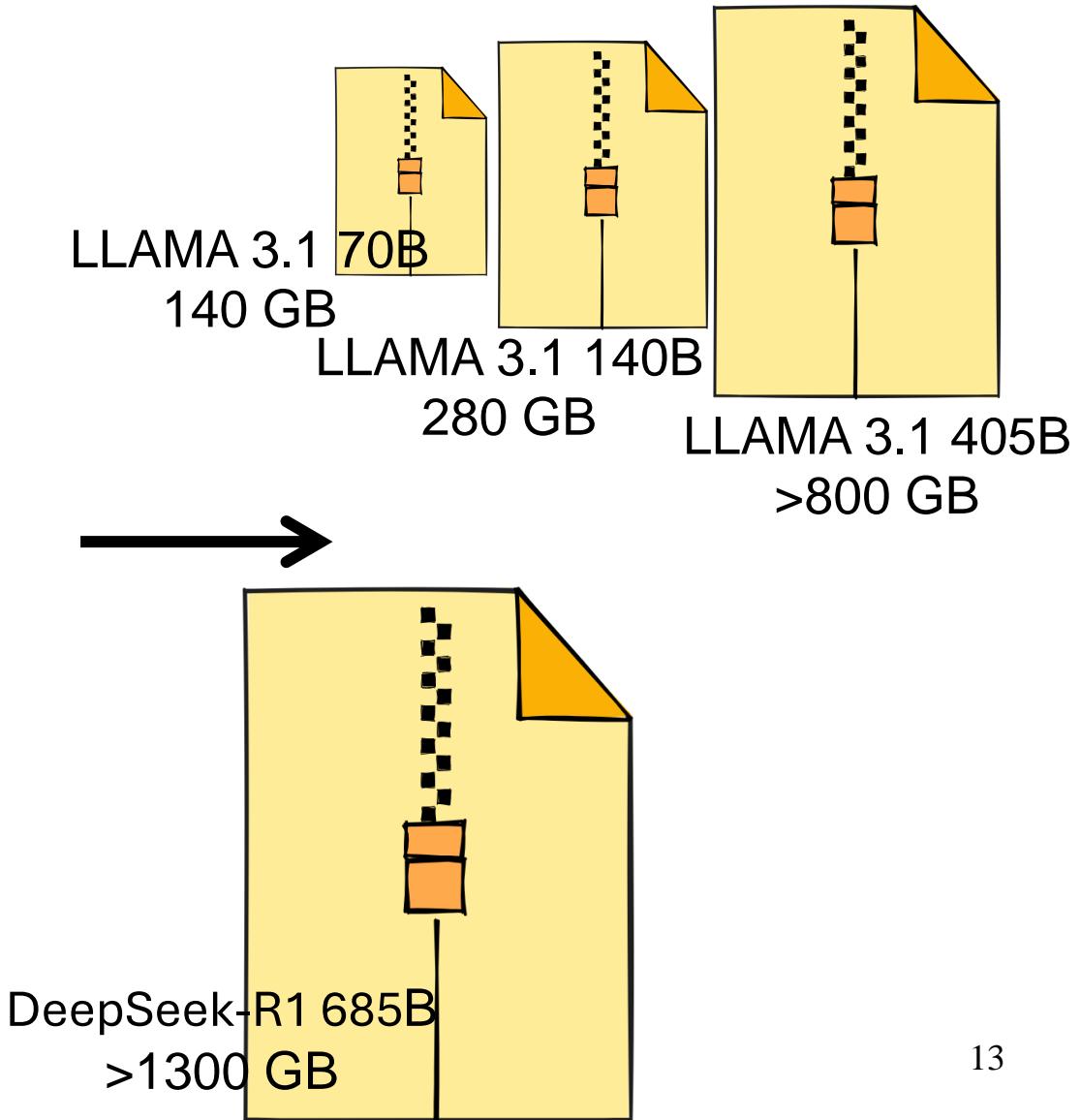
Early in the week, DeepSeek's AI Assistant took the coveted spot for most-downloaded free app in the U.S. on [Apple's](#) App Store, dethroning OpenAI's ChatGPT. Global tech stocks sold off, with chipmakers Nvidia and [Broadcom](#)  losing a combined \$800 billion in market cap on Monday.

A [new report from SemiAnalysis](#), a semiconductor research and consulting firm, added more context to DeepSeek's expenses. The firm estimated that DeepSeek's hardware spend is "well higher than \$500M over the company history," adding that R&D costs and total cost of ownership are significant. Generating "synthetic data" for the model to train on would require "considerable amount of compute," SemiAnalysis wrote.

# How Large???



> 5000 GPUs  
weeks, months  
~ \$10M



# **Large Language Models**

# Tokenizers

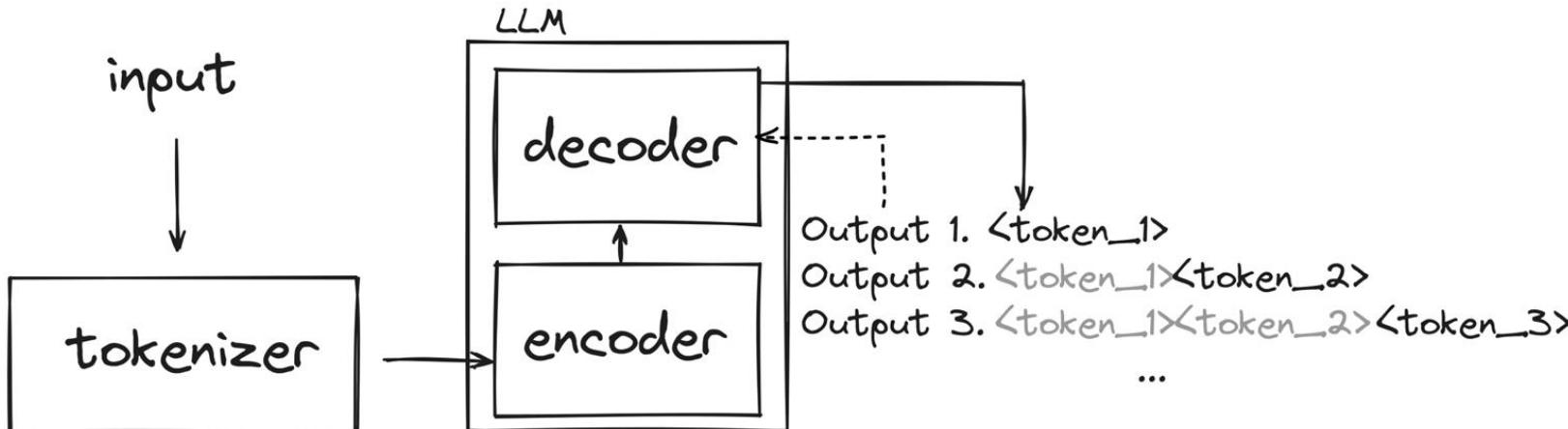
Tokenization is breaking down human language into smaller units like words or subwords to convert text into machine-readable numerical representations.

Ah, the siren call of cosmopolitan metropolises! One could scarcely conceive of a more tapestried itinerary than a peregrination commencing in the inimitable Paris, a city redolent with artistic efflorescence and gourmand delights, thence proceeding to the venerable Rome, a veritable palimpsest of ancient grandeur and baroque magnificence, culminating in the multifarious London, a sprawling agglomeration pulsating with contemporary verve and historical gravitas.

Ah, il richiamo ammaliatore delle metropoli cosmopolite! Si potrebbe a malapena concepire un itinerario più variopinto di un pellegrinaggio che inizi nell' inimitabile Parigi, una città ricca di efflorescenza artistica e delizie gourmand, procedendo poi alla venerabile Roma, un vero e proprio palinsesto di antica grandezza e barocca magnificenza, culminando nella multiforme Londra, un vasto agglomerato pulsante di vivacità contemporanea e gravità storica.

# Three main steps for text generation

- .The input text is passed to a tokenizer that generates unique numerical representation of every token.
- .The tokenized input text is passed to the Encoder part of the pre-trained model. The Encoder processes the input and generates a feature representation that encodes inputs meaning and context.
- .The Decoder takes the feature representation from the Encoder and starts generating new text based on that context token by token.

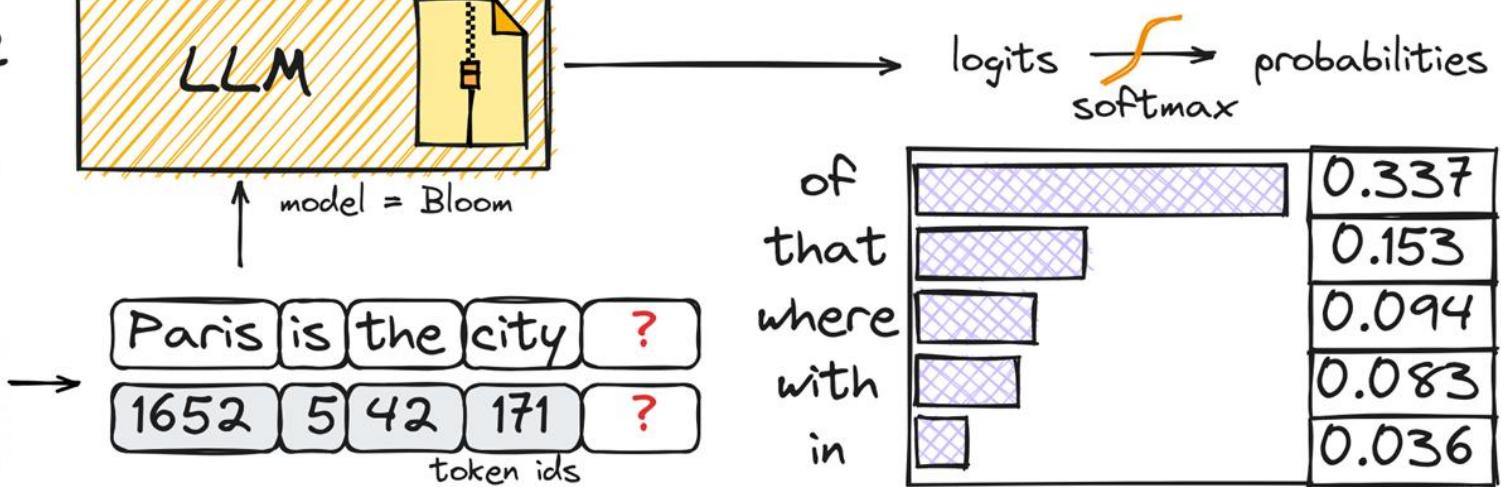
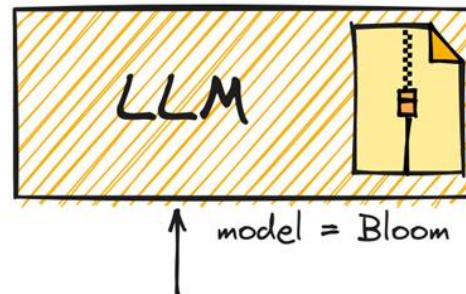


# Decoding the outputs

Say, we want to generate the continuation of the phrase "Paris is the city ...". The Encoder (we'll be using Bloom-560m model) sends logits for all the tokens we have (if you don't know what logits are – consider them as scores) that can be converted, using softmax function, to probabilities of the token being selected for generation

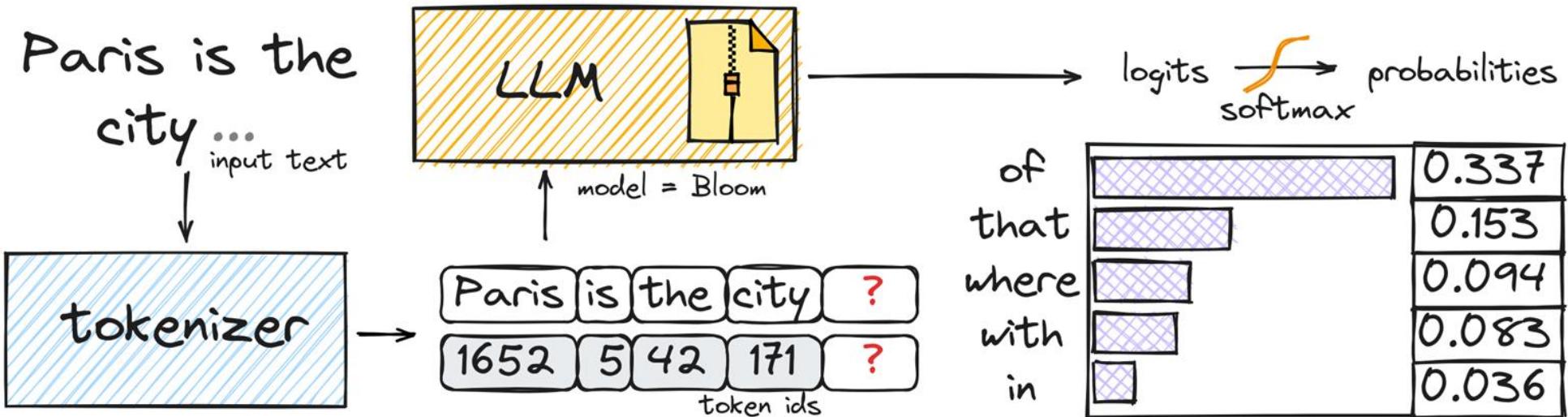
Paris is the  
city ...  
↓  
input text

tokenizer

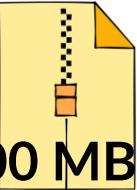


# Possible following tokens

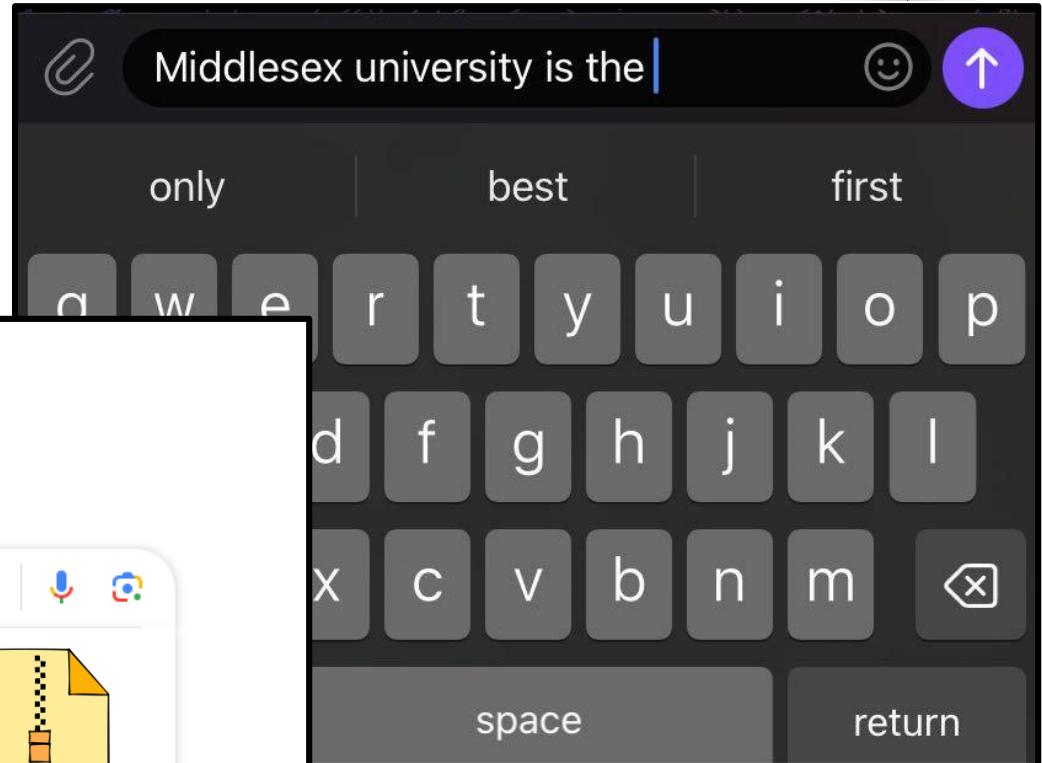
- .Paris is the city **of** love.
- .Paris is the city **that** never sleeps.
- .Paris is the city **where** art and culture flourish.
- .Paris is the city **with** iconic landmarks.
- .Paris is the city **in** which history has a unique charm.



