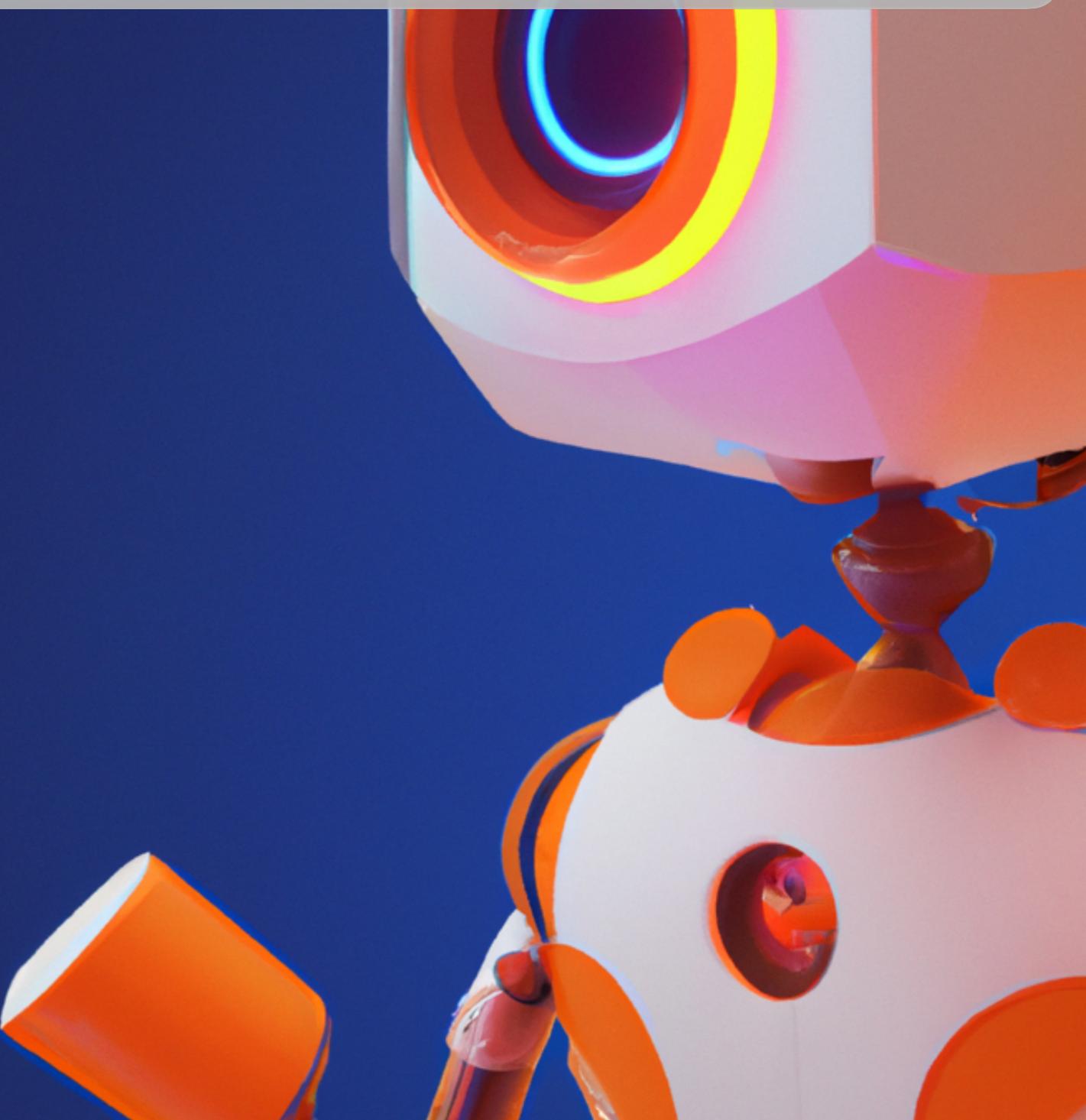




Generative Artificial Intelligence with OpenAI

Bruno Gonçalves
graphs4sci.substack.com

<https://github.com/DataForScience/OpenAI>



Question

<https://github.com/DataForScience/G4DS>

- What's your job title?

- Data Scientist
- Statistician
- Data Engineer
- Researcher
- Business Analyst
- Software Engineer
- Other

Question

<https://github.com/DataForScience/G4DS>

- How experienced are you in Python?

- Beginner (<1 year)
- Intermediate (1 -5 years)
- Expert (5+ years)

Question

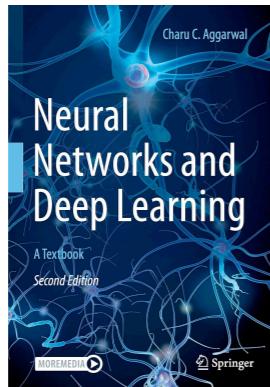
<https://github.com/DataForScience/G4DS>

- How did you hear about this webinar?

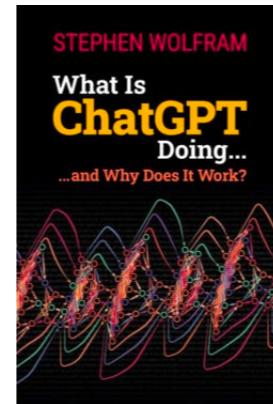
- O'Reilly Platform
- Newsletter
- data4sci.com Website
- Previous event
- Other?

References

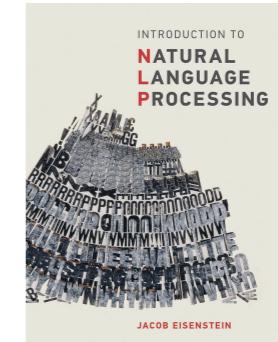
<https://github.com/DataForScience/OpenAI>



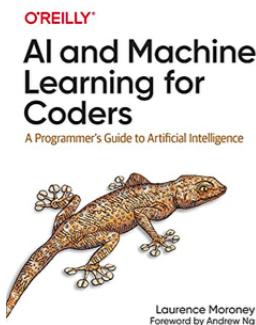
<https://amzn.to/48rZn9X>



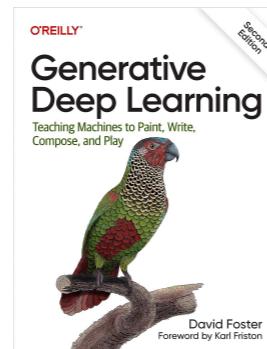
<https://amzn.to/3LBRdBY>



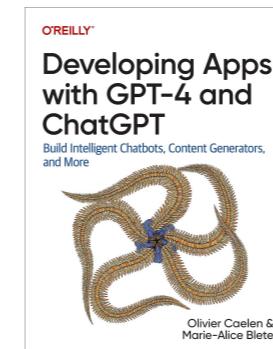
<https://amzn.to/3ZMnTih>



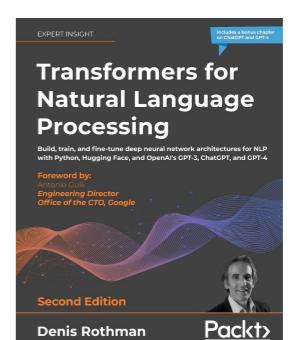
<https://amzn.to/3KjB5W4>



<https://amzn.to/3t8PuxM>



<https://amzn.to/3RHRkQa>



<https://amzn.to/46kOo02>

Table of Contents



1. Generative AI and OpenAI
2. GPT Models
3. Embeddings
4. Image Generation
5. Code Generation and Explanation



1. Generative AI and OpenAI

Basic Principles

- **Generative Artificial Intelligence** is a class of ML models that can generate output similar to the data it was trained on. In particular:
 - **Generative Adversarial Networks (GAN)** -
 - **Variational Autoencoders (VAE)** - Represent the input data into a latent space
 - **Recurrent Neural Networks (RNN)** - Generate an output sequence based on a sequence of inputs
 - **Transformer-Based Models** - Encoder/Decoder based architecture
- Wide range of applications:
 - Text Generation
 - Image Generation
 - Video Synthesis
 - Data Augmentation
 - etc

Language Models

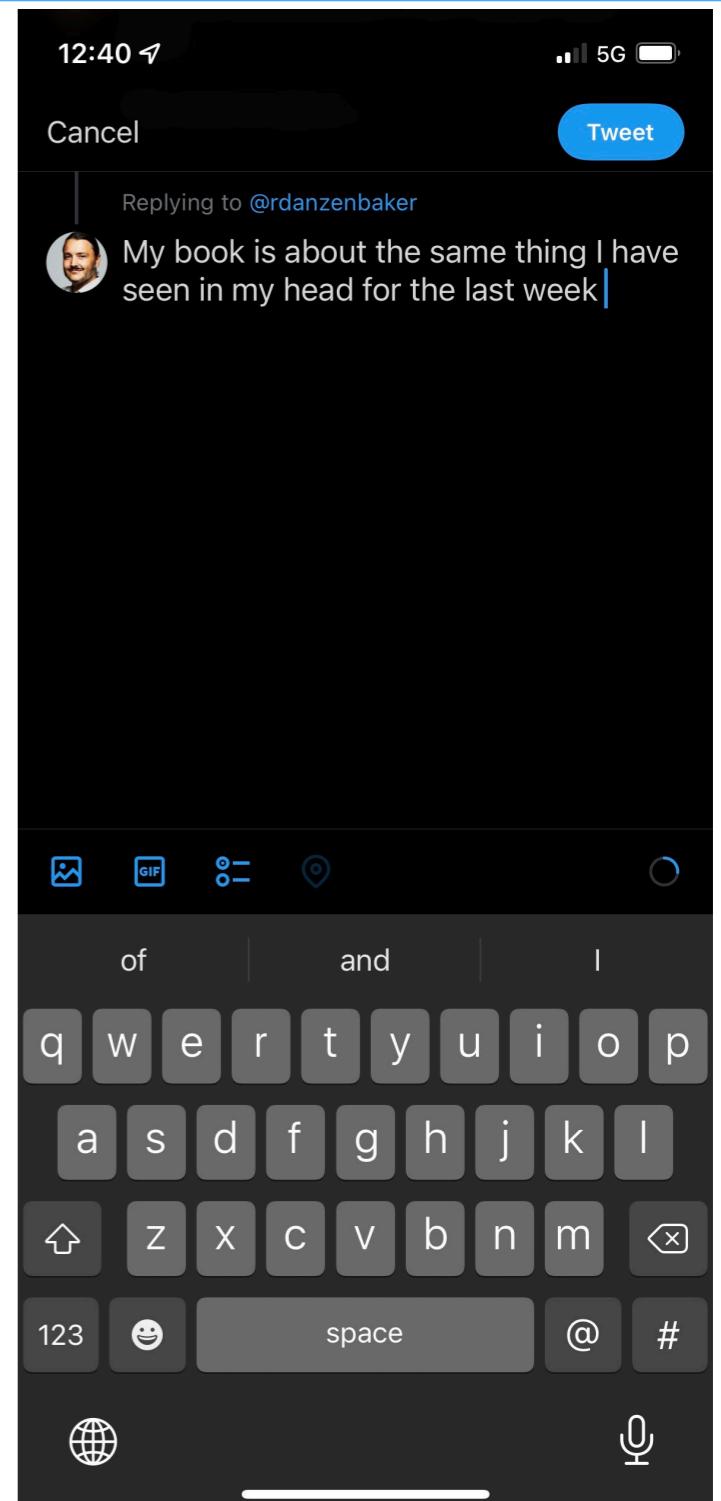
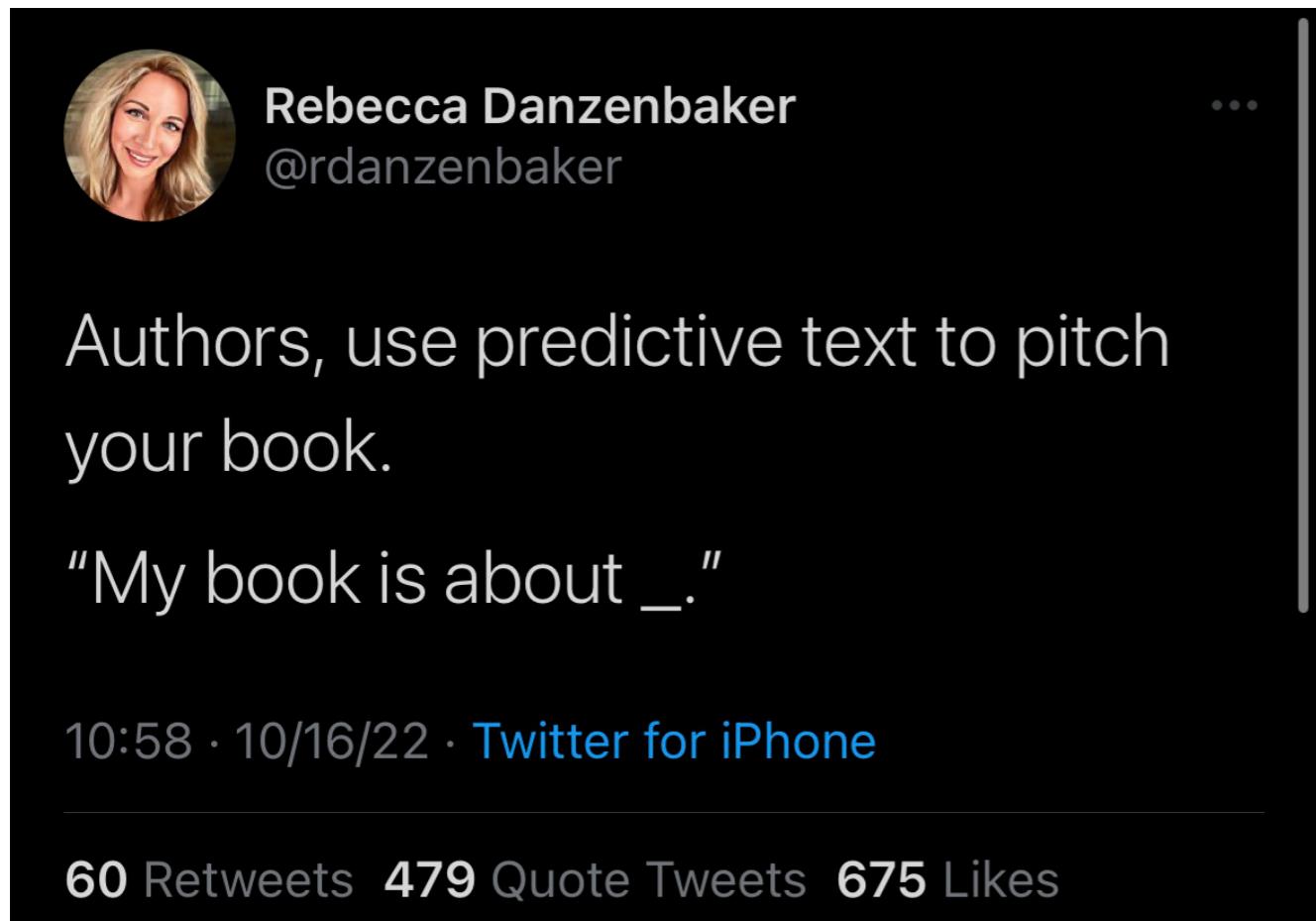
- A foundational component of Natural Language Processing and AI.
- Software that predicts and generates human language based on patterns observed in training data.
- Applications to:
 - Machine translation
 - Summarization
 - Sentiment analysis
 - Chatbots and Virtual Assistants
 - Content Generation
 - etc...

Language Models

- If you use the predictive text feature in your digital devices, you're already familiar with how language models work.
- It's just an endless game of "[guess the next word](#)" ...

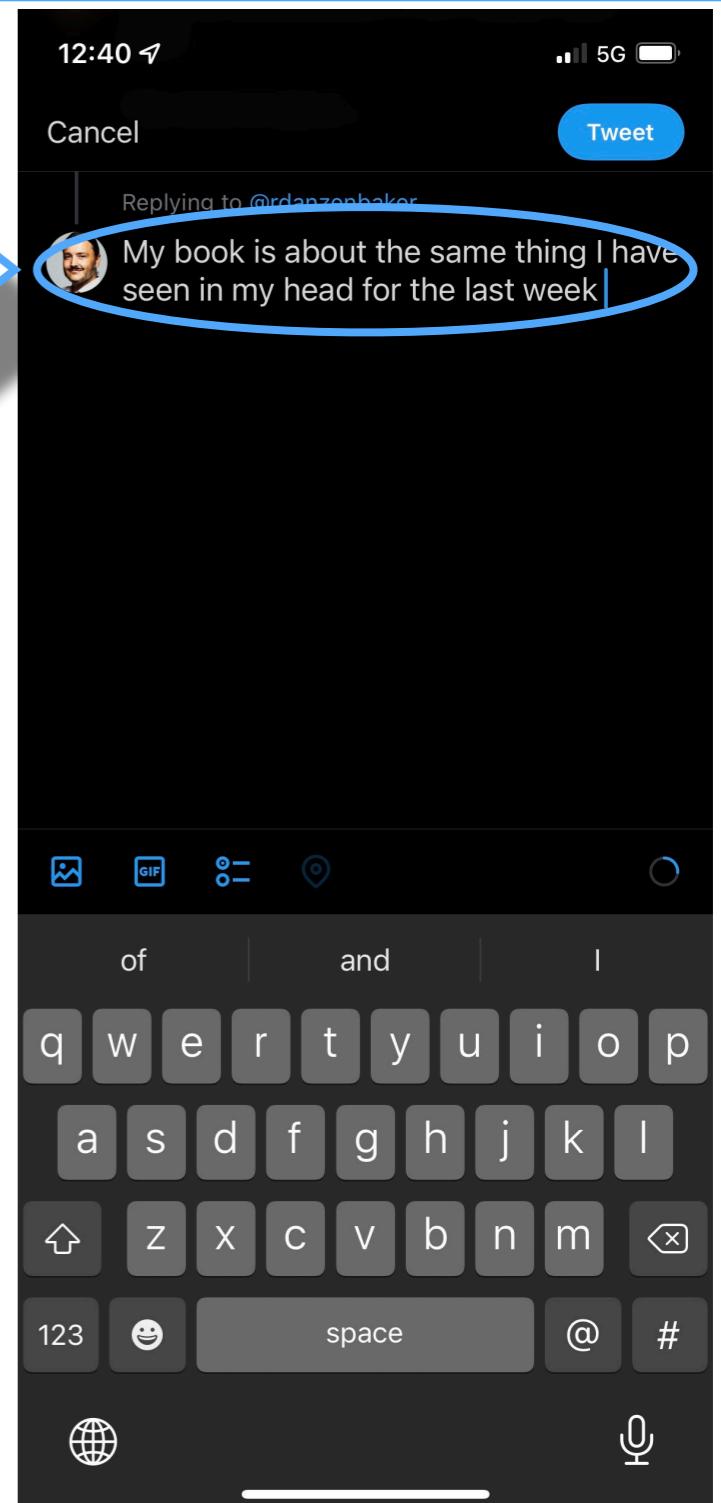
Language Models

- If you use the predictive text feature in your digital devices, you're already familiar with how language models work.
- It's just an endless game of "guess the next word" ...



Language Models

- If you use the predictive text feature in your digital devices, you're already familiar with how language models work.
- It's just an endless game of "guess the next word".

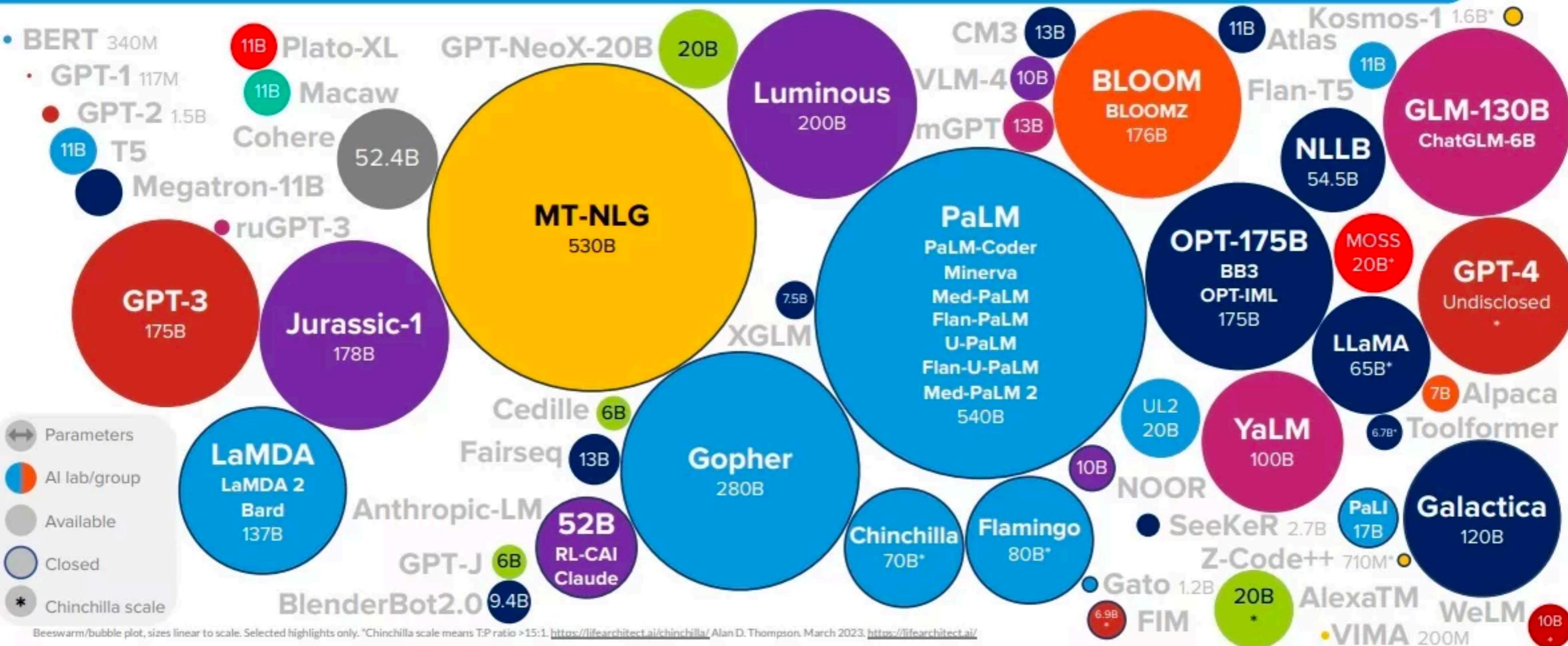


Large Language Models

<https://pureinsights.com/blog/2023/what-are-large-language-models-langs-search-and-ai-perspectives/>