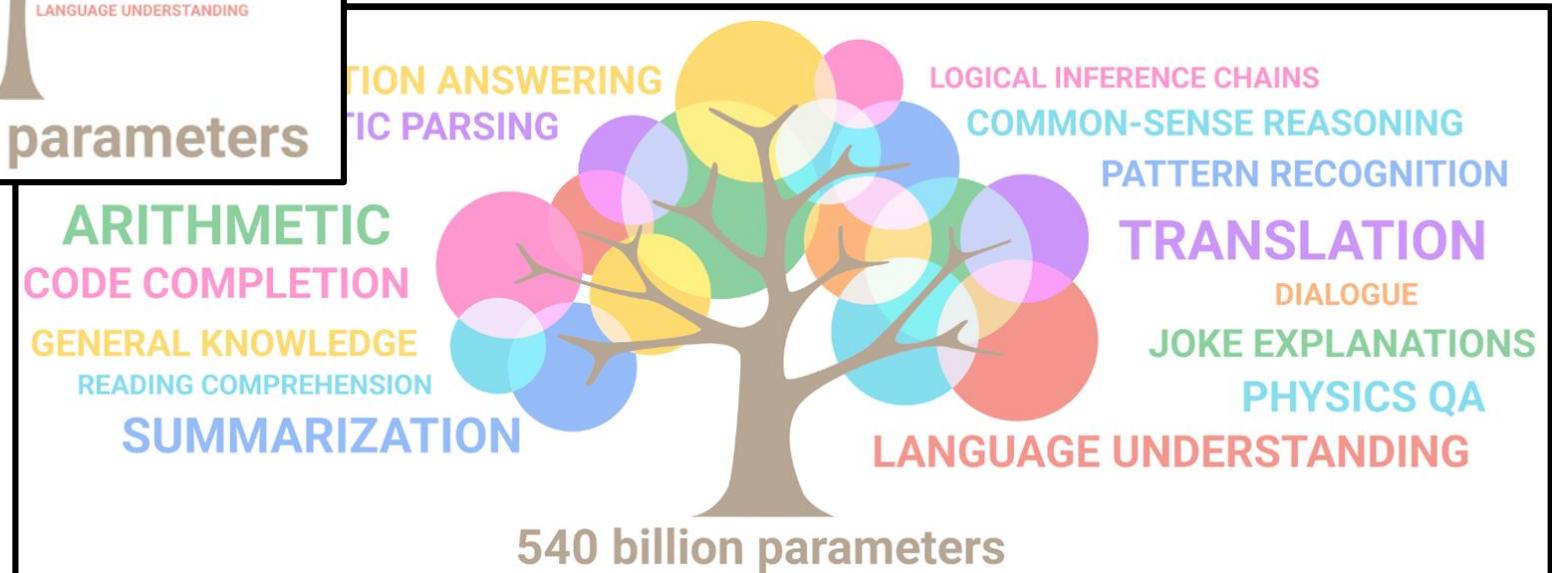
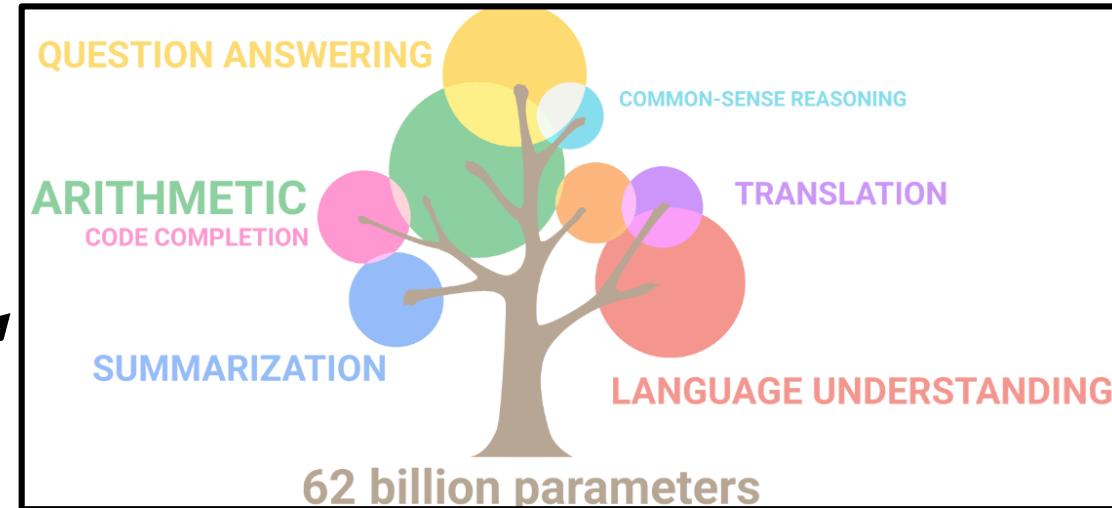
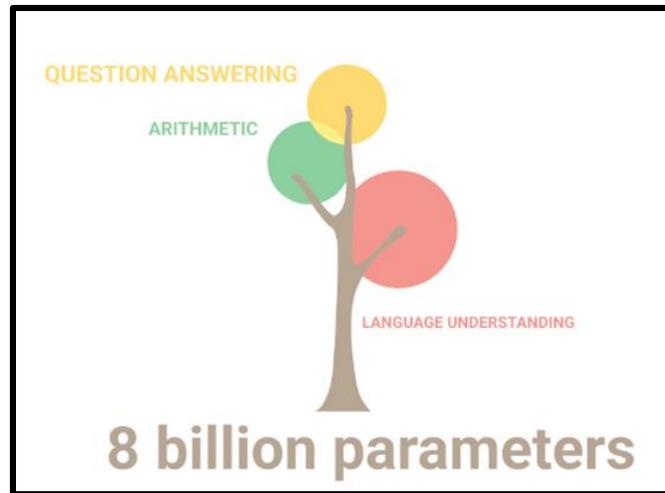
# Language models b.c.



Language models existed before ChatGPT. The difference is the size. We're talking about a shift from hundreds of megabytes to hundreds of gigabytes!



# Model skills



# Playground OpenAI

Paris is the city of light, love, and beauty. It is known for its iconic landmarks, rich history, and romantic atmosphere. There is no shortage of things to do in Paris, from visiting the Louvre Museum to exploring the Eiffel Tower.

Paris is a place where you can get lost in time, surrounded by history and culture.

getting lost in the past centuries. Paris is a place where = 1.77% that = 1.29%

for = 0.68%

with = 0.41%

Total: -0.08 logprob on 1 tokens  
(96.58% probability covered in top 5 logits)

**gpt-3.5-turbo-instruct**

# Knowledge of a Large Language Model

## Paris

Article Talk

From Wikipedia, the free encyclopedia

*This article is about the capital of France. For other uses, see [Paris \(disambiguation\)](#).*

**Paris** (French pronunciation: [paʁi] ⓘ) is the [capital](#) and [most populous city](#) of [France](#). With an official estimated [population](#) of 2,102,650 residents as of 1 January 2023<sup>[2]</sup> in an area of more than 105 km<sup>2</sup> (41 sq mi),<sup>[5]</sup> Paris is the [fourth-most populated city](#) in the [European Union](#) and the [30th most densely populated city](#) in the [world](#) in 2022.<sup>[6]</sup> Since the 17th century, Paris has been one of the [world's major centres](#) of [finance](#), [diplomacy](#), [commerce](#), [culture](#), [fashion](#), and [gastronomy](#). For its leading role in the [arts](#) and [sciences](#), as well as its early and extensive system of street lighting, in the 19th century, it became known as the [City of Light](#).<sup>[7]</sup>

The City of Paris is the centre of the [Île-de-France region](#), or Paris Region, with an official estimated population of 12,271,794 inhabitants on 1 January 2023, or about 19% of the population of France.<sup>[2]</sup> The Paris Region had a [GDP](#) of €765 billion (US\$1.064 trillion, [PPP](#))<sup>[8]</sup> in 2021, the highest in the European Union.<sup>[9]</sup> According to the [Economist Intelligence Unit](#) Worldwide Cost of Living Survey, in 2022, Paris was the city with the ninth-highest cost of living in the world.<sup>[10]</sup>

- Facts
- Dates
- Entity Relations
- Punctuation
- Synonyms
- Semantics
- Translation
  
- Complex:
- Reasoning
- Math

# 1. Greedy Sampling

In the greedy strategy, the model always chooses the token it believes is the most probable at each step – it doesn't consider other possibilities or explore different options. The model selects the token with the highest probability and continues generating text based on the selected choice.

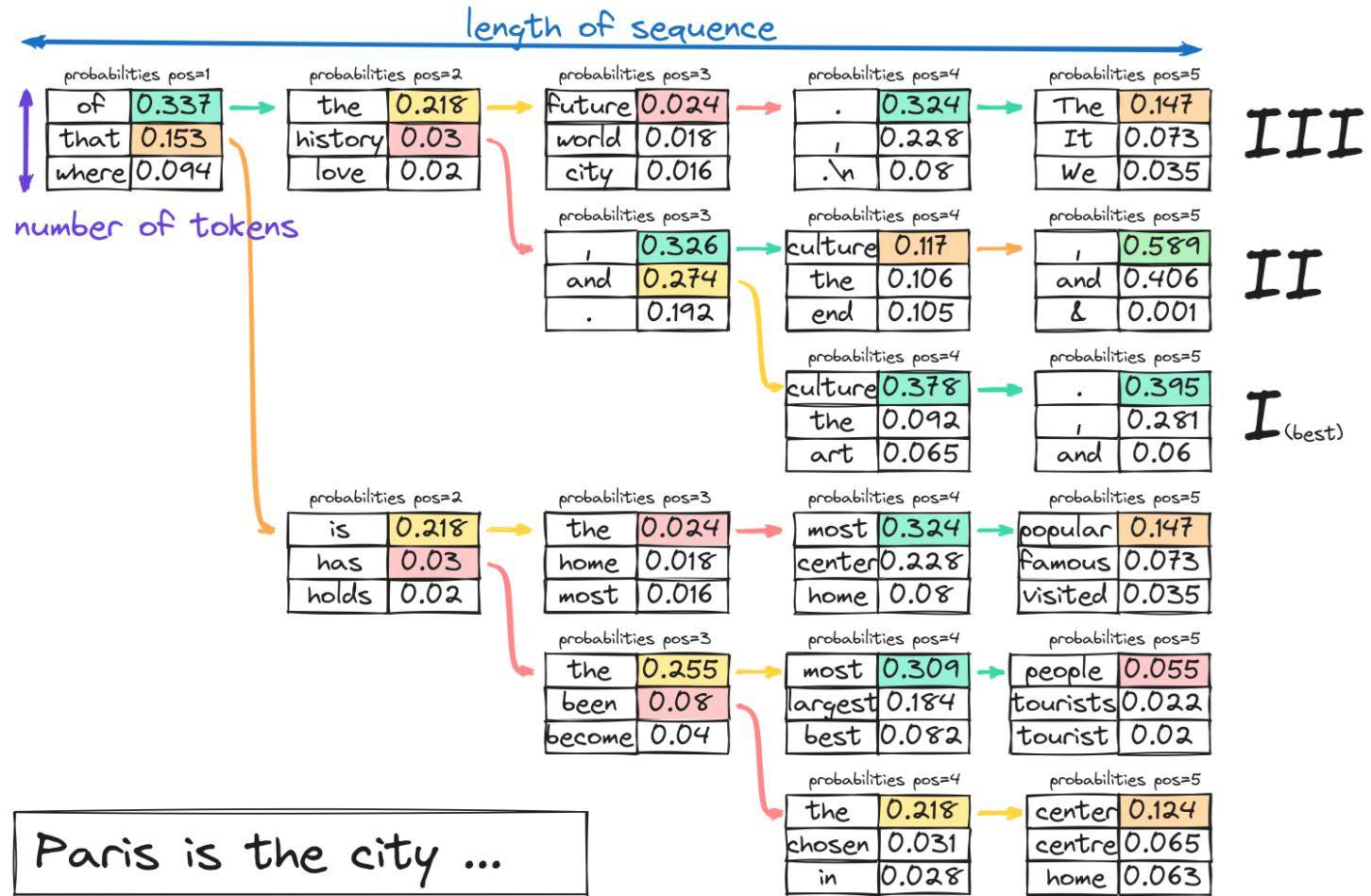
Generated output: *Paris is the city of the future. The*



## 2. Beam search

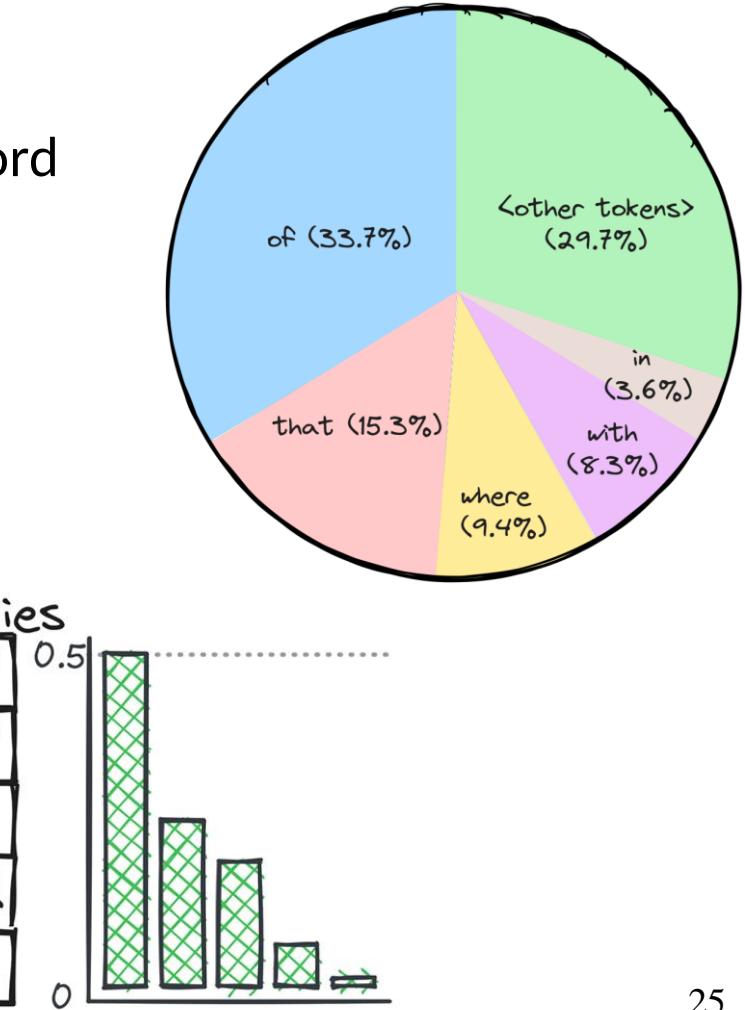
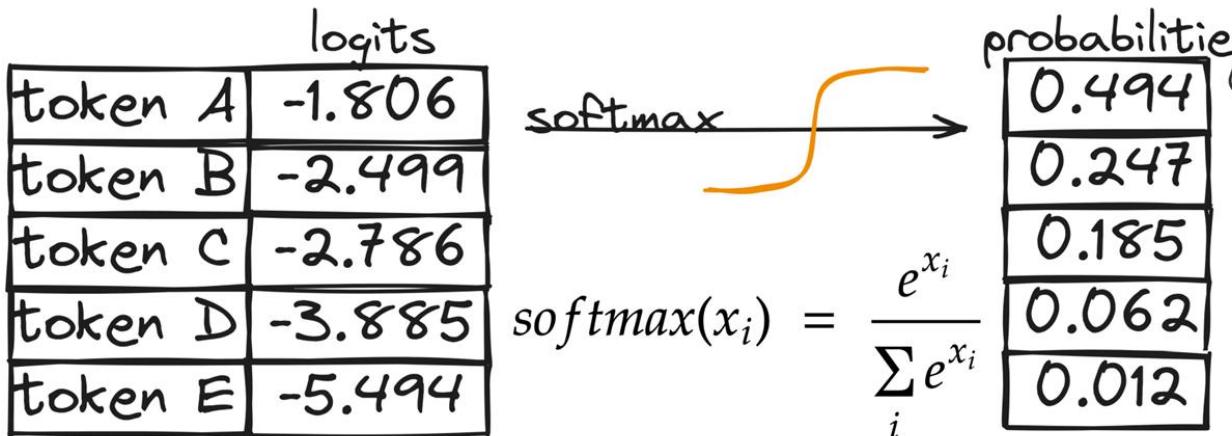
In beam search, instead of just considering the most likely token at each step, the model considers a set of the top "k" most probable tokens. This set of k tokens is called a "beam."

Generated output:  
Paris is the city of  
history and culture.



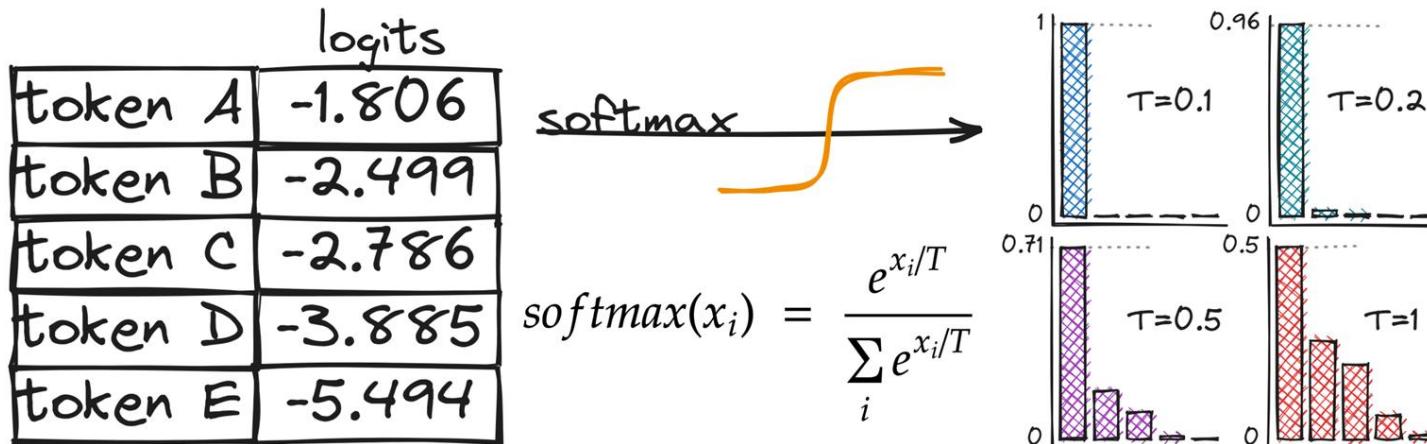
### 3. Normal Random Sampling

In normal random sampling you select the next word by choosing a random value and mapping it to the token got picket. Imagine it as spinning a wheel, where the area of each token is defined by its probability. The higher the probability – the more chances the token would get selected.



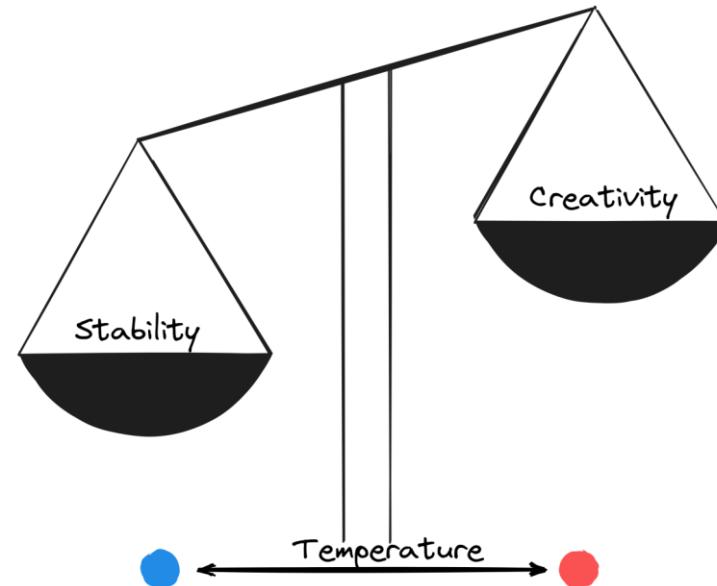
## 4. Random Sampling + Temperature

We've been using the softmax function to convert logits to probabilities. We now introduce temperature – a hyperparameter that affects the randomness of the text generation. The difference with the classic softmax is in the denominator - we divide by T. Higher values of temperature (e.g., 1.0) make the output more diverse, while lower values (e.g., 0.1) make it more focused and deterministic. In fact, T = 1 will lead to the softmax function we used initially.



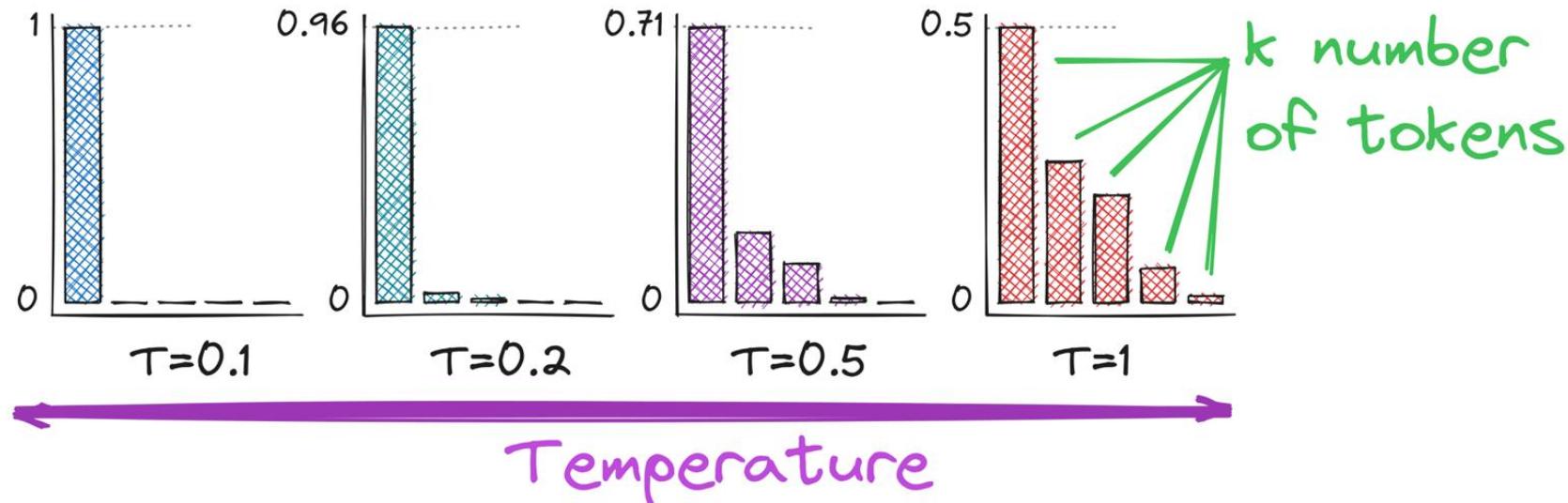
# Temperature

Temperature controls the randomness of token selection during decoding. Higher temperature boosts creativity, whereas lower temperature is more about coherence and structure. While embracing creativity allows for fascinating linguistic adventures, tempering it with stability ensures the elegance of the generated text.



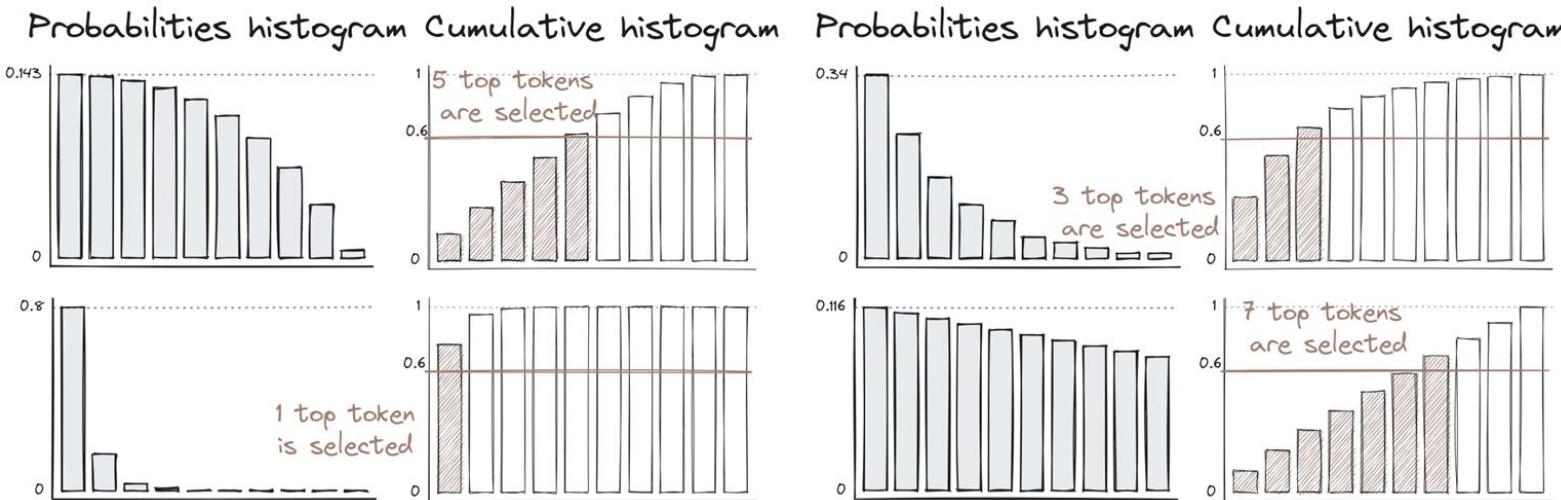
## 5. Top-k sampling

We can now shift probabilities with temperature. Another enhancement is to use top-k tokens rather than all of them. This will increase the stability of the text generation, not decreasing creativity too much. Basically, it's now random sampling with temperature for only top k tokens.



# 6. Top-p or nucleus sampling

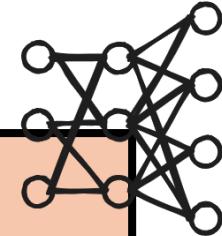
Nucleus sampling is designed to address some limitations of different sampling techniques. Instead of specifying a fixed number of "k" tokens to consider, a probability threshold "p" is used. This threshold represents the cumulative probability that you want to include in the sampling.



# Summary

| Method                           | Description  | Diversity | Performance | Quality of Output          |
|----------------------------------|--|-----------|-------------|----------------------------|
| Greedy Search                    | Selects the most probable token at each step.          | Low       | High        | May lack diversity         |
| Beam Search                      | Considers top-K probable tokens in parallel.           | Moderate  | Medium-High | Improved diversity         |
| Random Sampling                  | Randomly selects tokens based on probabilities.        | High      | Medium      | May lack coherence         |
| Random Sampling with Temperature | Adds randomness based on temperature.                  | High      | Medium      | More controlled randomness |
| Top-K Sampling                   | Selects top-K probable tokens randomly.                | High      | Medium      | Improved diversity         |
| Nucleus Sampling                 | Selects tokens until cumulative probability $\leq p$ . | High      | Medium-High | Improved coherence         |

# Practice with LLMs and tokenizers



# .from\_pretrained(<model>)

In many pre-trained language models, the tokenizer and the model architecture are designed and trained together. The reason for this coupling is that the tokenizer and the model architecture need to be compatible with each other to ensure consistent tokenization and decoding.

If the tokenizer and the model architecture were different or not synchronized, it could lead to tokenization errors, mismatched embeddings, and incorrect predictions.



```
1 tokenizer = AutoTokenizer.from_pretrained("some_model")
2 model = BloomForCausalLM.from_pretrained("some_model")
```

# API keys

You will first need an API key for the LLM provider you want to use. Currently, we must choose between proprietary or open-source foundation models based on a trade-off mainly between performance and cost.

Many open-source models are organized and hosted on Hugging Face as a community hub



```
1 os.environ["OPENAI_API_KEY"] = ... # API_TOKEN  
2 os.environ["HUGGINGFACEHUB_API_TOKEN"] = ... # API_TOKEN
```

# API keys

Proprietary models are closed-source foundation models owned by companies. They usually are larger than open-source models and thus have better performance, but they may have expensive APIs.

*Examples: OpenAI, cohere, AI21 Labs, Anthropic, etc.*

Open-source models are usually smaller models with lower capabilities than proprietary models, but they are more cost-effective than proprietary ones.

*Examples: BLOOM (BigScience), LLaMA (Meta), Flan-T5 (Google), etc.*



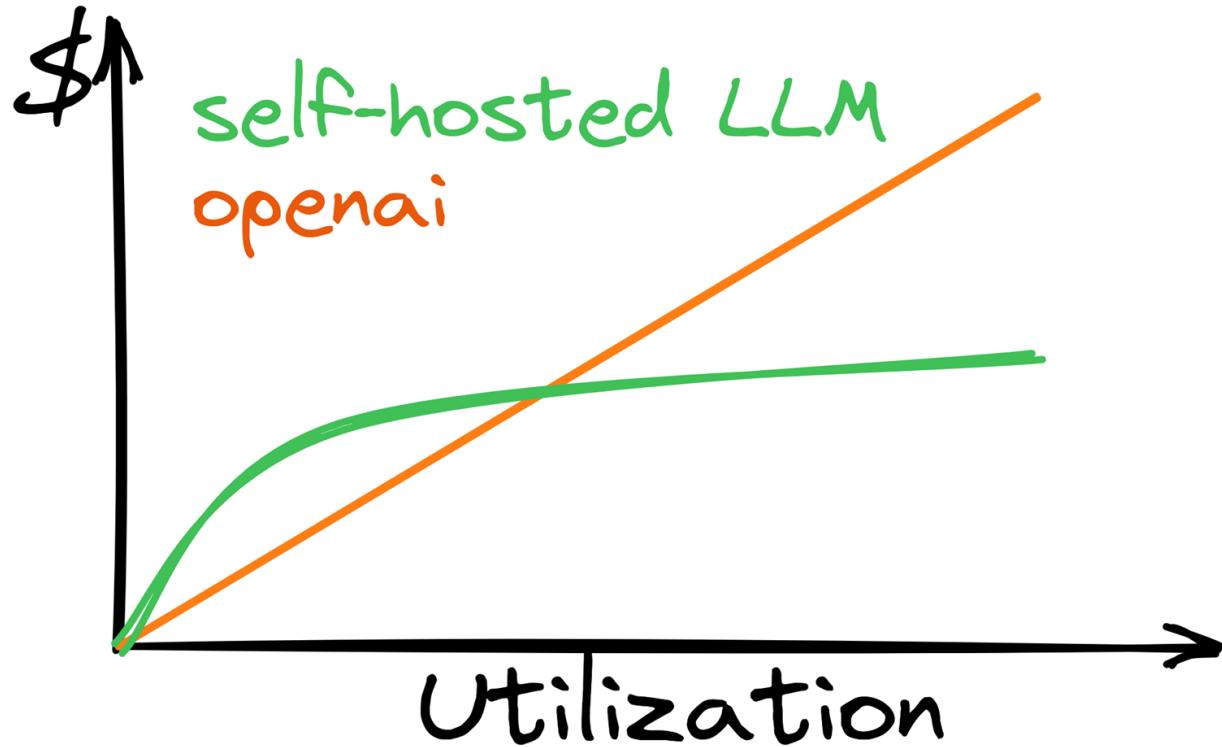
```
1 # Proprietary LLM from e.g. OpenAI
2 from langchain.llms import OpenAI
3 llm = OpenAI(model_name="text-davinci-003")
4
5 # Alternatively, open-source LLM hosted on Hugging Face
6 from langchain import HuggingFaceHub
7 llm = HuggingFaceHub(repo_id = "google/flan-t5-xl")
```

# Cost per token

| Model Name                  | Provider  | Context Length | Input Cost (\$/1K tokens) | Output Cost (\$/1K tokens) |
|-----------------------------|-----------|----------------|---------------------------|----------------------------|
| OpenAI GPT-4o               | OpenAI    | 128,000        | \$0.005                   | \$0.015                    |
| Anthropic Claude 3.5 Sonnet | Anthropic | 200,000        | \$0.003                   | \$0.015                    |
| Google Gemini 1.5 Pro       | Google    | 1,000,000      | \$0.007                   | \$0.021                    |
| Cohere Command R            | Cohere    | 128,000        | \$0.15                    | \$0.60                     |
| Mistral Mixtral 8x7B        | Mistral   | 32,768         | \$0.0007                  | \$0.0007                   |
| Meta Llama 3 70B            | Meta      | 8,192          | \$0.00059                 | \$0.00079                  |

April 2025

# Self-hosted or API?



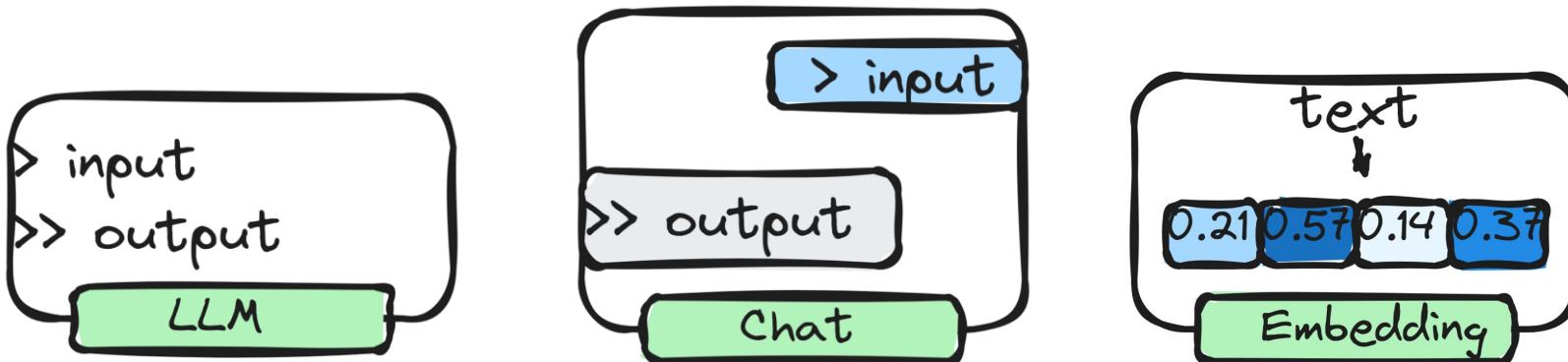
# What to choose?

| You want           | To choose          |
|--------------------|--------------------|
| private data usage | self-hosted LLM    |
| control over model | self-hosted LLM    |
| quick development  | openai             |
| low costs          | openai             |
| minimal infra      | openai (probably)  |
| top quality        | openai (currently) |

# Models

LangChain supports a variety of different models, so you can choose the one that suits your needs best:

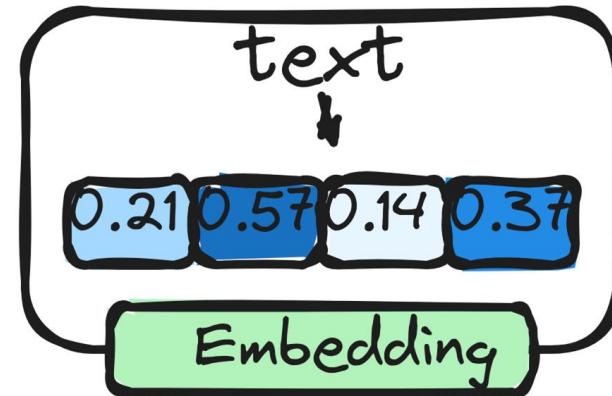
- **Language Models** are used for generating text
  - LLMs utilize APIs that take input text and generate text outputs
  - ChatModels employ models that process **chat messages** and produce responses
- **Text Embedding Models** convert text into numerical representations