- LLMs are... **LARGE!**

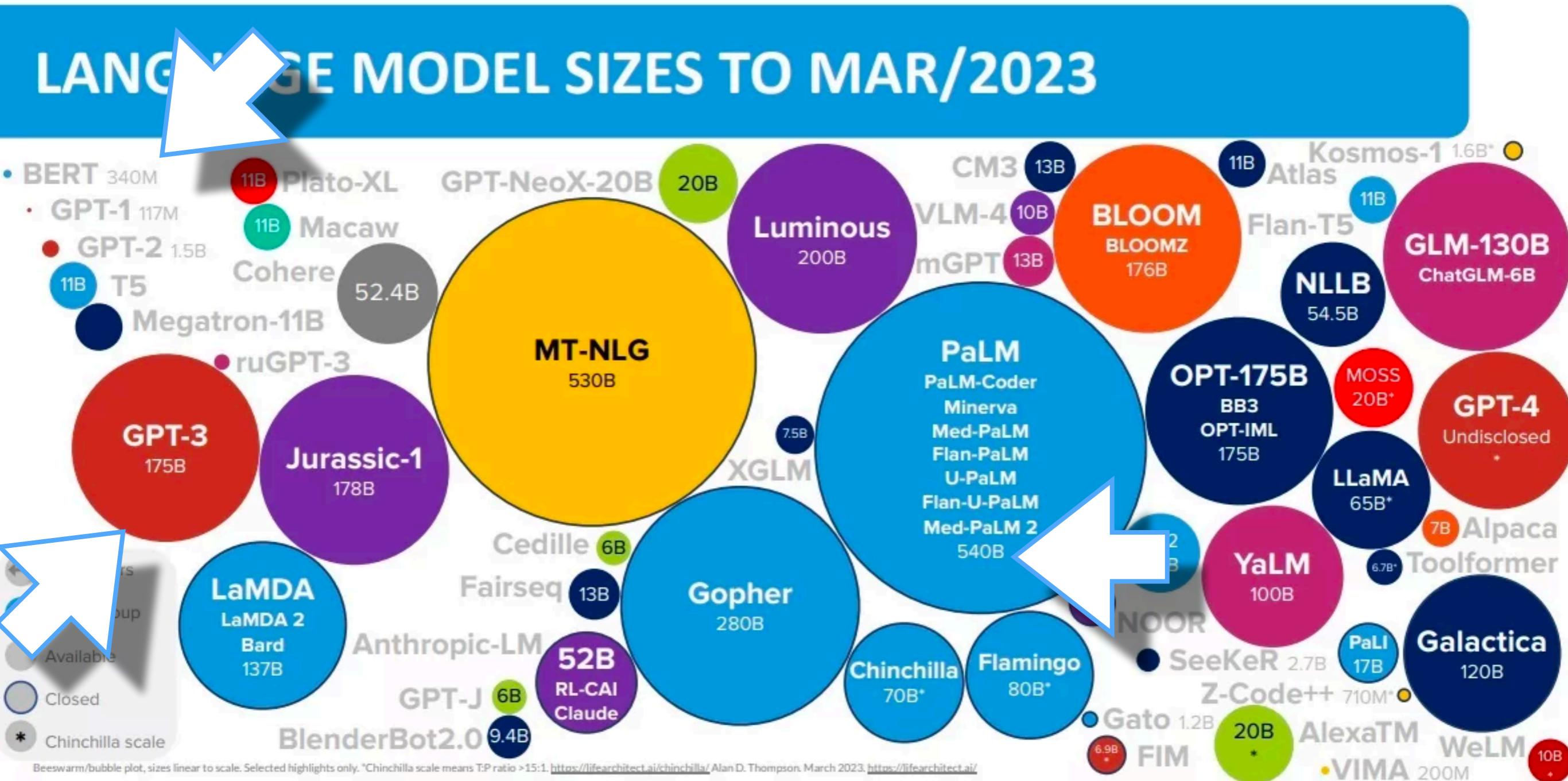
LANGUAGE MODEL SIZES TO MAR/2023



Large Language Models

<https://pureinsights.com/blog/2023/what-are-large-language-models-langs-search-and-ai-perspectives/>

- LLMs are... **LARGE!**

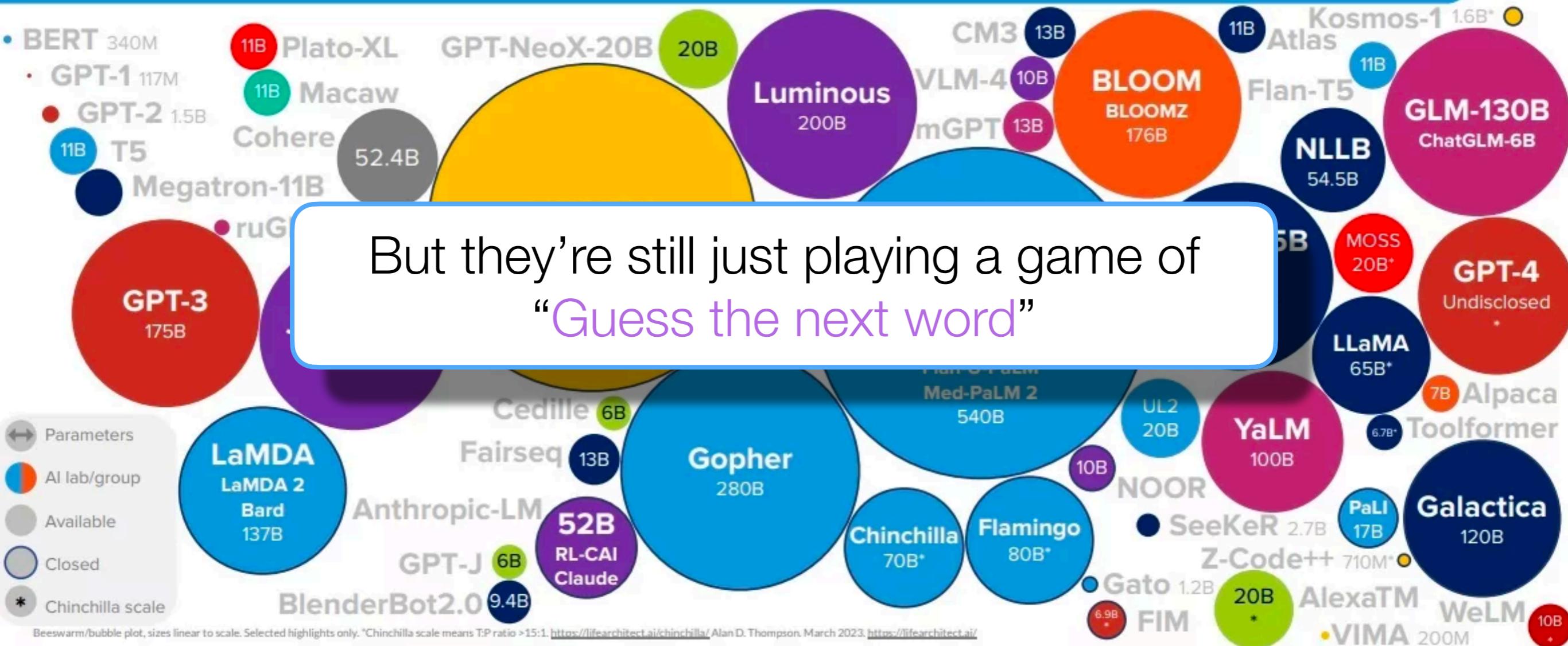


Large Language Models

<https://pureinsights.com/blog/2023/what-are-large-language-models-langs-search-and-ai-perspectives/>

- LLMs are... **LARGE!**

LANGUAGE MODEL SIZES TO MAR/2023



Transformers

- More than meets the eye!



Transformers

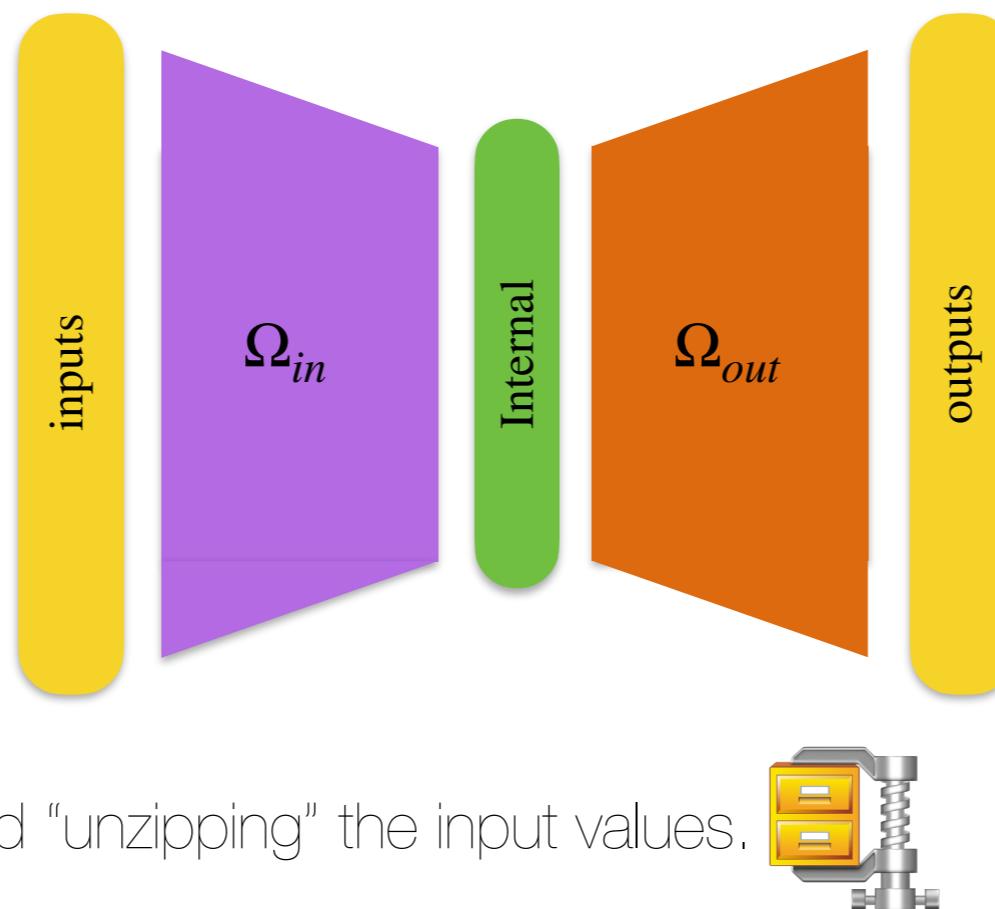
arXiv:1706.03762

- More than meets the eye!
- “[Attention is all you need](#)” (2017), by A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin
- Foundational model for current state-of-the-art models in NLP, CV, Reinforcement Learning, Speech, etc...
- Overcomes limitations of previous SOTA models, like Recurrent Neural Networks and Convolutional Neural Networks
- Uses [Encoder/Decoder](#) architecture with [Attention](#) mechanisms to capture context.