# Embeddings

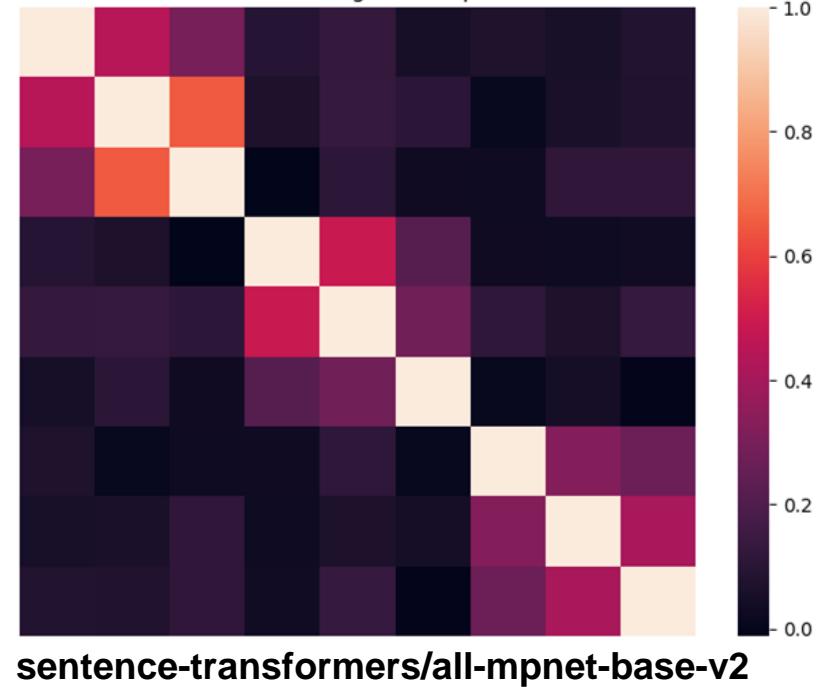
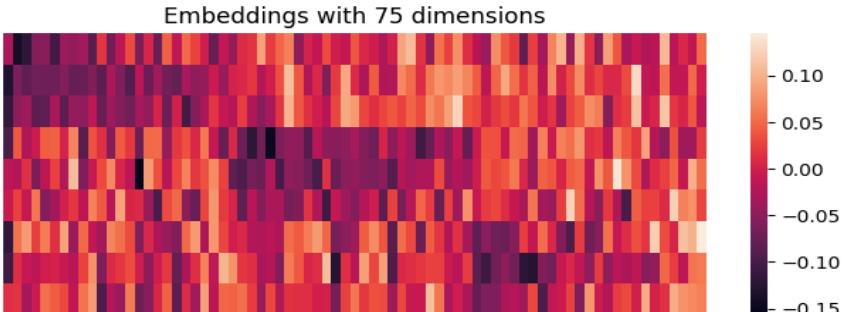
Embeddings are compact numerical representations of words or entities that help computers understand and process language more effectively. These representations encode the meaning and context of words, allowing machines to work with language in a more meaningful and efficient way.

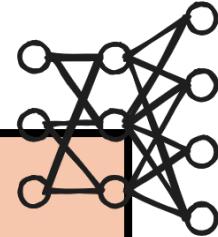
- OpenAIEmbeddings
- HuggingFaceEmbeddings
- GPT4AllEmbeddings
- etc
- SpacyEmbeddings
- FakeEmbeddings



# Embeddings example

- 1) Best travel neck pillow for long flights
- 2) Lightweight backpack for hiking and travel
- 3) Waterproof duffel bag for outdoor adventures
- 4) Stainless steel cookware set for cooktops
- 5) High-quality chef's knife set
- 6) High-performance stand mixer for baking
- 7) New releases in fiction literature
- 8) Inspirational biographies and memoirs
- 9) Top self-help books for personal growth





# Practice with embeddings

# Large Language and Chat Models

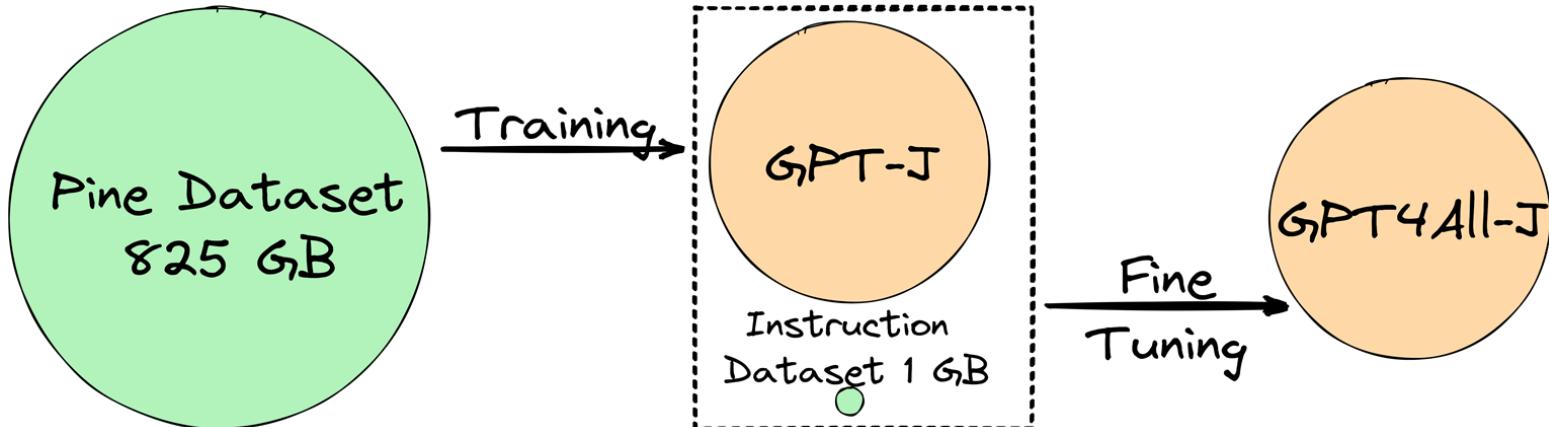
A **large language model** is an advanced type of language model that is trained using deep learning techniques on **massive amounts of text data**.

**Chat models** are usually backed by a language model, but their APIs are more structured. These models take a list of Chat Messages as input, and return a Chat Message.

| Text               | Prompt                      | Response                     |
|--------------------|-----------------------------|------------------------------|
| <any form of text> | <stackoverflow question X>  | <response X>                 |
| <article X>        | <Q&A platform question Y>   | <Q&A platform answer Y>      |
| <book Y>           |                             |                              |
| <history Z>        | <Human generated request Z> | <Human generated response Z> |
| etc                | <Chat history Person A>     | <Chat history Person B>      |

# Large Language and Chat Models

| text (string)  | prompt (string)  | response (string)  |
|--|--|--|
| "I've learned the nitrogen vacancies used in Memristors are for "switching", between excited...    | "<p>In Angular if I had a few panels, similar to tabs how do you open the panel on click and animat... | "To accomplish this in Angular, you can use ngAnimate and ng-show/ng-hide directives. Here's...    |
| "Volunteer Services Volunteer Services As Charleston Area Medical Center volunteers, our...        | "Given the following scientific paper: This paper presents a novel approach for temporal and semant... | "This scientific paper describes three different approaches for segmenting and analyzing edited... |
| "Q: Python: My return variable is always None So I found a strange thing that happens in python... | "<p>i am using the following code:</p> <pre><code>package Presentacion; import...                      | "The error is caused by a typo in the code. The correct class name is "PrintWriter" and not...     |



# Asking GPT a question

In order to ask model a question there is even no need to connect LangChain.

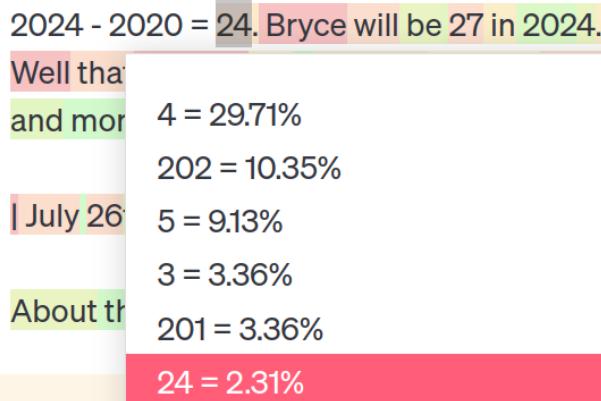
In this example, we're preparing the prompt, connecting to *openai* package and returning extracted response.

```
● ● ●
1 import openai
2
3 def get_completion(prompt, model="gpt-3.5-turbo"):
4     """
5         This function takes a user prompt, sends it to the model, and returns the generated response.
6
7     Parameters:
8         prompt (str): The user prompt to be sent to the model.
9         model (str): The name of the language model to use. Defaults to "gpt-3.5-turbo".
10
11    Returns:
12        str: The generated response from the language model.
13    """
14
15    # Prepare the user prompt in a format suitable for the OpenAI Chat API
16    messages = [{"role": "user", "content": prompt}]
17
18    # Use the OpenAI Chat API to get a completion (response) from the language model
19    response = openai.ChatCompletion.create(
20        model=model,
21        messages=messages,
22        temperature=0.0,
23    )
24
25    # Extract the generated response from the API response
26    generated_response = response.choices[0].message["content"]
27
28    return generated_response
```

# Asking GPT a question

As one can see, simple questions can be answered by the model correctly.

But for complicated requests LangChain may be the way to go.



```
1 get_completion("What do they call a Big Mac in France?")
2
3 >> 'In France, a Big Mac is called "Le Big Mac."'
4
5 get_completion("What is the average age of retirement of the
6 last 5 US presidents?")
7
8 >> The average age of retirement of the last 5 US presidents is
9 approximately 77 years old. Here are the retirement ages of the
10 last 5 presidents:
11 1. Donald Trump - Retired at the age of 74 in 2021.
12 2. Barack Obama - Retired at the age of 55 in 2017.
13 3. George W. Bush - Retired at the age of 62 in 2009.
14 4. Bill Clinton - Retired at the age of 54 in 2001.
15 5. George H. W. Bush - Retired at the age of 68 in 1993.
16
17 Adding up these retirement ages and dividing by 5 gives an
18 average retirement age of 77.
```

# **Basics of LangChain**

# Hallucinations

Hallucinations can be understood as answering confidently when asked for the weather forecast in a fictional city or providing fabricated references in an academic context

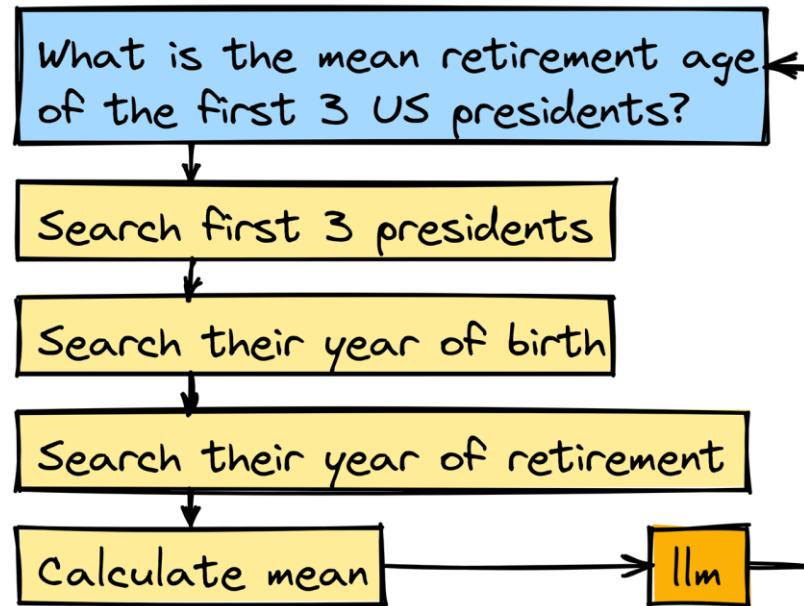
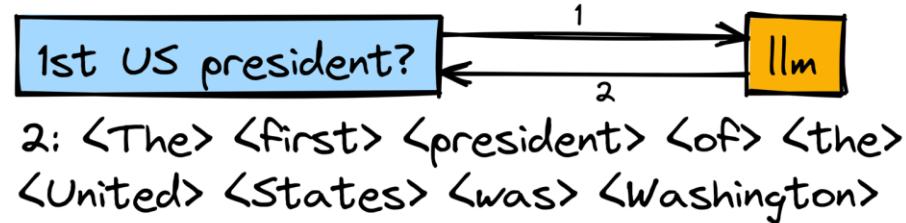
*But what are the reasons of that?*

The earliest mention of artificial intelligence in the New York Times was in a ~~February 19, 1950~~ November 1950 article titled “~~Thinking Machines.~~” “‘Revolution’ is Seen in ‘Thinking Machines.’” The article, by ~~Walter Sullivan~~, reported on a meeting of the ~~American Association for the Advancement of Science~~, where a number of scientists discussed the possibility of creating machines that could think.

# LLM LangChain Application

LLMs are used in LangChain applications in several purposes:

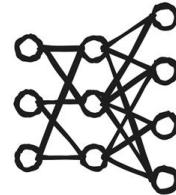
- language model to answer user response
- “llm butter” for LangChain components



# What is LangChain? Quick glance

LangChain is a framework that makes it easier to create applications using large language models (LLMs).

Models



Prompts



Chains



Memory



Indexes



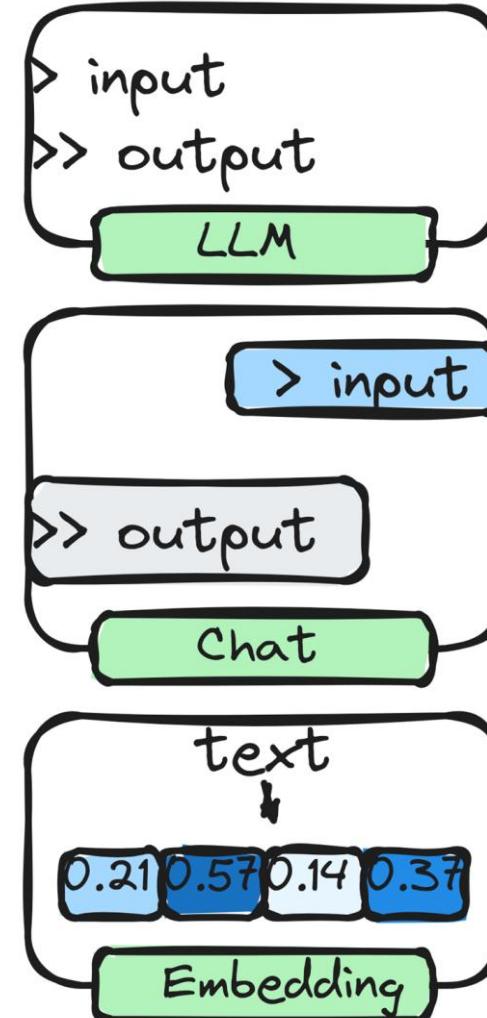
Agents and Tools



# Models: reminder

LangChain supports a variety of different models, so you can choose the one that suits your needs best

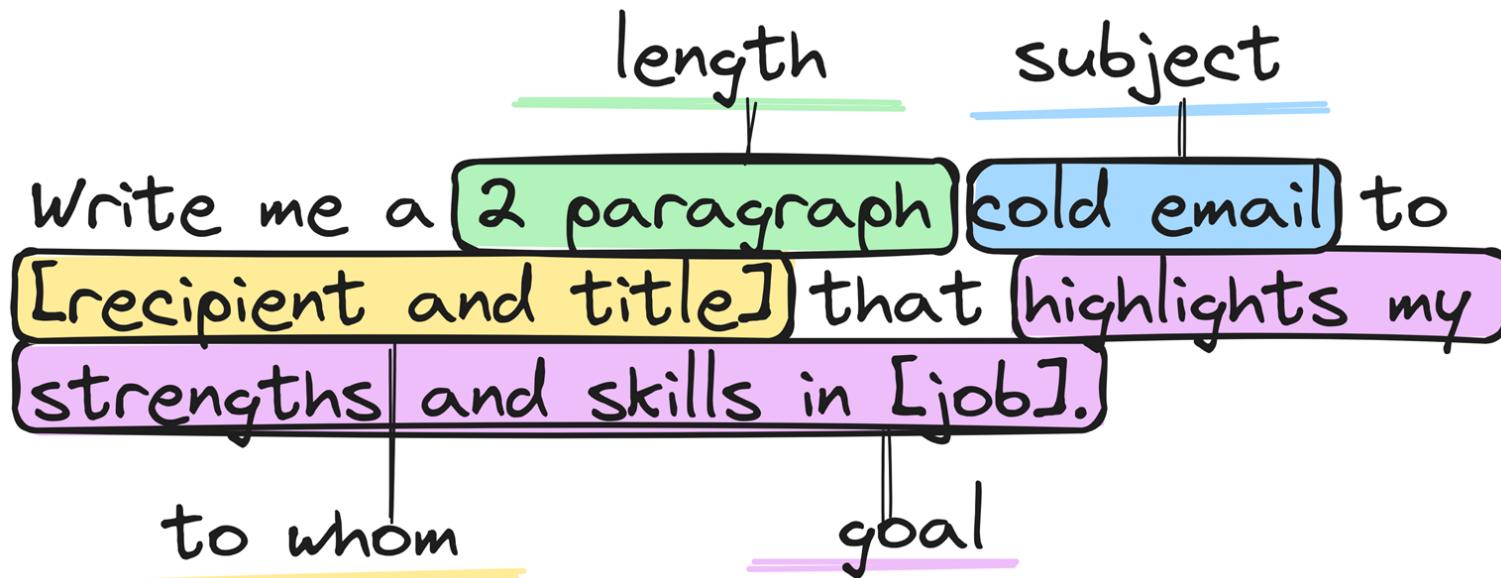
- **Language Models** are used for generating text
  - LLMs utilize APIs that take input text and generate text outputs
  - ChatModels employ models that process chat messages and produce responses.
- **Text Embedding Models** convert text into numerical representations



# Prompts

Prompts are the instructions that you give to the LLM. They tell the LLM what you want it to do, and how you want it to do it.

Prompts can take the form of a string (for Language Models) or a list of messages (for Chat Models).

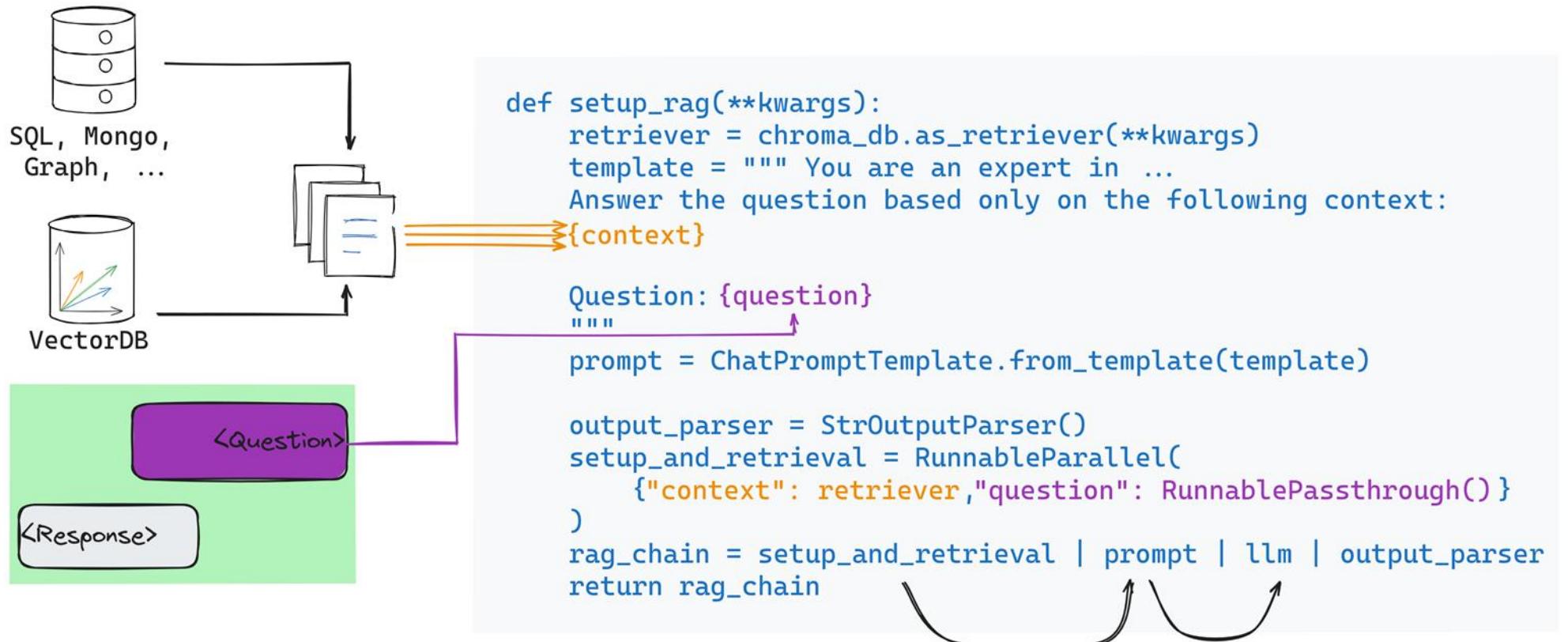


# Prompts



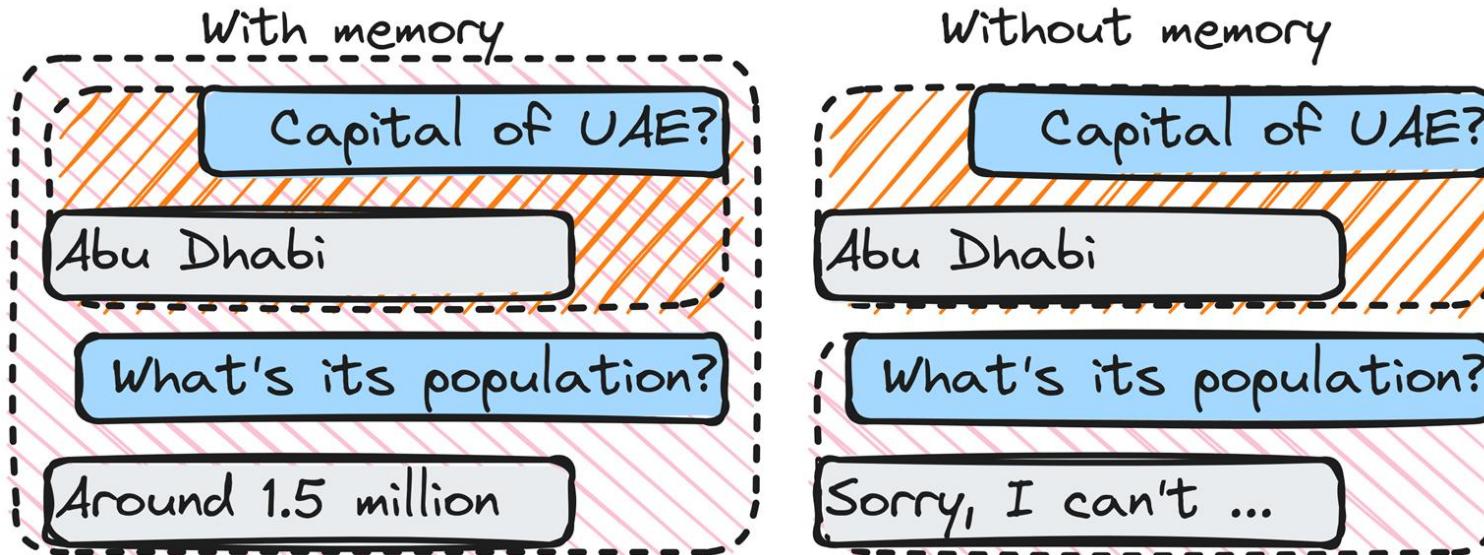
```
1 def setup_rag(**kwargs):
2     retriever = chroma_db.as_retriever(**kwargs)
3
4     template = """Answer the question based only on the following context:
5     {context}
6
7     Question: {question}
8     """
9
10    prompt = ChatPromptTemplate.from_template(template)
11    output_parser = StrOutputParser()
12
13    setup_and_retrieval = RunnableParallel(
14        {"context": retriever, "question": RunnablePassthrough()})
15    rag_chain = setup_and_retrieval | prompt | llm | output_parser
16
17    return rag_chain
```

# Prompts



# Memory

LangChain introduces memory components that enable the retention and utilization of past interactions. By default, LangChain components are stateless, treating each query independently. Memory in LangChain allows for the storage and management of previous chat messages.



# Vectorstores



```
1 from langchain.vectorstores import DocArrayInMemorySearch
2
3 db = DocArrayInMemorySearch.from_documents(
4     documents=documents,
5     embedding=embedding
6 )
7
8 docs = db.similarity_search(query)
9
10 #or
11
12 docs = db.similarity_search_with_score(query)
```

# Chroma

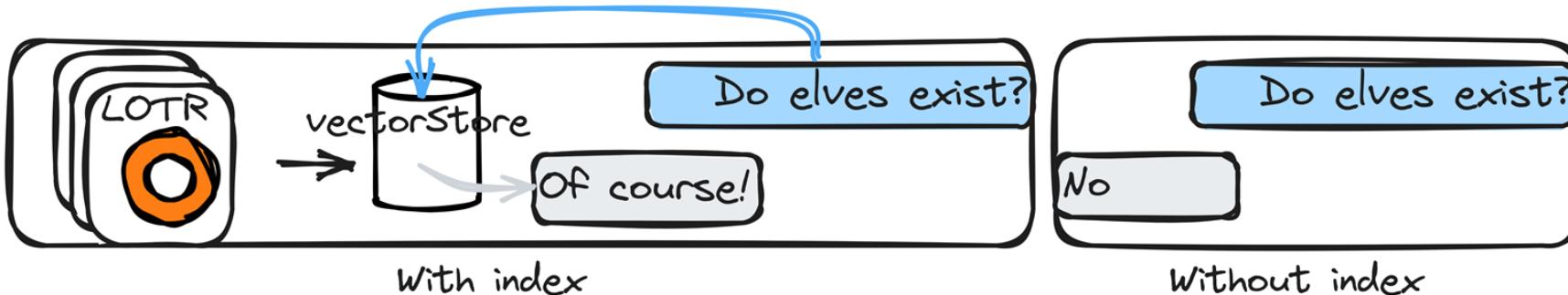


```
1 from langchain.vectorstores import Chroma
2
3 chroma_db = Chroma.from_documents(
4     documents=documents,
5     embedding=openai_embedding,
6     persist_directory=persist_directory
7 )
8
9 docs = chroma_db.similarity_search_with_score(query,k=3)
```

# Indexes

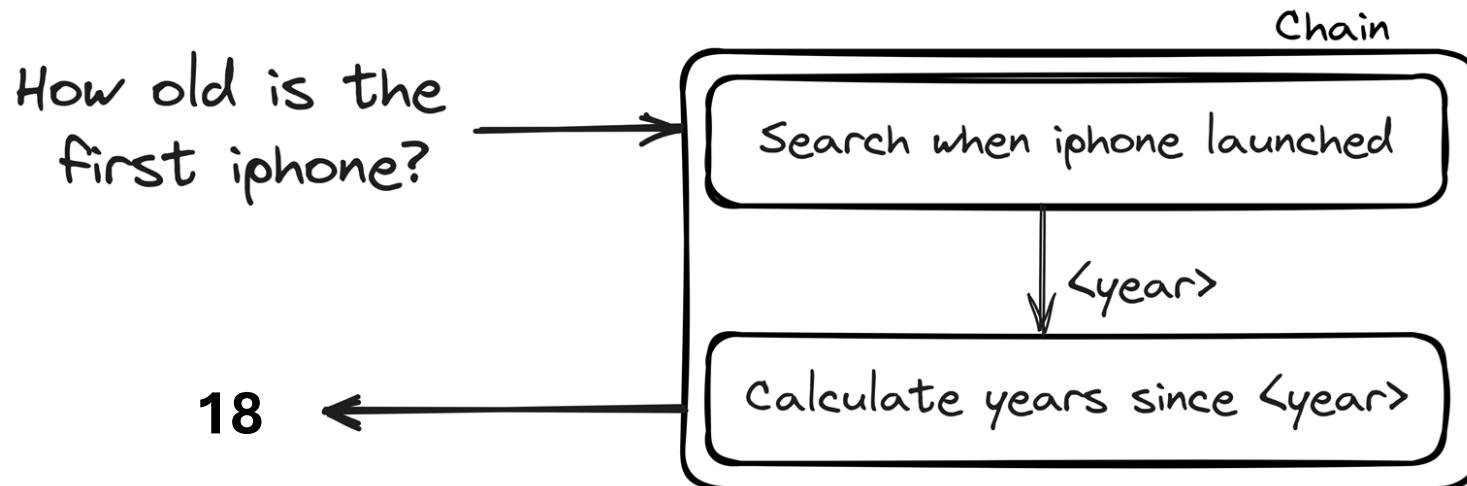
Indexes are databases of information that can be used to provide context to the LLM.

For example, if you are asking the LLM to answer a question about a particular topic, you could use an index to provide the LLM with a list of relevant articles or websites.



# Chains

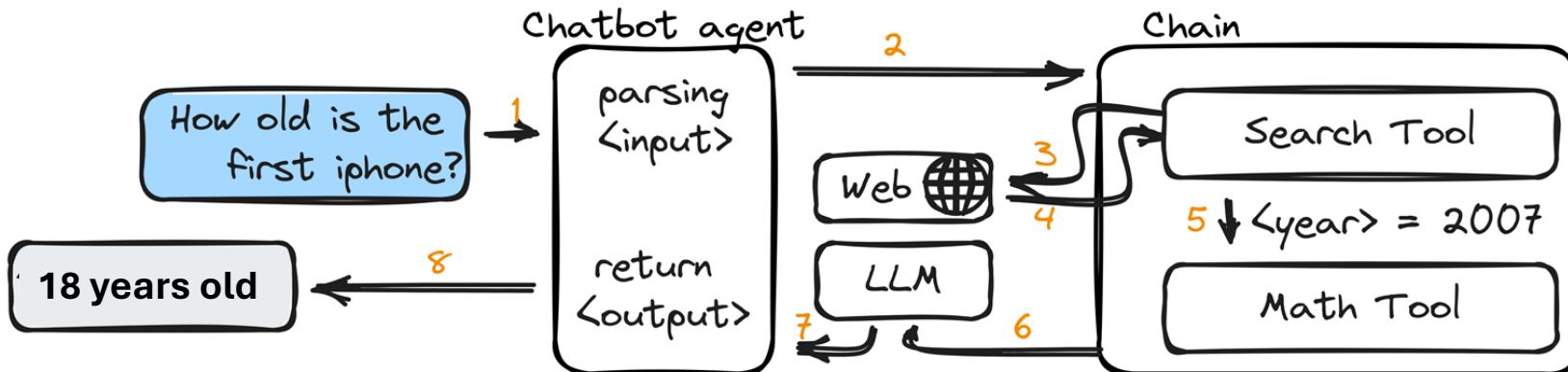
Chains are sequences of calls to the LLM. They allow you to perform complex tasks by chaining together multiple calls to the LLM. There are multiple types of chains to solve different cases.



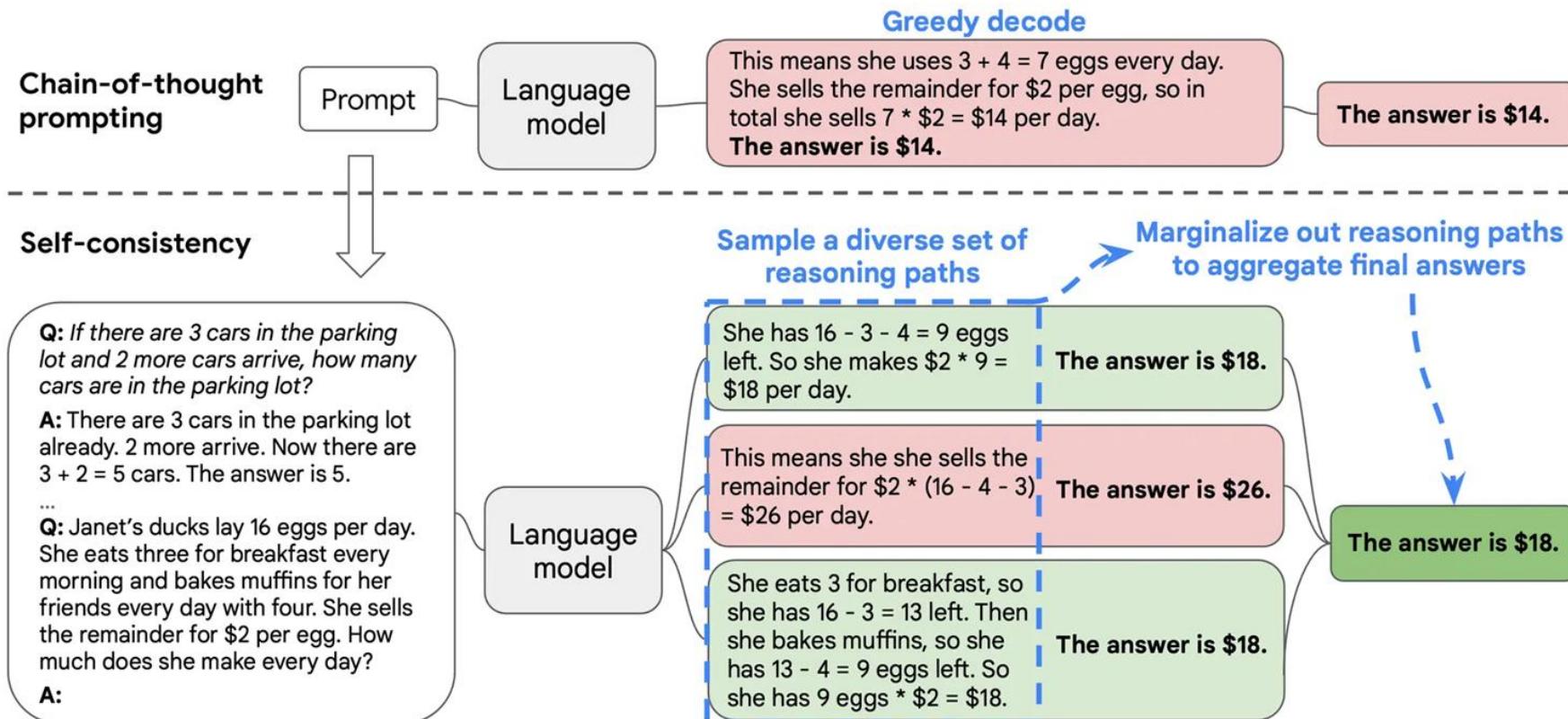
# Agents and Tools

Agents are objects that manage interaction between the user and the LLM. They handle things like *parsing input*, *generating output* and determining the sequence of actions to follow and tools to use.