Encoder-Decoder

- **Auto-Encoders** use the same values for both inputs and outputs
- The Internal/hidden layer(s) have a smaller number of units than the input
- The fundamental idea is that the Network needs to learn an **internal representation** of its **inputs** that is smaller but from which it is still possible to reconstruct the input.

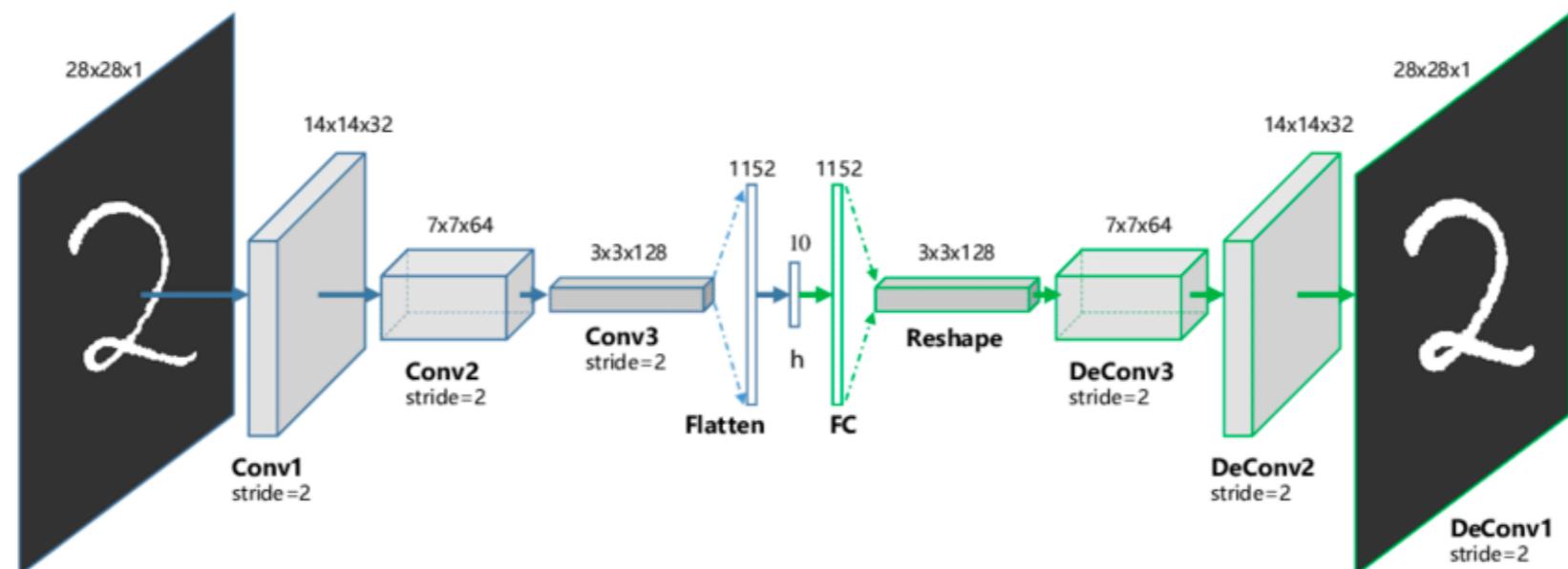


- Think of it as “zipping” and “unzipping” the input values.

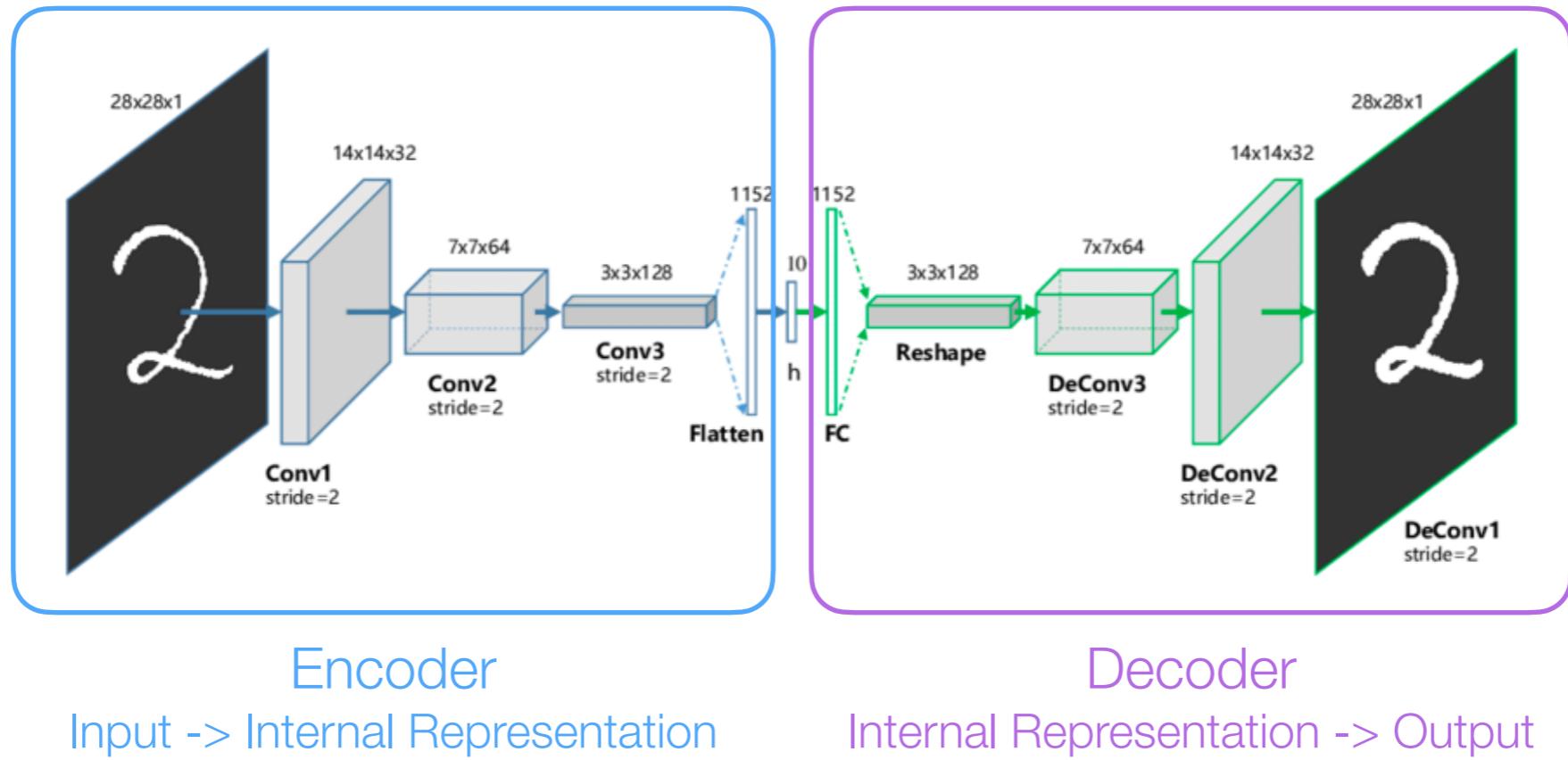
Auto-Encoders

https://www.researchgate.net/figure/The-structure-of-proposed-Convolutional-AutoEncoders-CAE-for-MNIST-In-the-middle-there_fig1_320658590

- After training, the parts of the network that generate the internal representation can be used as inputs to the Networks
- This is similar to what we did when we reused the word embeddings generated by training a word2vec network
- Auto-encoders can be arbitrarily complex, including many layers between the input and the internal representation (or Code) and are often used in Image Processing to generate efficient representations of complex images

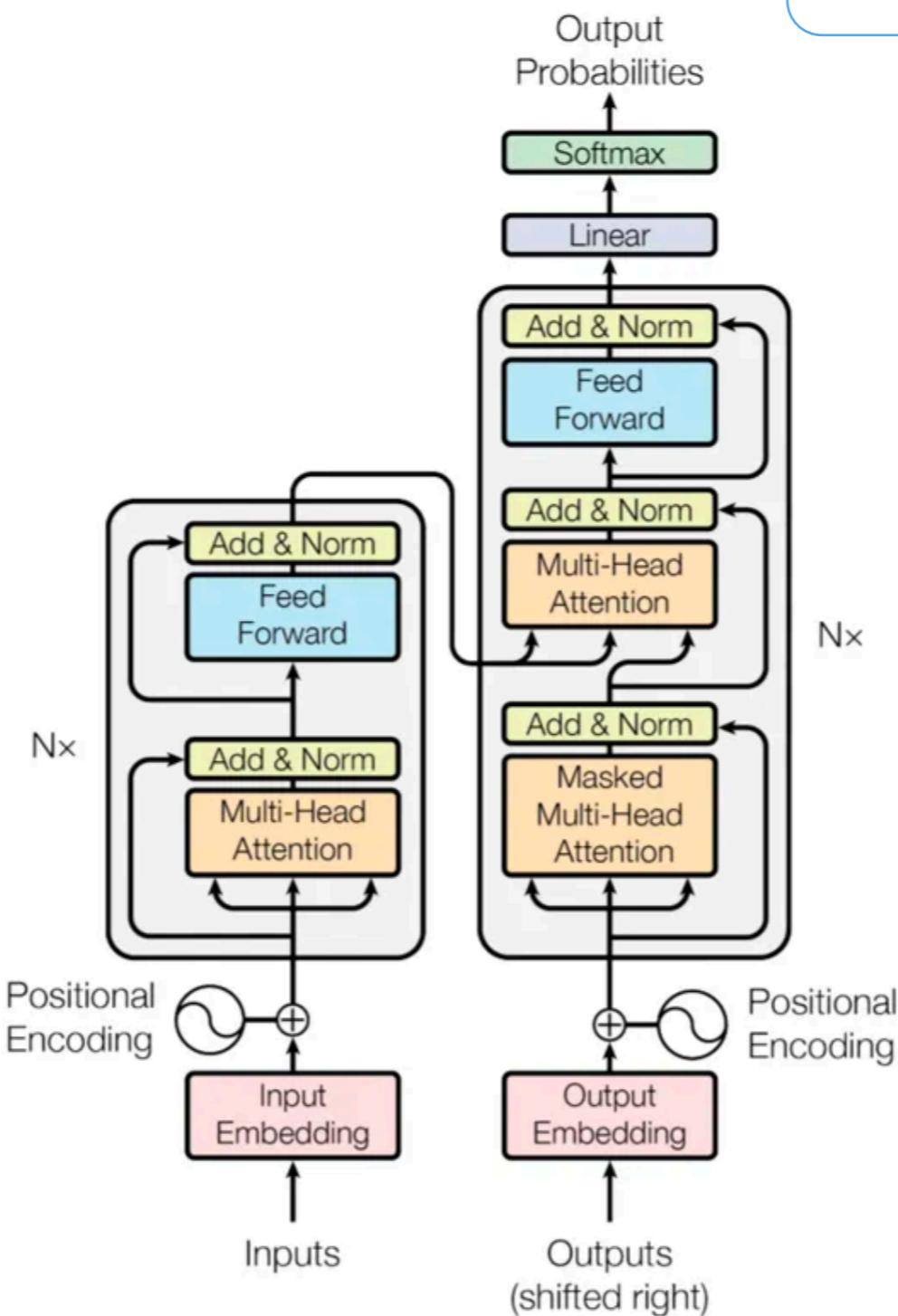


Encoder/Decoder Architecture



Transformers

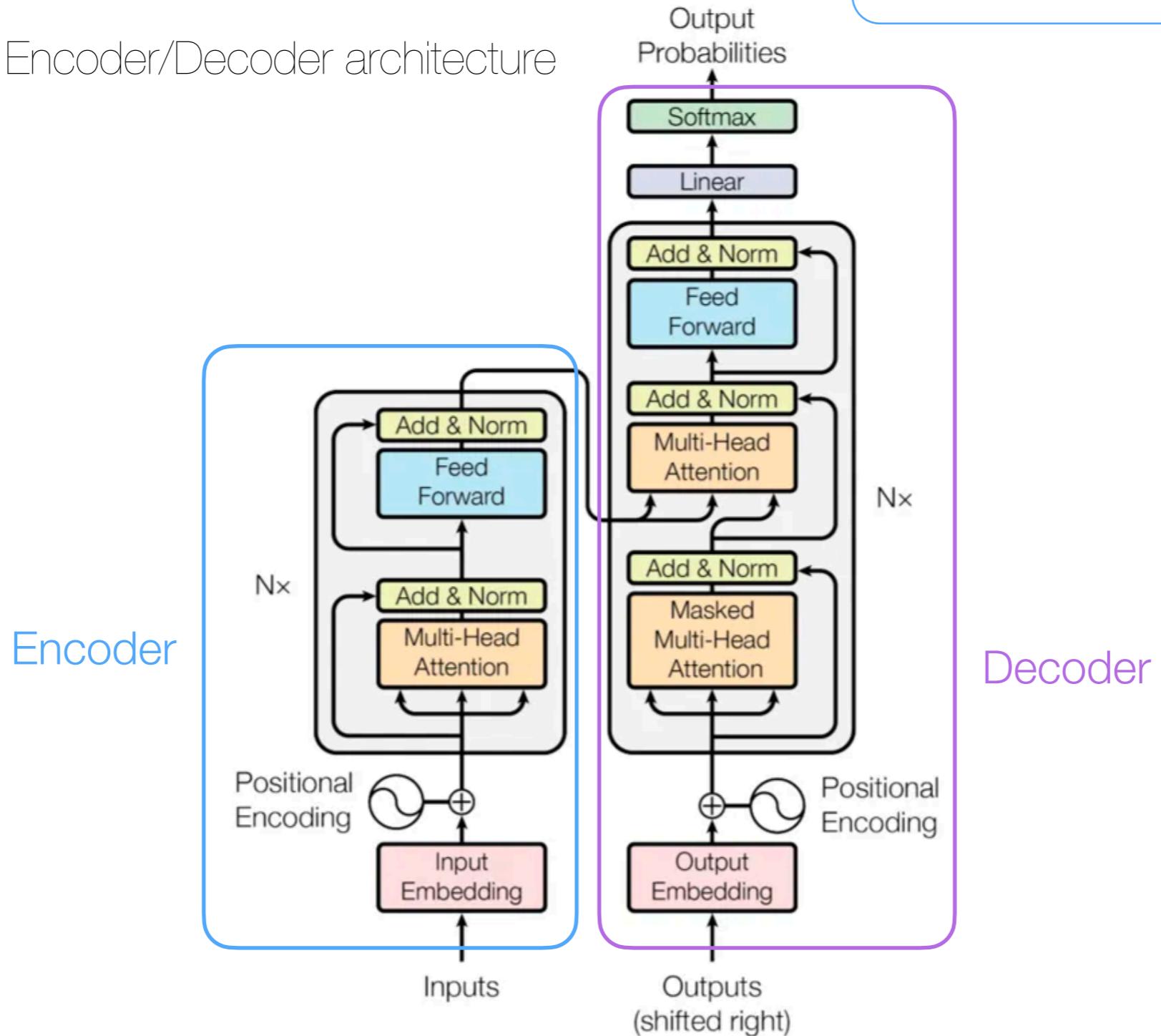
arXiv:1706.03762



Transformers

arXiv:1706.03762

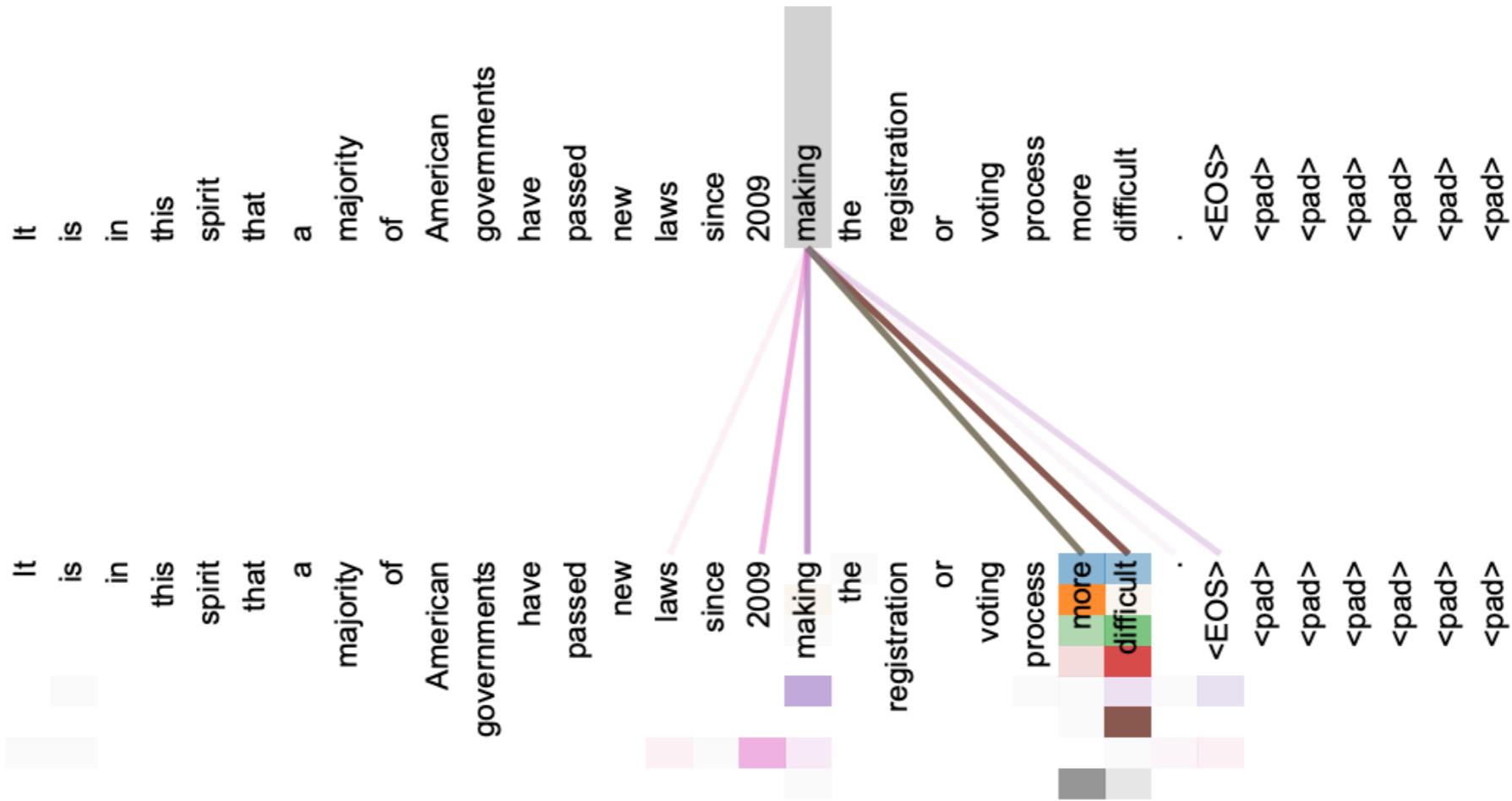
- Build on the Encoder/Decoder architecture by including



Attention

arXiv:1706.03762

- “Simple” mechanism to allow each token to take the context it appears in into account
- Requires exponentially more weights to be computed (from each token to every other token)