A tool is a function designed to perform a specific task. Examples of tools can range from *Google Search*, *DB lookups*, *API calls*, to other chains. The standard interface for a tool is a function that accepts a string as input and returns a string as output.



# Agents and Tools: Prompt Alternative

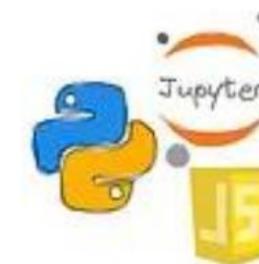


# Tools: Data Loaders

Data Integration  
& Databases



Communication  
platforms



Data Analysis  
& ML



Data Loaders

Social &  
web services

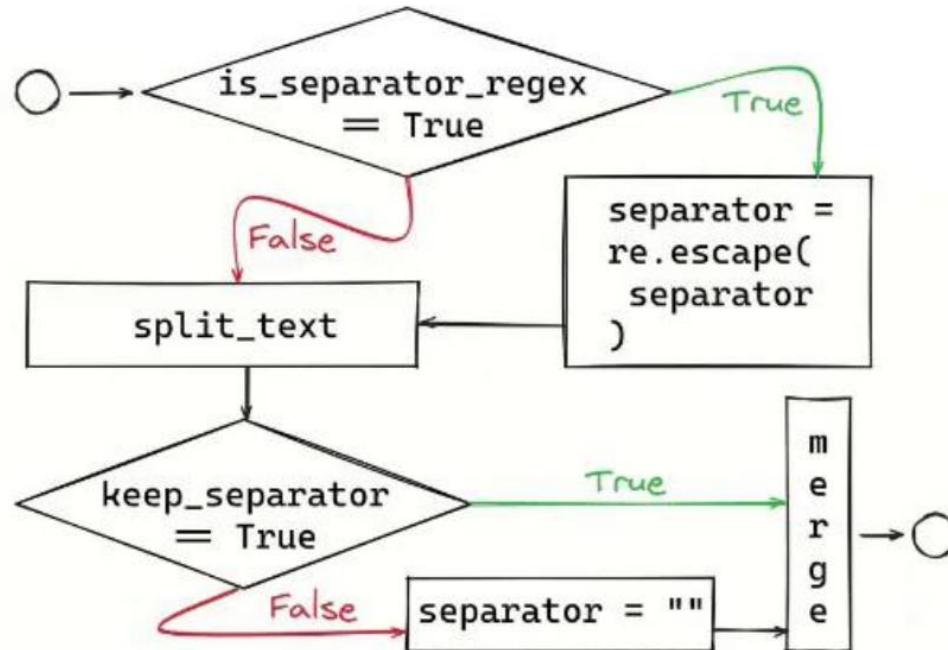


# Tools: Data Loaders

```
● ● ●  
1 # Example of loading and using document loaders  
2 def load_document_from_url(url: str, filename: str, loader_class: Any) -> Any:  
3     """Download a document from a URL, save it, and load it using a specified loader class.  
4  
5     Args:  
6         url: URL of the document to download.  
7         filename: Filename to save the downloaded document.  
8         loader_class: The class of loader to use for loading the document.  
9  
10    Returns:  
11        The loaded document.  
12    """  
13    download_and_save_file(url, filename)  
14    loader = loader_class(filename)  
15    return loader.load()  
16  
17 # Load and process documents  
18 text_document = load_document_from_url(text_url, "planets.txt", TextLoader)  
19 csv_document = load_document_from_url(csv_url, "insurance.csv", CSVLoader)  
20 pdf_document = load_document_from_url(pdf_url, "dubai_law.pdf", PyPDFLoader)
```

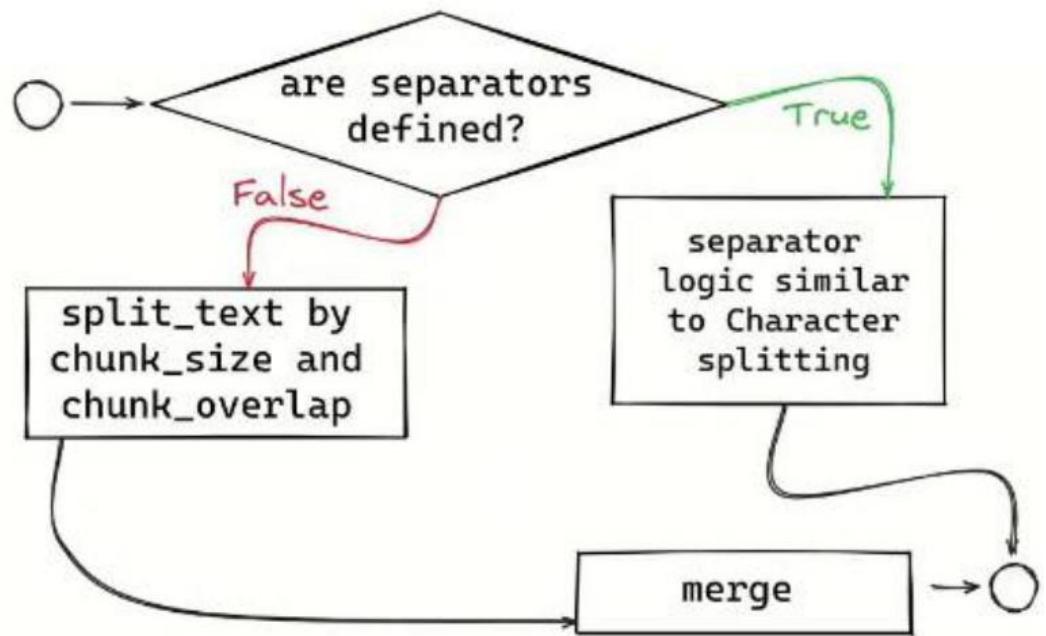
# Tools: Data Splitters

```
c_splitter = CharacterTextSplitter(  
    separator = ".",
    keep_separator = True
)
```



# Tools: Data Splitters

```
r_splitter = RecursiveCharacterTextSplitter(  
    chunk_size = 5  
    chunk_overlap = 2  
)
```



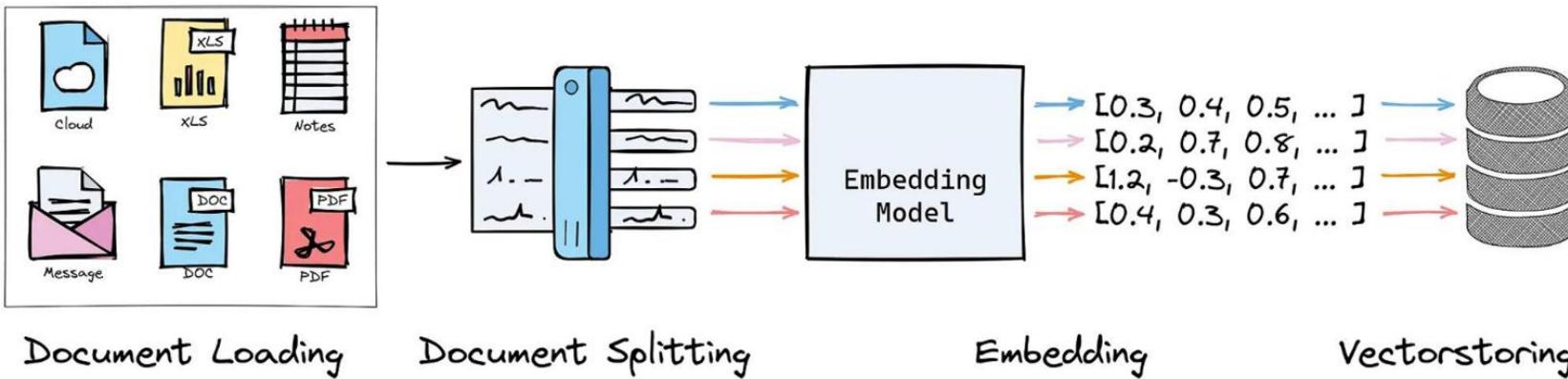
```
1 text_splitter = RecursiveCharacterTextSplitter(  
2     separators=[ "\n"], chunk_size=250, chunk_overlap=0, keep_separator=False  
3 )  
4 # split_text  
5 chunks = text_splitter.split_text(text_document[0].page_content)  
6  
7 #split_documents  
8 chunks = text_splitter.split_documents(text_document)
```

# Tools: Data Splitters

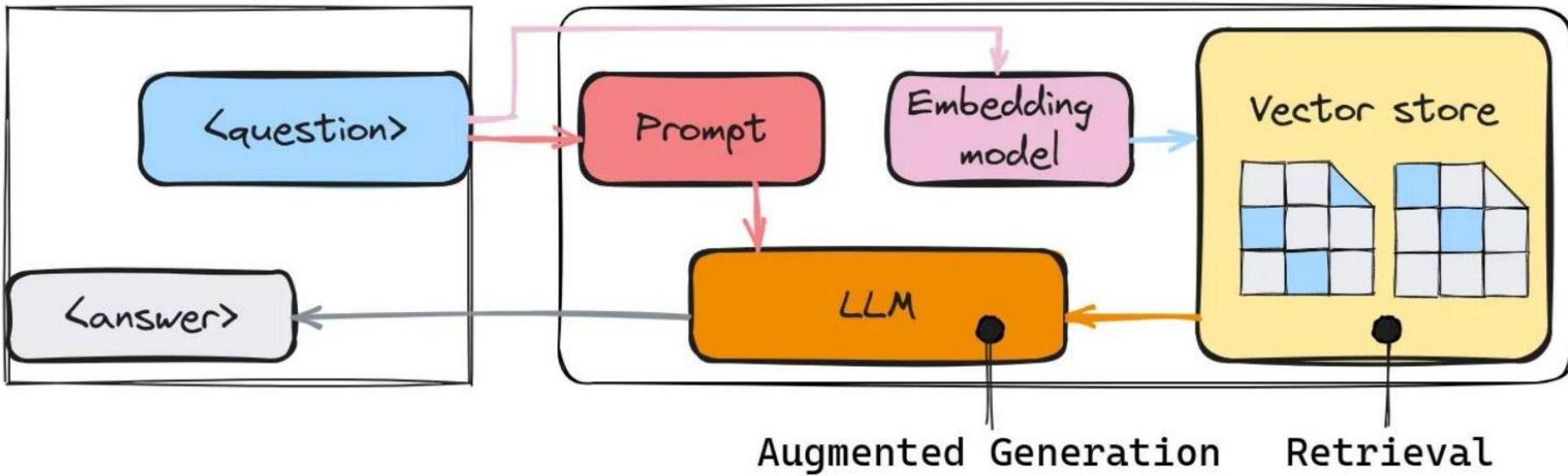
|                  |                                   |  |
|------------------|-----------------------------------|--|
| Recursive        | A list of user defined characters | Recursively splits text. Splitting text recursively serves the purpose of trying to keep related pieces of text next to each other. This is the recommended way to start splitting text. |
| HTML             | HTML tags                         | Splits text based on HTML tags. Notably, this adds in relevant information about where that chunk came from (based on the HTML)  |
| Markdown         | Markdown specific characters      | Splits text based on Markdown-specific characters. Notably, this adds in relevant information about where that chunk came from (based on the Markdown)                                   |
| Code             | Code specific characters.         | Splits text based on characters specific to coding languages. 15 different languages are available to choose from.   |
| Token            | Tokens                            | Splits text on tokens count with a specified method to measure tokens.   |
| Character        | A user defined character          | Splits text based on a user defined character. One of the simpler methods.   |
| Semantic Chunker | Sentences                         | First splits on sentences. Then combines ones next to each other if they are semantically similar enough   |

# Data Preprocessing Pipeline

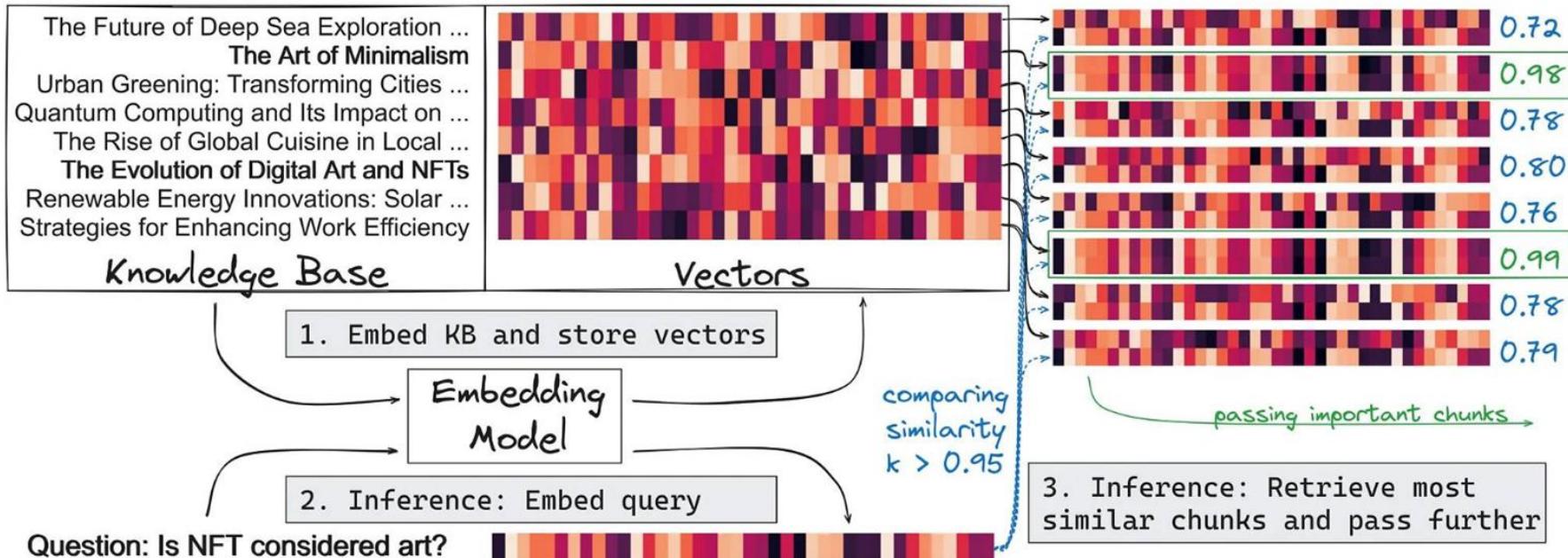
- 1) Load a document
- 2) Split the document
- 3) Embed the split chunks
- 4) Store the chunks



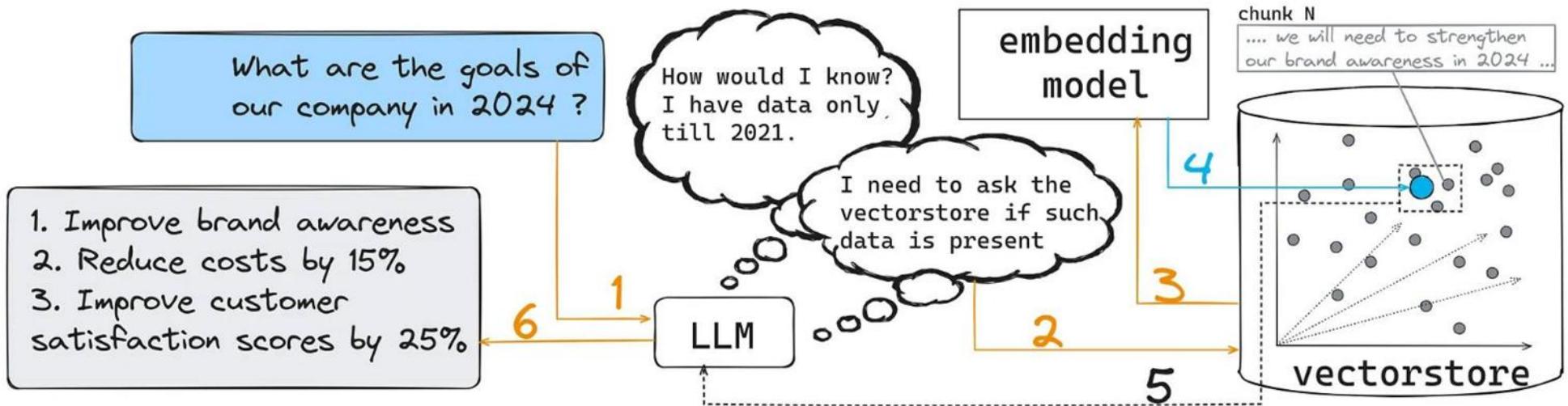
# Retrieval-Augmented Generation (RAG)



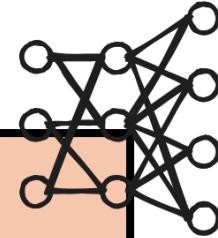
# RAG Pipeline



# RAG Pipeline



# Practice with LangChain



# **Finetuning Models**

# Why buzz with finetuning?

A person, that can **search** all doctors knowledge (vectorstore)



An person **acting** as a doctor (prompt)



A specialized **trained** doctor, (fine-tuned model)

# LLM Training Pipeline

Large language models like can go through a 3-step training process:

1. Pretraining (initial training or self-supervised learning)
2. Supervised finetuning
3. RLHF (human-supervised finetuning)

Initially the models learn from huge amounts of text without specific labels. This way we obtain pretrained models. In supervised finetuning, they get refined to follow particular instructions better. During the alignment stage, the models can be finetuned even further to respond in a more helpful and safe way to user's input.

# Pretraining

During pretraining, the model learns from a huge collection of text containing billions to trillions of words. For every phase, it's given a sentence and is tasked with predicting the next word or token that should come after it.

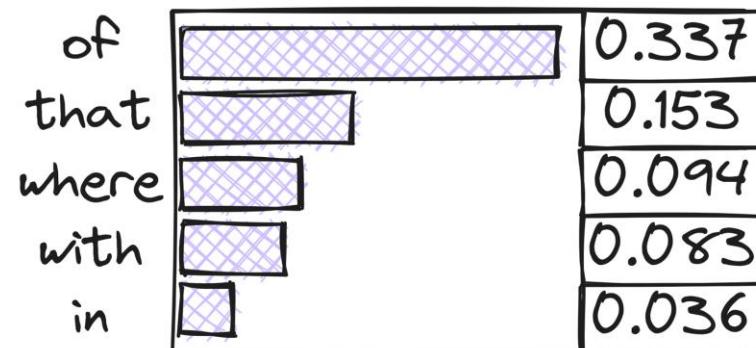
Dataset:  
**100B-10T tokens**

Task:  
**Predict next token on unlabeled texts**

Output:  
base “pretrained” model

|       |    |     |      |   |
|-------|----|-----|------|---|
| Paris | is | the | city | ? |
| 1652  | 5  | 42  | 171  | ? |

token ids



# .from\_pretrained(<model>)

In many pretrained language models, the tokenizer and the model architecture are designed and trained together. The reason for this coupling is that the tokenizer (“token producer”) and the model architecture need to be compatible with each other to ensure consistent tokenization and text generation.

If the tokenizer and the model architecture were different or not synchronized, it could lead to errors, mismatched embeddings, and incorrect predictions.



```
1 tokenizer = AutoTokenizer.from_pretrained("some_model")
2 model = BloomForCausalLM.from_pretrained("some_model")
```

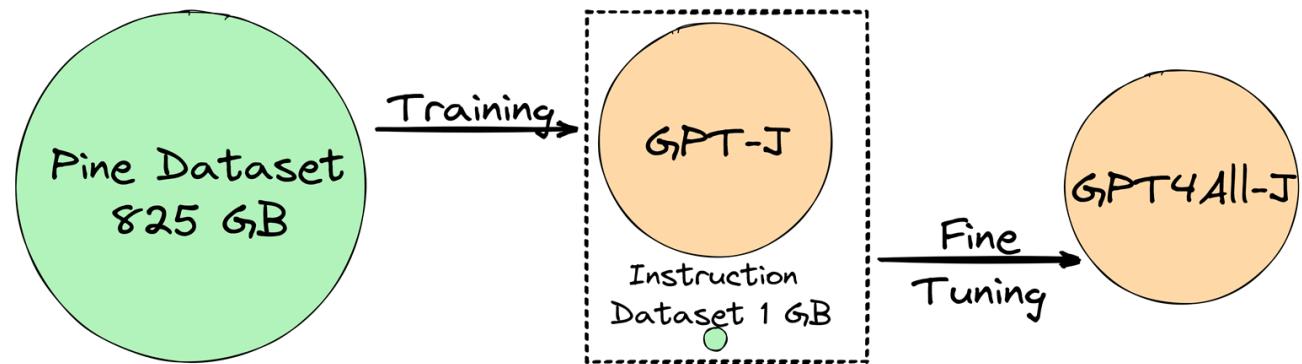
# Supervised finetuning

In the supervised finetuning stage, we continue predicting the next token, but now we use pairs of instructions and their corresponding desired outputs. The instruction is what we give the model as input (sometimes along with optional text), and the output is the response we want the model to generate, similar to our expectations.

Dataset:  
**10K-100K tokens**

Task:  
**Predict next token  
on unlabeled texts**

Output:  
**finetuned model**



# Supervised finetuning

**text (string)**

"I've learned the nitrogen vacancies used in  
Memristors are for "switching", between excited...

"Volunteer Services Volunteer Services As  
Charleston Area Medical Center volunteers, our...

"Q: Python: My return variable is always None So I  
found a strange thing that happens in python...



**prompt (string)**

"<p>In Angular if I had a few panels, similar to  
tabs how do you open the panel on click and animat...

"Given the following scientific paper: This paper  
presents a novel approach for temporal and semanti...

"<p>i am using the following code:</p> <pre>  
<code>package Presentacion; import...

**response (string)**

"To accomplish this in Angular, you can use  
ngAnimate and ng-show/ng-hide directives. Here's...

"This scientific paper describes three different  
approaches for segmenting and analyzing edited...

"The error is caused by a typo in the code. The  
correct class name is "PrintWriter" and not...

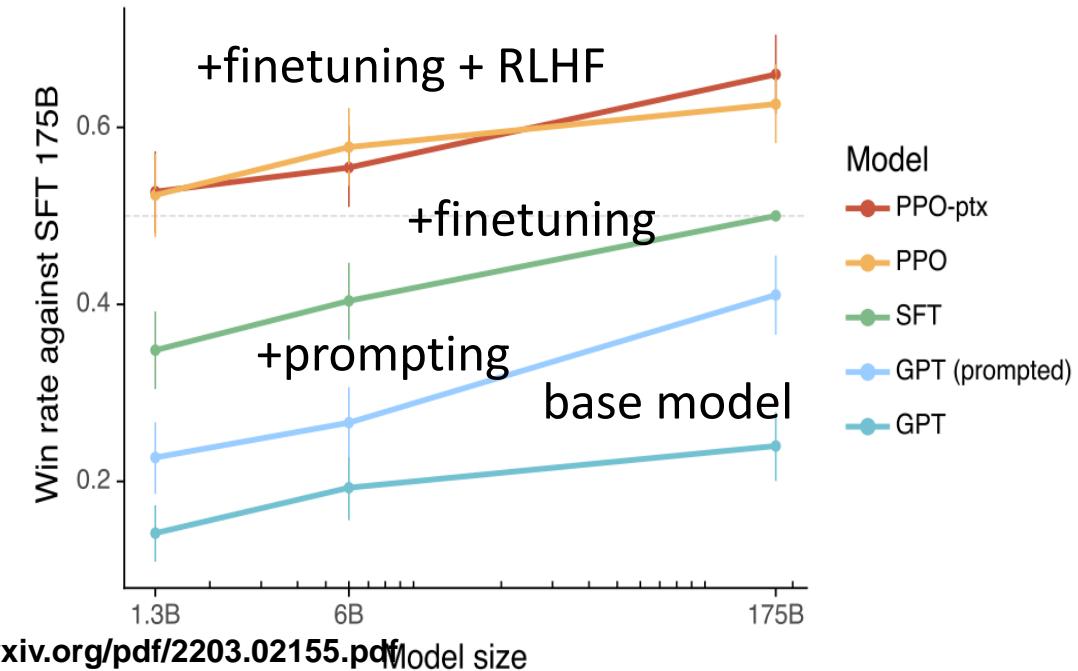
# RLHF Alignment

RLHF (Reinforcement Learning from Human Feedback) is an important component of the current method used to train advanced language models. It helps include people's feedback when finetuning models, which ultimately makes the model more useful and secure.

Dataset:  
**50K-100K tokens**

Task:  
**Align with user (human)  
preferences**

Output:  
**human feedback-tuned model**

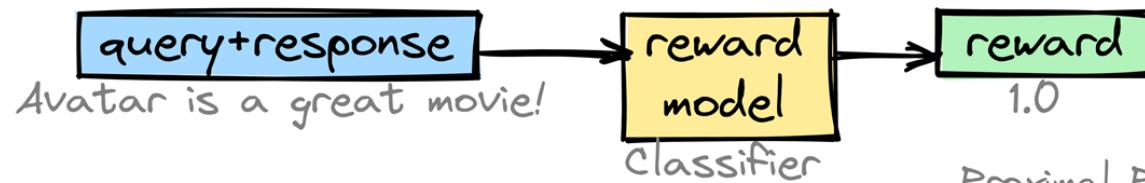


# Finetuning Models

## Rollout

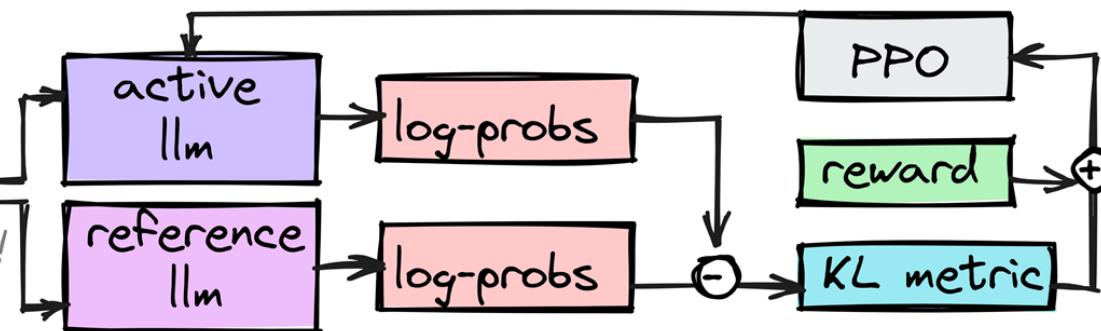


## Evaluation

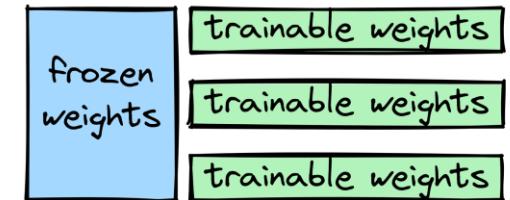


## Optimization

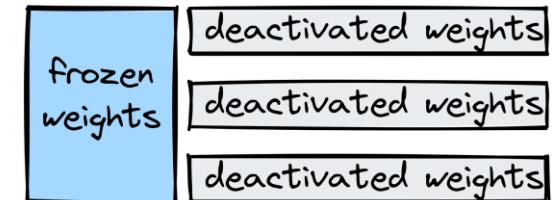
query+response  
Avatar is a great movie!



## Active llm



## Reference llm



# (Instruct)GPT RLHF pipeline

The Instruct(GPT) RLHF pipeline involves taking a pretrained model and refining it through supervised training (similar to step 2 “Supervised finetuning” in the traditional training pipeline). Afterwards, the updated model is further refined using proximal policy optimization.

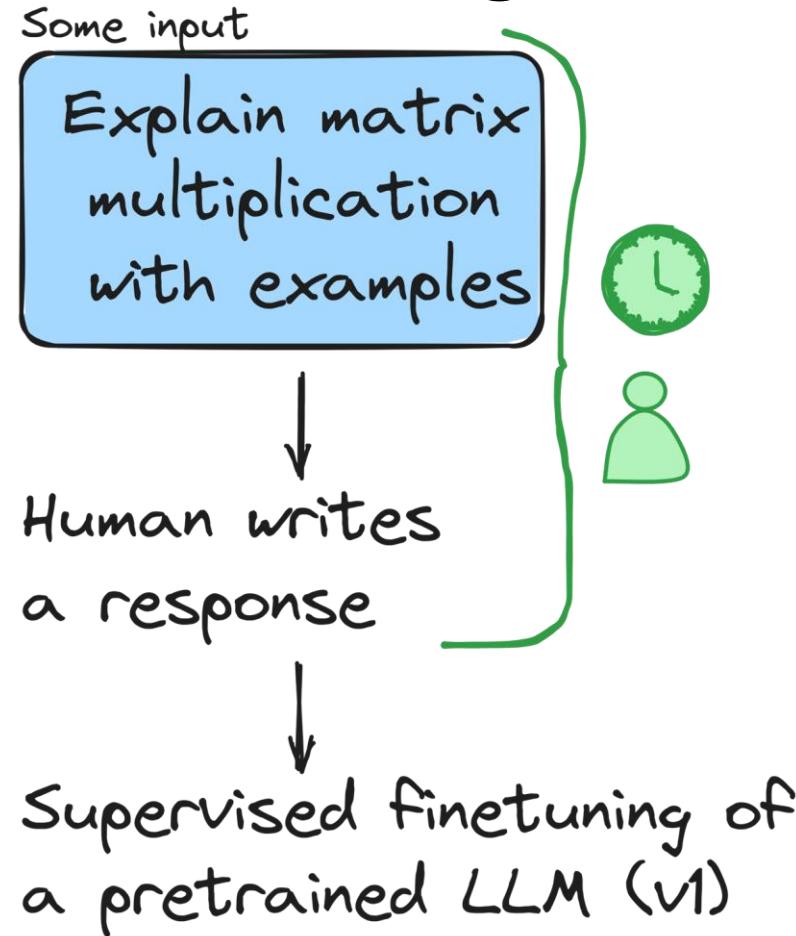
The RLHF pipeline is a 3-step training process:

1. Refined training of the pretrained model through supervision
2. Development of a model for providing rewards
3. Additional refinement using proximal policy optimization (PPO)

# RLHF pipeline: Supervised finetuning

In the first step of RLHF pipeline, we either generate or select prompts (potentially from a dataset or database) and request humans to produce high-quality responses.

We utilize this collection of data to finetune the pre-existing base model in a guided manner.



# RLHF pipeline: Training reward LLM

In RLHF pipeline step 2, we utilize the finetuned model via supervised training to construct a reward model for the next step. This involves generating multiple responses for each prompt and having individuals rank them according to preference.

To transform the model from RLHF pipeline step 1 to a reward model, we replace its output layer (the next-token layer) with a regression layer that has a single output node.