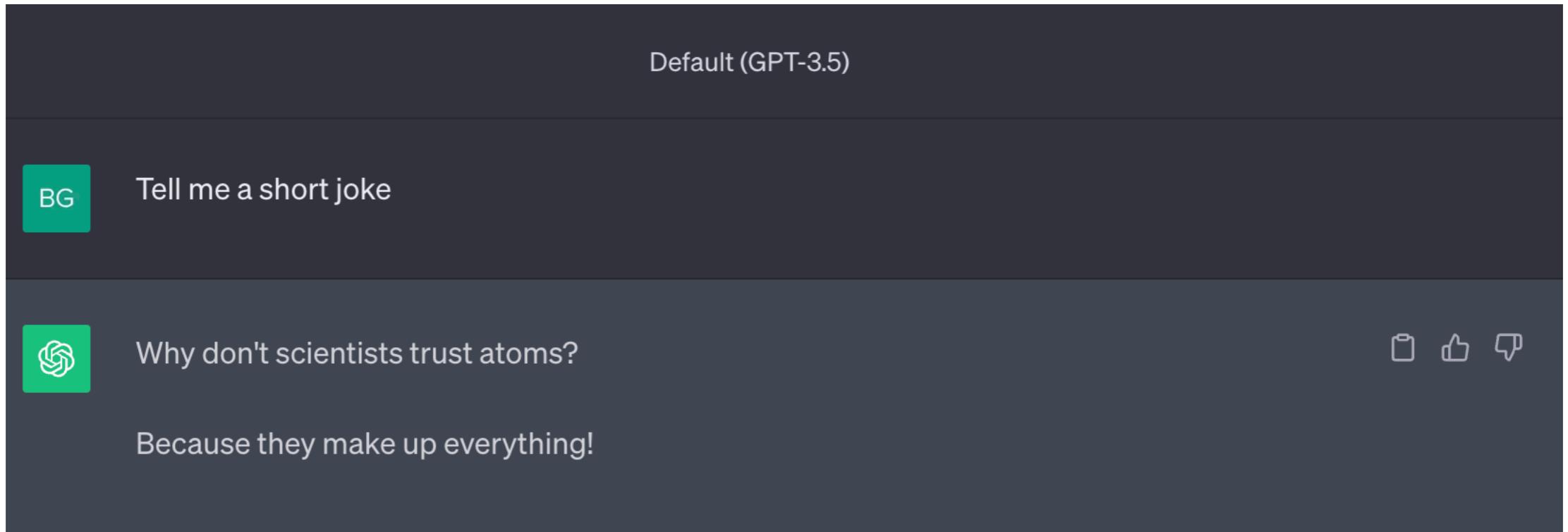
- Weights indicate the importance of each word relative to the current one

Temperature

- “Temperature” controls the degree of randomness in the output of a LLM.
- The higher the temperature, the higher the degree of randomness in the output as the model isn’t as limited by the training data.
 - **High Temperature** ($>=1$): High degree of randomness/creativity. Output might be non-sensical and will include many new sentences and combinations of words.
 - **Medium Temperature** ($\sim 0.5-1$): Compromise between increased creativity and meaningfulness.
 - **Low Temperature** (~ 0.0): Highly deterministic output with a lower degree of creativity. Generates text closer to the training dataset
- The optimal temperature value is task/application dependent as it will impact the style of text generation.

Large Language Models

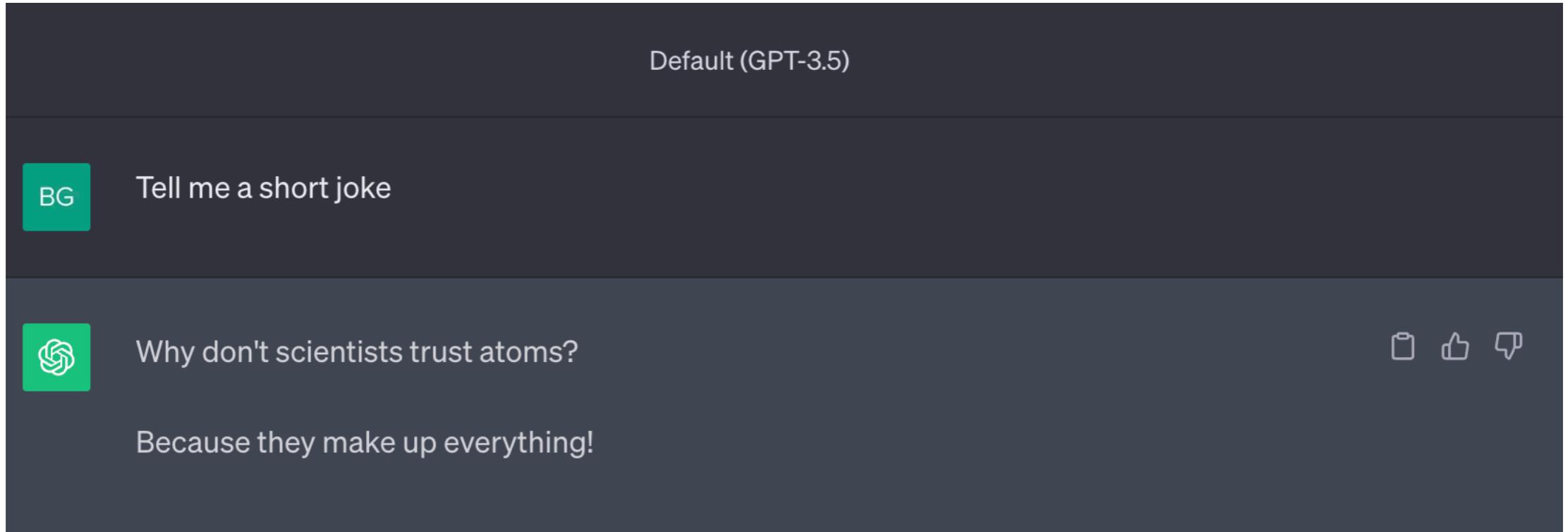
- The combination of this approach and huge amounts of data is almost magical:



- making it easy for us to forget that, in the end, the computer is just playing a game of guess the next word with limited understanding of the content of the text it is producing

Large Language Models

- The combination of this approach and huge amounts of data is almost magical:



- making it easy for us to forget that, in the end, the computer is just playing a game of guess the next word with limited understanding of the content of the text it is producing

Hallucinations

<https://www.reuters.com/legal/new-york-lawyers-sanctioned-using-fake-chatgpt-cases-legal-brief-2023-06-22/>

- LLMs are just trying to guess the next word with limited or no understanding of what they're "talking about"
- The output produced can easily be non-sensical, or include information and details that are completely fabricated.

Hallucinations

<https://www.reuters.com/legal/new-york-lawyers-sanctioned-using-fake-chatgpt-cases-legal-brief-2023-06-22/>

- LLMs are just trying to guess the next word with limited or no understanding of what they're "talking about"
- The output produced can easily be non-sensical, or include information and details that are completely fabricated.
- A famous recent example:



World ▾ Business ▾ Markets ▾ Sustainability ▾ Legal ▾ Breakingviews Technology ▾ Inve

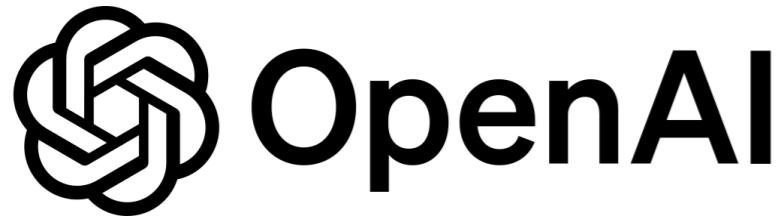
Disrupted

New York lawyers sanctioned for using fake ChatGPT cases in legal brief

By **Sara Merken**

June 26, 2023 4:28 AM EDT · Updated 3 months ago





openai.com

- American Research Lab, founded in 2015 by Ilya Sutskever, a former Google employee
- Heavily funded by Microsoft (\$10B in 2023)
- Creator of some of the current state of the art models of Generative AI
 - May 2020 - [GPT-3](#)
 - Jan 2021 - [DALL-E](#)
 - Aug 2021 - [Codex](#)
 - Jul 2022 - [DALL-E 2](#)
 - Nov 2022 - [ChatGPT](#) (based on GPT-3.5)
 - Mar 2023 - [GPT-4](#)



Generative AI and OpenAI
<https://github.com/DataForScience/OpenAI>



2. GPT Models

Basic Usage

- You might be familiar with the basic web interface known as ChatGPT where you interact with the system through a basic text prompt as if you were text messaging your friends:

The screenshot shows a dark-themed version of the ChatGPT interface. At the top, it says "Default (GPT-3.5)". A user message "BG What was Superman's weakness?" is shown, followed by a response from the AI. The AI's response is as follows:

Superman's most famous weakness is Kryptonite. Kryptonite is a mineral from his home planet, Krypton, and exposure to it weakens him and can even be fatal depending on the amount and duration of exposure. The effects of Kryptonite on Superman can range from weakness and nausea to more severe symptoms, such as loss of his powers and, in extreme cases, death.

There are different forms of Kryptonite, each with slightly different effects. The most common is green Kryptonite, but there are also variations like red Kryptonite, which has unpredictable and temporary effects, and gold Kryptonite, which permanently removes Superman's powers.

In addition to Kryptonite, Superman also has vulnerabilities to other elements or conditions, depending on the comic book or storyline. For example, he is vulnerable to magic, psychic attacks, and certain supernatural forces. However, Kryptonite remains his most iconic and well-known weakness.

Basic Usage

- Programmatically, things look a bit different
- The basic API call is [ChatCompletion.create\(\)](#)
- It takes two required arguments:
 - **model** - The model to use. ChatGPT is based on **gpt-3.5-turbo**
 - **messages** - A list of dictionaries representing the conversation so far. Each element has several possible fields:
 - **"role"** [required] - Three options
 - **"system"** - Instructs the model on how to behave
 - **"user"** - Represents user input
 - **"assistant"** - Corresponds to the output generated by the system
 - **"content"** [required] - Free-form text
 - **"name"** [optional] - An optional name field to be used to identify the participants in the conversation

Basic Usage

```
1 response = openai.ChatCompletion.create(
2     model="gpt-3.5-turbo",
3     messages=[
4         {"role": "user", "content": "What was Superman's weakness?"},
5     ]
6 )
```

- response is a `openai.openai_object.OpenAIObject` that you can treat as a `JSON` object

```
<OpenAIObject chat.completion id=chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR at 0x7fe6884c5fd0> JSON: {
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "Superman's main weakness is Kryptonite, a mineral from his home planet of Krypton. Exposure to Kryptonite weakens and eventually kills him. Additionally, Superman is vulnerable to magic and mind control.",
        "role": "assistant"
      }
    }
  ],
  "created": 1696104102,
  "id": "chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR",
  "model": "gpt-3.5-turbo-0613",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 43,
    "prompt_tokens": 13,
    "total_tokens": 56
  }
}
```

Basic Usage

```
1 response = openai.ChatCompletion.create(
2     model="gpt-3.5-turbo",
3     messages=[
4         {"role": "user", "content": "What was Superman's weakness?"},
5     ]
6 )
```

- response is a `openai.openai_object.OpenAIObject` that you can treat as a `JSON` object

```
<OpenAIObject chat.completion id=chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR at 0x7fe6884c5fd0> JSON: {
    "choices": [
        {
            "finish_reason": "stop",
            "index": 0,
            "message": {
                "content": "Superman's main weakness is Kryptonite, a mineral from his home planet of Krypton. Exposure to Kryptonite weakens and eventually kills him. Additionally, Superman is vulnerable to magic and mind control.",
                "role": "assistant"
            }
        }
    ],
    "created": 1696104102,
    "id": "chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR",
    "model": "gpt-3.5-turbo-0613",
    "object": "chat.completion",
    "usage": {
        "completion_tokens": 43,
        "prompt_tokens": 13,
        "total_tokens": 56
    }
}
```

The meat of the answer generated by the model

Basic Usage

```
1 response = openai.ChatCompletion.create(  
2     model="gpt-3.5-turbo",  
3     messages=[  
4         {"role": "user", "content": "What was Superman's weakness?"},  
5     ]  
6 )
```

- response is a `openai.openai_object.OpenAIObject` that you can treat as a `JSON` object

```
<OpenAIObject chat.completion id=chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR at 0x7fe6884c5fd0> JSON: {  
    "choices": [  
        {  
            "finish_reason": "stop",  
            "index": 0,  
            "message": {  
                "content": "Superman's main weakness is Kryptonite, a mineral from his home planet of Krypton. Exposure to  
Kryptonite weakens and eventually kills him. Additionally, Superman is vulnerable to magic and mind control.",  
                "role": "assistant"  
            }  
        }  
    ],  
    "created": 1696104102,  
    "id": "chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR",  
    "model": "gpt-3.5-turbo-0613",  
    "object": "chat.completion",  
    "usage": {  
        "completion_tokens": 43,  
        "prompt_tokens": 13,  
        "total_tokens": 56  
    }  
}
```

The reason the model stopped generating output.

Basic Usage

```
1 response = openai.ChatCompletion.create(
2     model="gpt-3.5-turbo",
3     messages=[
4         {"role": "user", "content": "What was Superman's weakness?"},
5     ]
6 )
```

- response is a `openai.openai_object.OpenAIObject` that you can treat as a `JSON` object

```
<OpenAIObject chat.completion id=chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR at 0x7fe6884c5fd0> JSON: {
    "choices": [
        {
            "finish_reason": "stop",
            "index": 0,
            "message": {
                "content": "Superman's main weakness is Kryptonite, a mineral from his home planet of Krypton. Exposure to Kryptonite weakens and eventually kills him. Additionally, Superman is vulnerable to magic and mind control.",
                "role": "assistant"
            }
        }
    ],
    "created": 1696104102,
    "id": "chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR",
    "model": "gpt-3.5-turbo-0613",
    "object": "chat.completion",
    "usage": {
        "completion_tokens": 43,
        "prompt_tokens": 13,
        "total_tokens": 56
    }
}
```

The output generated, formatted as the prompt message

Basic Usage

```
1 response = openai.ChatCompletion.create(
2     model="gpt-3.5-turbo",
3     messages=[
4         {"role": "user", "content": "What was Superman's weakness?"},
5     ]
6 )
```

- response is a `openai.openai_object.OpenAIObject` that you can treat as a `JSON` object

```
<OpenAIObject chat.completion id=chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR at 0x7fe6884c5fd0> JSON: {
    "choices": [
        {
            "finish_reason": "stop",
            "index": 0,
            "message": {
                "content": "Superman's main weakness is Kryptonite, a mineral from his home planet of Krypton. Exposure to Kryptonite weakens and eventually kills him. Additionally, Superman is vulnerable to magic and mind control.",
                "role": "assistant"
            }
        }
    ],
    "created": 1696104102,
    "id": "chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR",
    "model": "gpt-3.5-turbo-0613",
    "object": "chat.completion",
    "usage": {
        "completion_tokens": 43,
        "prompt_tokens": 13,
        "total_tokens": 56
    }
}
```

The index of the current output, in case several were generated

Basic Usage

```
1 response = openai.ChatCompletion.create(  
2     model="gpt-3.5-turbo",  
3     messages=[  
4         {"role": "user", "content": "What was Superman's weakness?"},  
5     ]  
6 )
```

- response is a `openai.openai_object.OpenAIObject` that you can treat as a `JSON` object

```
<OpenAIObject chat.completion id=chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR at 0x7fe6884c5fd0> JSON: {  
    "choices": [  
        {  
            "finish_reason": "stop",  
            "index": 0,  
            "message": {  
                "content": "Superman's main weakness is Kryptonite, a mineral from his home planet of Krypton. Exposure to  
Kryptonite weakens and eventually kills him. Additionally, Superman is vulnerable to magic and mind control.",  
                "role": "assistant"  
            }  
        }  
    ],  
    "created": 1696104102,  
    "id": "chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR",  
    "model": "gpt-3.5-turbo-0613",  
    "object": "chat.completion",  
    "usage": {  
        "completion_tokens": 43,  
        "prompt_tokens": 13,  
        "total_tokens": 56  
    }  
}
```

The model version used

The last 4 digits correspond to the date when this
version of the model was released. June 13th

Basic Usage

```
1 response = openai.ChatCompletion.create(  
2     model="gpt-3.5-turbo",  
3     messages=[  
4         {"role": "user", "content": "What was Superman's weakness?"},  
5     ]  
6 )
```

- response is a `openai.openai_object.OpenAIObject` that you can treat as a `JSON` object

```
<OpenAIObject chat.completion id=chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR at 0x7fe6884c5fd0> JSON: {  
    "choices": [  
        {  
            "finish_reason": "stop",  
            "index": 0,  
            "message": {  
                "content": "Superman's main weakness is Kryptonite, a mineral from his home planet of Krypton. Exposure to  
Kryptonite weakens and eventually kills him. Additionally, Superman is vulnerable to magic and mind control.",  
                "role": "assistant"  
            }  
        }  
    ],  
    "created": 1696104102,  
    "id": "chatcmpl-84aRiK39f34nXruKM9BZcESRA4HbR",  
    "model": "gpt-3.5-turbo-0613",  
    "object": "chat.completion",  
    "usage": {  
        "completion_tokens": 43,  
        "prompt_tokens": 13,  
        "total_tokens": 56  
    }  
}
```

Information on the model used
and **how many tokens** were consumed

Pricing

openai.com/pricing

- Unfortunately, OpenAI is not free to use, and the cost depends on the model used and the context size. Cost can vary by **20x** from one model to another

GPT-3.5 Turbo models are capable and cost-effective.

`gpt-3.5-turbo` is the flagship model of this family and is optimized for dialog.

`gpt-3.5-turbo-instruct` is an Instruct model and only supports a 4K context window.

[Learn about GPT-3.5 Turbo ↗](#)

Model	Input	Output
4K context	\$0.0015 / 1K tokens	\$0.002 / 1K tokens
16K context	\$0.003 / 1K tokens	\$0.004 / 1K tokens

With broad general knowledge and domain expertise, GPT-4 can follow complex instructions in natural language and solve difficult problems with accuracy.

[Learn about GPT-4](#)

Model	Input	Output
8K context	\$0.03 / 1K tokens	\$0.06 / 1K tokens
32K context	\$0.06 / 1K tokens	\$0.12 / 1K tokens

Pricing

openai.com/pricing

- Unfortunately, OpenAI is not free to use, and the cost depends on the model used and the context size. Cost can vary by **20x** from one model to another

GPT-3.5 Turbo models are capable and cost-effective.

`gpt-3.5-turbo` is the flagship model of this family and is optimized for dialog.

`gpt-3.5-turbo-instruct` is an Instruct model and only supports a 4K context window.

[Learn about GPT-3.5 Turbo ↗](#)

Input and output tokens are also
priced differently

Model	Input	Output
4K context	\$0.0015 / 1K tokens	\$0.002 / 1K tokens
16K context	\$0.003 / 1K tokens	\$0.004 / 1K tokens

With broad general knowledge and domain expertise, GPT-4 can follow complex instructions in natural language and solve difficult problems with accuracy.

[Learn about GPT-4](#)

Model	Input	Output
8K context	\$0.03 / 1K tokens	\$0.06 / 1K tokens
32K context	\$0.06 / 1K tokens	\$0.12 / 1K tokens

Rate Limits

- Rate limits are imposed to prevent DDoS attacks and misconfigured applications from running rampant

platform.openai.com/account/rate-limits

Rate limits

OpenAI enforces rate limits on the requests you can make to the API. These are applied over tokens-per-minute, requests-per-minute (in some cases requests-per-day), or in the case of image models, images-per-minute.

Learn more in the [rate limits documentation](#), or reference the [default rate limits](#) for our models. Your organization's rate limits are listed below.

MODEL	TPM	RPM
CHAT		
gpt-3.5-turbo	90,000	3,500
gpt-3.5-turbo-0301	90,000	3,500
gpt-3.5-turbo-0613	90,000	3,500
gpt-3.5-turbo-16k	180,000	3,500
gpt-3.5-turbo-16k-0613	180,000	3,500
gpt-3.5-turbo-instruct	250,000	3,000
gpt-3.5-turbo-instruct-0914	250,000	3,000
gpt-4	10,000	200
gpt-4-0314	10,000	200
gpt-4-0613	10,000	200

ChatGPT infrastructure is expensive



The image shows a screenshot of a Forbes article. At the top right is the word "Forbes" in white. Below it, the article's category path is shown: "FORBES > INNOVATION > CONSUMER TECH". The main title of the article is "ChatGPT Burns Millions Every Day. Can Computer Scientists Make AI One Million Times More Efficient?". Below the title, the author is identified as "John Koetsier Senior Contributor ⓘ Journalist, analyst, author, and speaker.". To the right of the author's name is a blue "Follow" button. Further down, there are two interactive icons: a speech bubble with the number "2" and a play button icon with the text "Listen to article 9 minutes". The date of publication is listed as "Feb 10, 2023, 03:09pm EST".

FORBES > INNOVATION > CONSUMER TECH

ChatGPT Burns Millions Every Day. Can Computer Scientists Make AI One Million Times More Efficient?

John Koetsier Senior Contributor ⓘ
Journalist, analyst, author, and speaker.

Follow

2

Feb 10, 2023, 03:09pm EST

Listen to article 9 minutes

Basic Usage

```
▼ 1 response = openai.ChatCompletion.create(  
 2     model="gpt-3.5-turbo",  
▼ 3     messages=[  
 4         {"role": "user", "content": "What is a type of Kryptonite?"},  
 5     ],  
 6     n=3  
 7 )
```

- We can also request multiple outputs simultaneously using the `n` argument to `openai.ChatCompletion.create()`.
- Each completion is listed in the responses `choices` array with a sequential index



Code - GPT Models

<https://github.com/DataForScience/OpenAI>

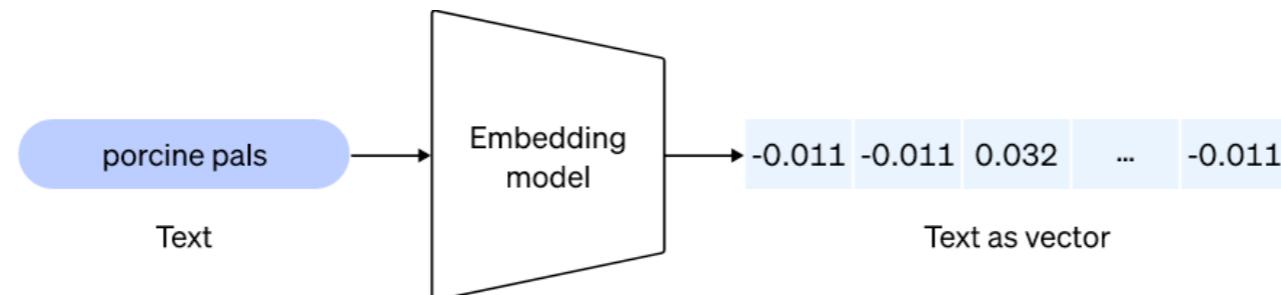


3. Embeddings

Embeddings

<https://openai.com/blog/new-and-improved-embedding-model>

- Embeddings are a fundamental concept in natural language processing (NLP)
- An embedding is simply a mapping between a piece of text (a word, a sentence, etc) and a dense numerical vector