Some input

Explain matrix multiplication with examples

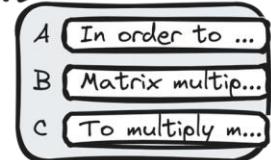
↓  
Finetuned LLM (v1)

writes responses

↓  
Human ranks LLM

(v1) responses

C > A > B

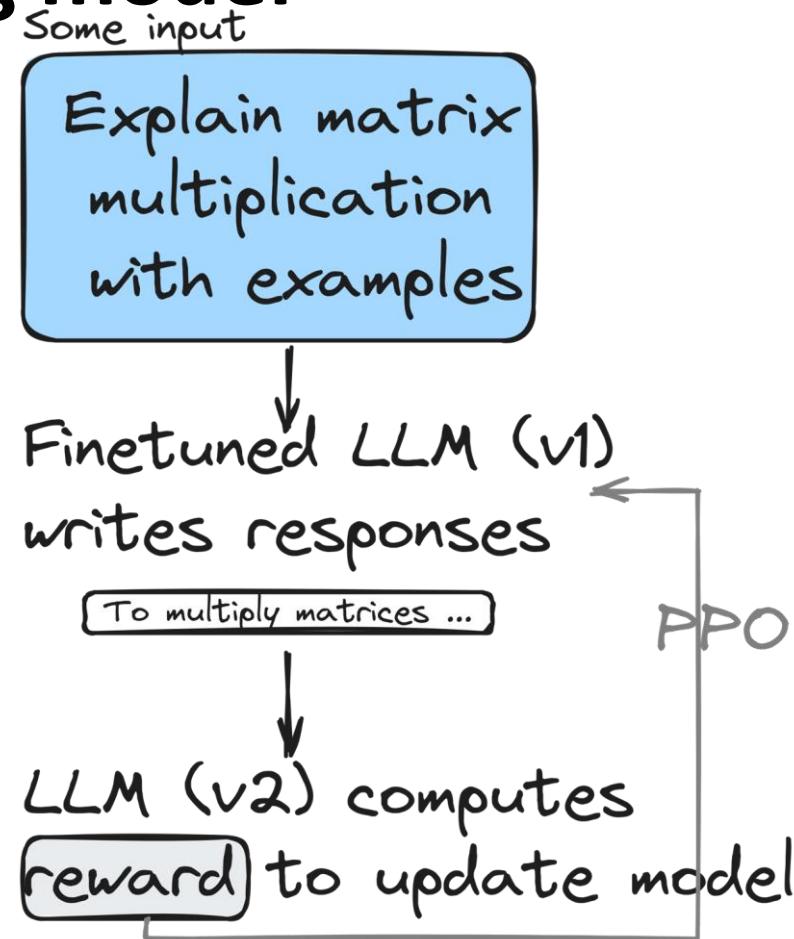


↓  
Train reward LLM (v2)

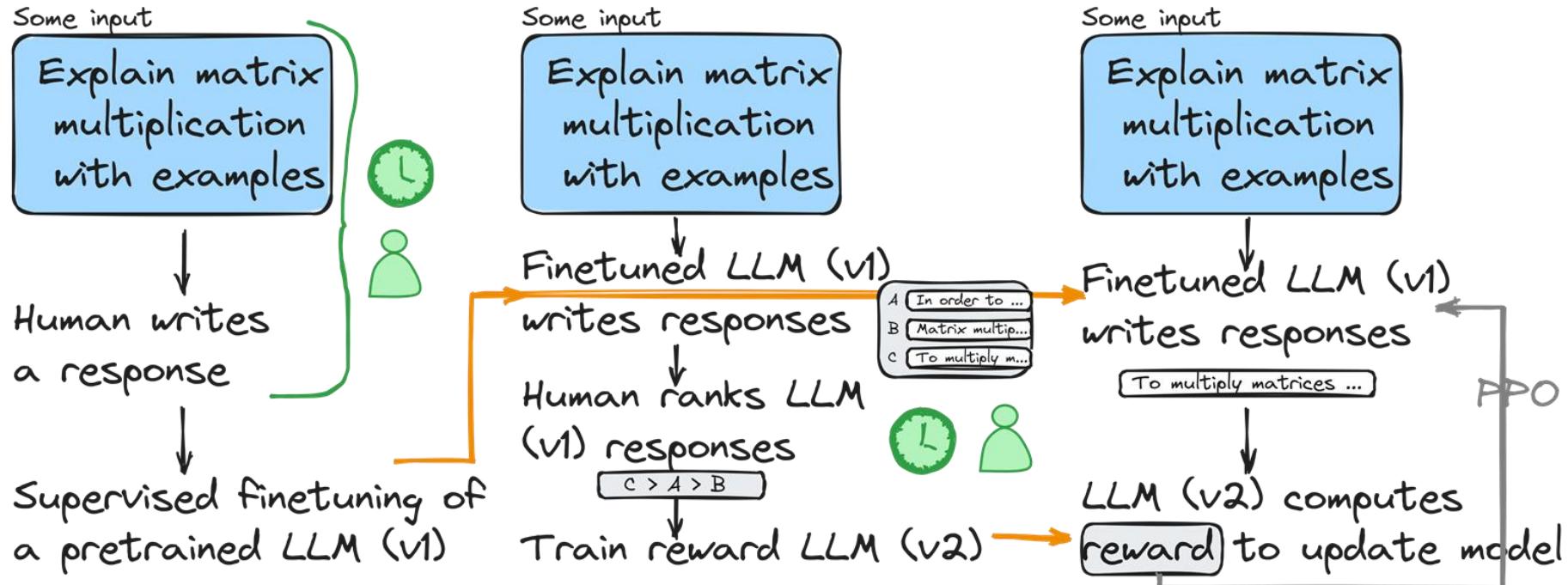
# RLHF pipeline: Updating model

In the 3<sup>rd</sup> and last step of the RLHF pipeline, we employ the reward model (v2) to further finetune the previous model that underwent supervised finetuning (v1).

We adjust the v1 model using proximal policy optimization (PPO) guided by the reward scores obtained from the reward model we established in RLHF pipeline step 2.



# (Instruct)GPT RLHF pipeline

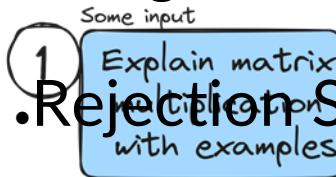


# LLAMA RLHF pipeline

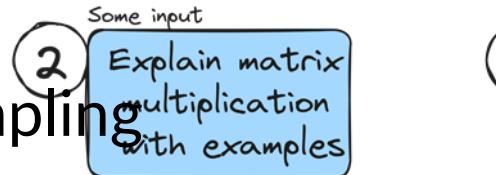
The Meta AI Llama 2 model, while using a similar RLHF approach to InstructGPT, introduces several noteworthy differences:

- Two Reward Models

- Margin Loss



Supervised finetuning of a pretrained LLM (v1)

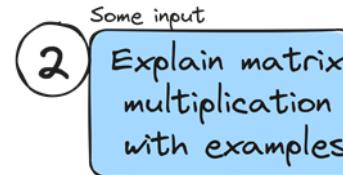


Human assigns a label

A In order to ...  
B Matrix multip...  
C To multiply min...  
D DATA SCIENCE...

much better  
better  
bit better  
min better

Train 1st reward LLM (v2 - helpfulness)

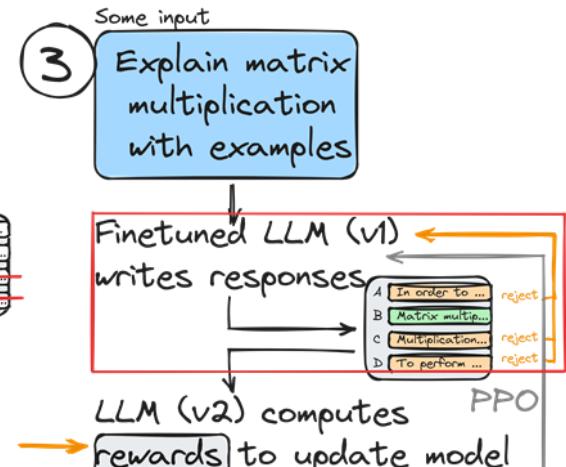


Human assigns a label

A In order to ...  
B Matrix multip...  
C To multiply min...  
D DATA SCIENCE...

much better  
better  
bit better  
min better

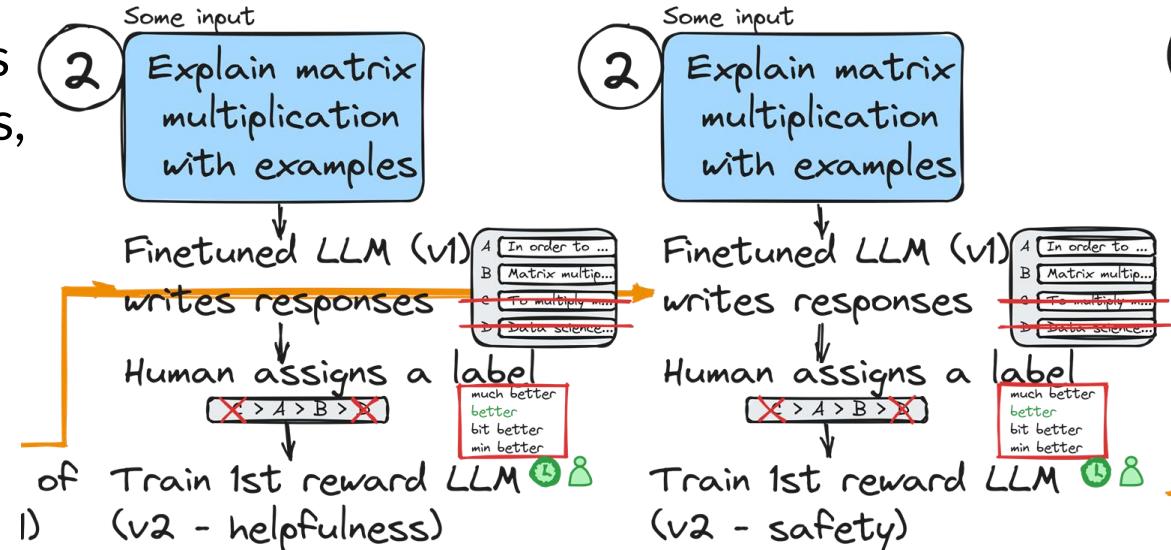
Train 1st reward LLM (v2 - safety)



# LLAMA RLHF pipeline

Two Reward Models: Llama2 employs two reward models, one focused on helpfulness and the other on safety. The final optimization of the model is based on a combination of these two scores.

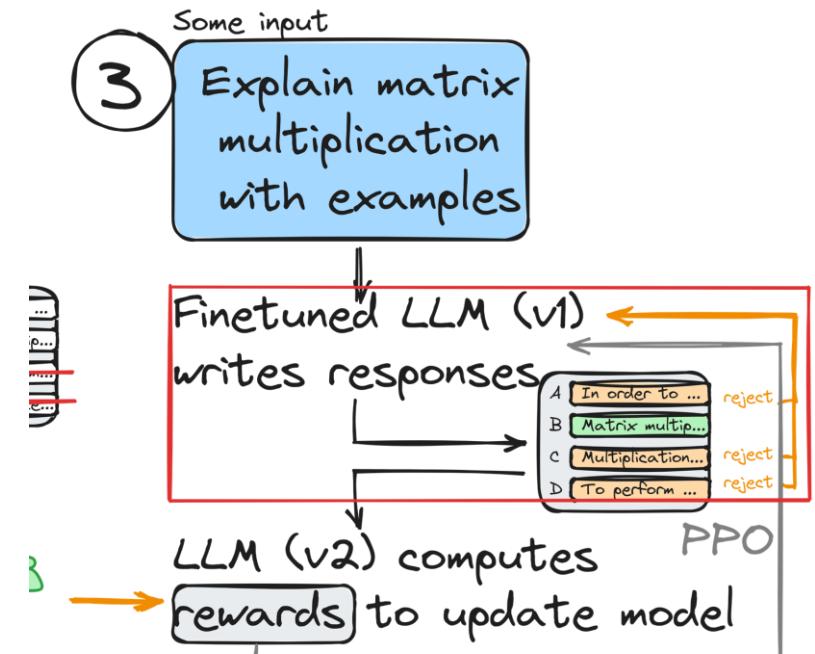
Margin Loss: In the process of ranking model responses, Llama2 introduces a "margin" label, which measures the gap between preferences. This margin parameter helps refine the ranking loss calculation.



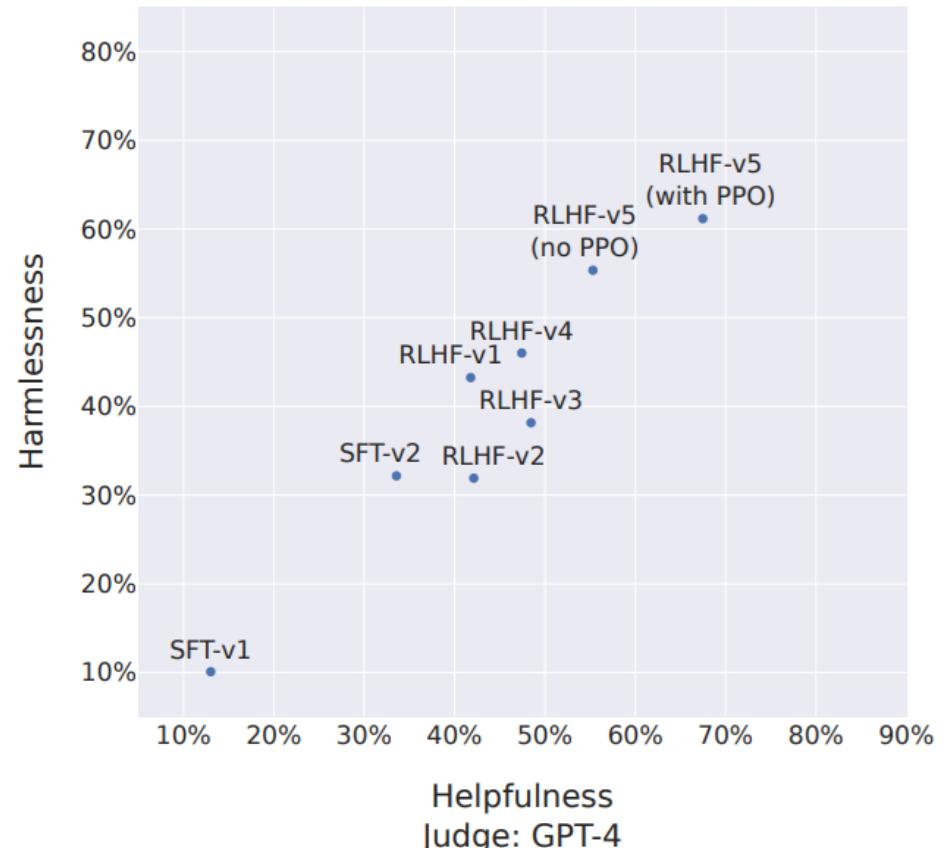
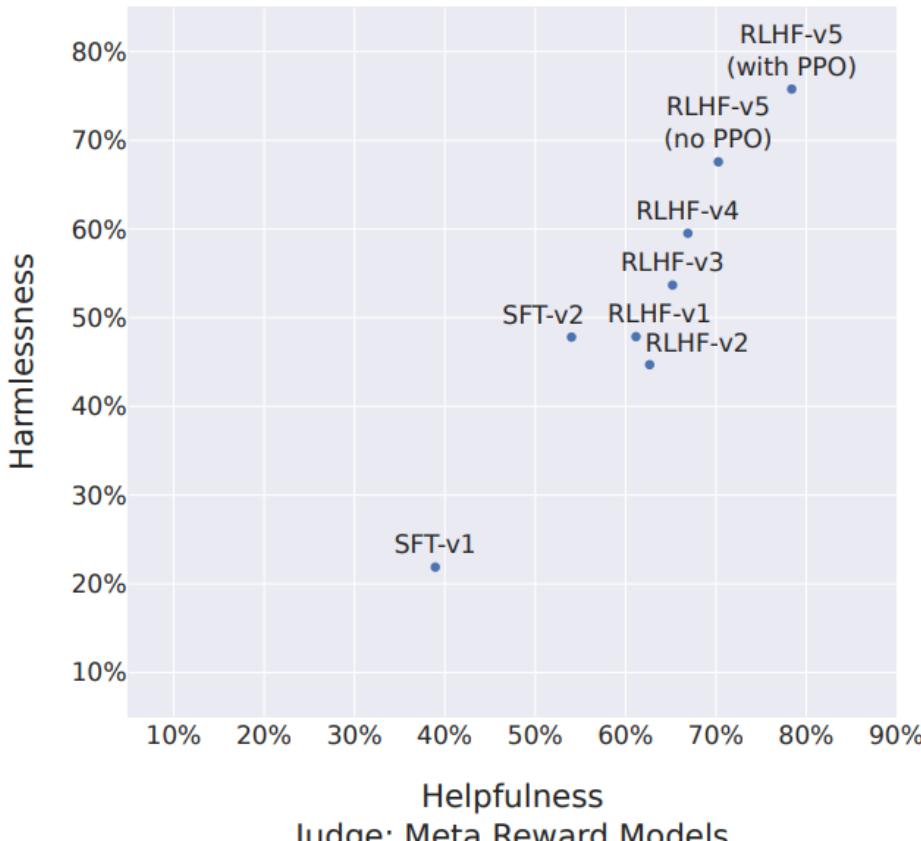
# LLAMA RLHF pipeline

Rejection Sampling: Llama2 uses an iterative RLHF approach, creating multiple versions of the model (from RLHF-V1 to RLHF-V5). In addition to PPO, they employ rejection sampling.

This technique involves generating multiple outputs and selecting the one with the highest reward for gradient updates during optimization. It's different from PPO, which updates based on a single sample.



# RLHF Pipelines Comparison



# RLHF Alternatives

## Constitutional AI: Harmlessness from AI Feedback

Researchers introduce a self-training method based on a set of rules provided by humans.

## The Wisdom of Hindsight Makes Language Models Better Instruction Followers

The study introduces HIR (Hindsight Instruction Labeling), a two-step method involving prompt sampling and training, which effectively converts cases where the Language Model deviates from instructions

## RLAIF: Scaling Reinforcement Learning from Human Feedback with AI Feedback

The study on RLAIF demonstrates that ratings used for reward model training in RLHF can be generated by an LLM rather than solely relying on human input.

...

# Finetuning: Modern way

Which response do you prefer?

Your choice will help make ChatGPT better.



Response 1

If you're seeing that headers are not loaded correctly after skipping 2 rows, it's possible that the third row (which you intend to use as headers) contains values that aren't suitable as column names.

You can address this in a few ways:

## 1. Explicitly Mention the Header Row:

If the headers are on the third row of the file, then they would be at index `2` (since indices are 0-based). You can tell `pd.read\_csv()` to use that row as the header:

python

Copy code

```
df = pd.read_csv(selected_file_path, delimiter='; ',
```



Response 2

If, after skipping 2 rows, the header rows aren't being read correctly, it probably means that the third row (index 2) in your file is the actual header, but pandas doesn't know that yet.

To tell pandas to use the third row (index 2) as the header after skipping the first 2 rows, you'll need to set the `header` parameter:

python

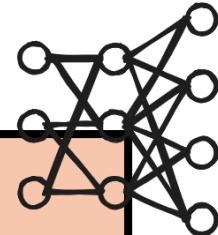
Copy code

```
df = pd.read_csv(selected_file_path, delimiter='; ',
```

Here:

- `skiprows=2` will skip the first two rows.
- `header=0` tells pandas to use the first row it encounters

## 2. Read Without Headers and Rename Later:



# Practice with Finetuning