- Embeddings are trained specifically to include semantic and even contextual information about the text being embedded
- We can measure **how close the meaning** of two snippets of text by measuring how **similar** their embedding vectors are

Embedding Models

- OpenAI provides several embedding models.
- The recommendation is to use [text-embedding-ada-002](#) for most use cases
- To retrieve the embedding vector for a piece of text, we can use the [Embedding API](#) endpoint:

```
▼ 1 response = openai.Embedding.create(  
 2   model="text-embedding-ada-002",  
 3   input="Your text goes here",  
 4 )
```

- The response object is structured similarly to usual ChatCompletion response, but now we have a data field instead of a messages field

```
<OpenAIObject list at 0x7faa78a902c0> JSON: {  
  "object": "list",  
  "data": [  
    {  
      "object": "embedding",  
      "index": 0,  
      "embedding": [...]  
    }  
  ],  
  "model": "text-embedding-ada-002-v2",  
  "usage": {  
    "prompt_tokens": 4,  
    "total_tokens": 4  
  }  
}
```

Embedding Utility functions

- OpenAPI also provides a series of embedding utility functions. In particular, we can get the embedding for a specific piece of text using `get_embedding()`

```
1 | get_embedding("text", engine=embedding_model)
```

- Which returns just the corresponding embedding vector
- There are several distance metrics we can use to compute the similarity between two vectors. The most common one is Cosine Similarity that measures the cosine of the angle between two vectors
- OpenAI provides `distances_from_embedding()` as a utility function to allow us to quickly compute the distance between one query vector and our entire set of embeddings

```
1 | distances = distances_from_embeddings(query_embedding, embeddings, distance_metric="cosine")
```

- `distances_from_embedding()` supports 4 different distance metrics:
 - Cosine similarity ("cosine"),
 - Euclidean distance ("L2"),
 - Manhattan distance ("L1")
 - Chebyshev distance ("Linf")

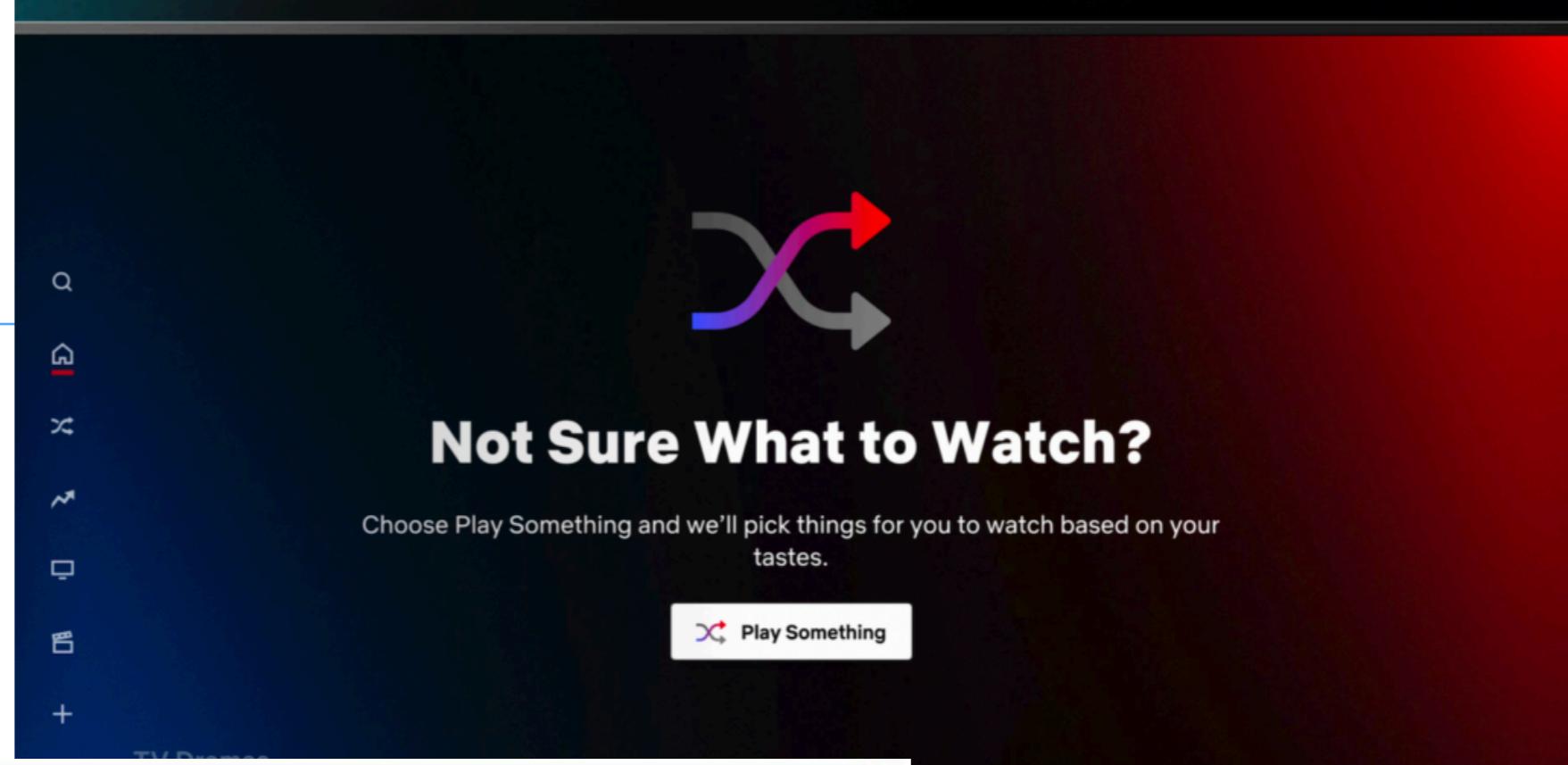
Embeddings and Encodings

- Embedding models work at the token level
- Embedding models are trained on a specific encoding of the tokens generated by a given tokenizer, so we must be careful to use the correct encoding
- To get the encoding associated with a specific embedding model, we use:

```
1 | encoding = tiktoken.encoding_for_model(embedding_model)
```

- And we must pass this encoding as an argument to the [get_embedding](#) function

Recommendations

A screenshot of the Amazon.com website. The header features the "amazon.com" logo. On the right, a large blue banner says "Recommended for You". Below it, a message states: "Amazon.com has new recommendations for you based on items you purchased or told us you own." Three book covers are shown in a grid, each with a "LOOK INSIDE!" button. The books are: "Google Apps Deciphered: Compute in the Cloud to Streamline Your Desktop", "Google Apps Administrator Guide: A Private-Label Web Workspace", and "Googlepedia: The Ultimate Google Resource (3rd Edition)".



Code - Embeddings
<https://github.com/DataForScience/OpenAI>



4. Image Generation

Image Models

arXiv:2204.06125

- Image models like **DALL-E** build on top of the **LLM** approach outlined above
- Essentially, **DALL-E replaces the LLM decoder** with an image decoder

Image Models

arXiv:2204.06125

- Image models like **DALL-E** build on top of the **LLM** approach outlined above
- Essentially, **DALL-E replaces the LLM decoder** with an image decoder
- The image decoder is trained using **Contrastive Training**:
 - A large list of images and textual descriptions is used to train the image decoder to generate an internal representation that matches the encoded description

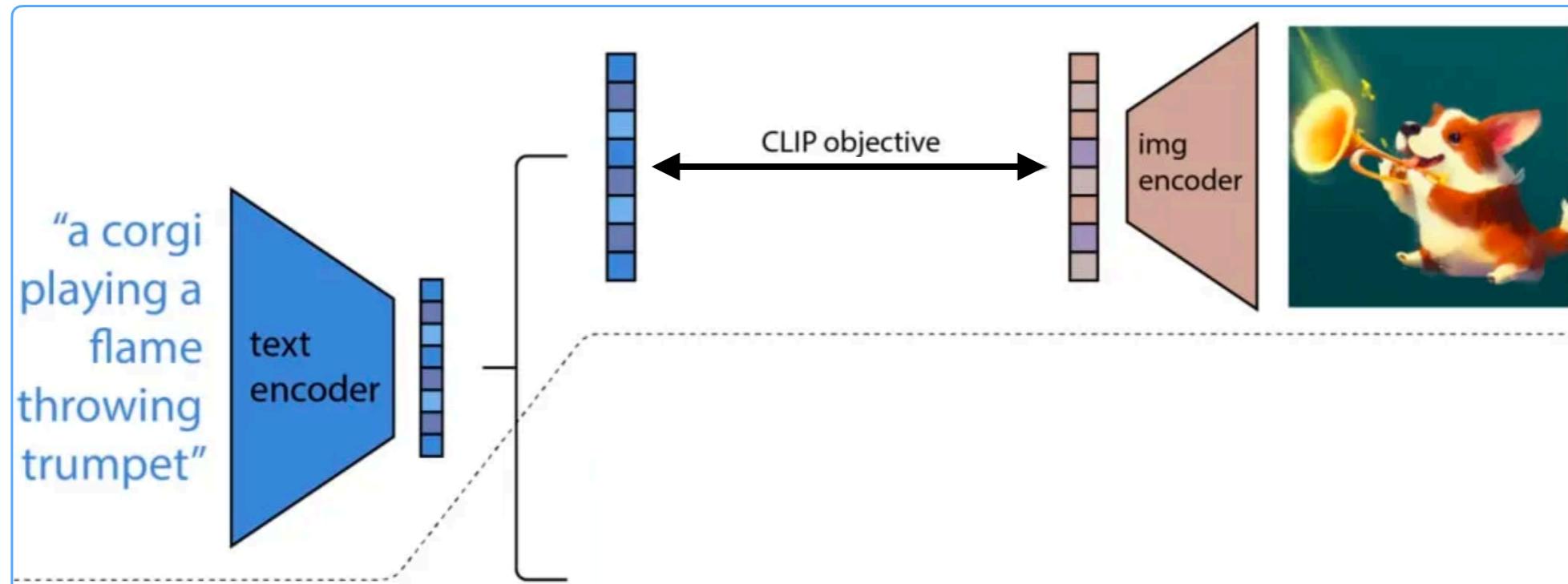


Image Models

arXiv:2204.06125

- Image models like **DALL-E** build on top of the **LLM** approach outlined above
- Essentially, **DALL-E replaces the LLM decoder** with an image decoder
- The image decoder is trained using **Contrastive Training**:
 - A large list of images and textual descriptions is used to train the image decoder to generate an internal representation that matches the encoded description
 - Inverting the process produces a model that converts textual descriptions into images

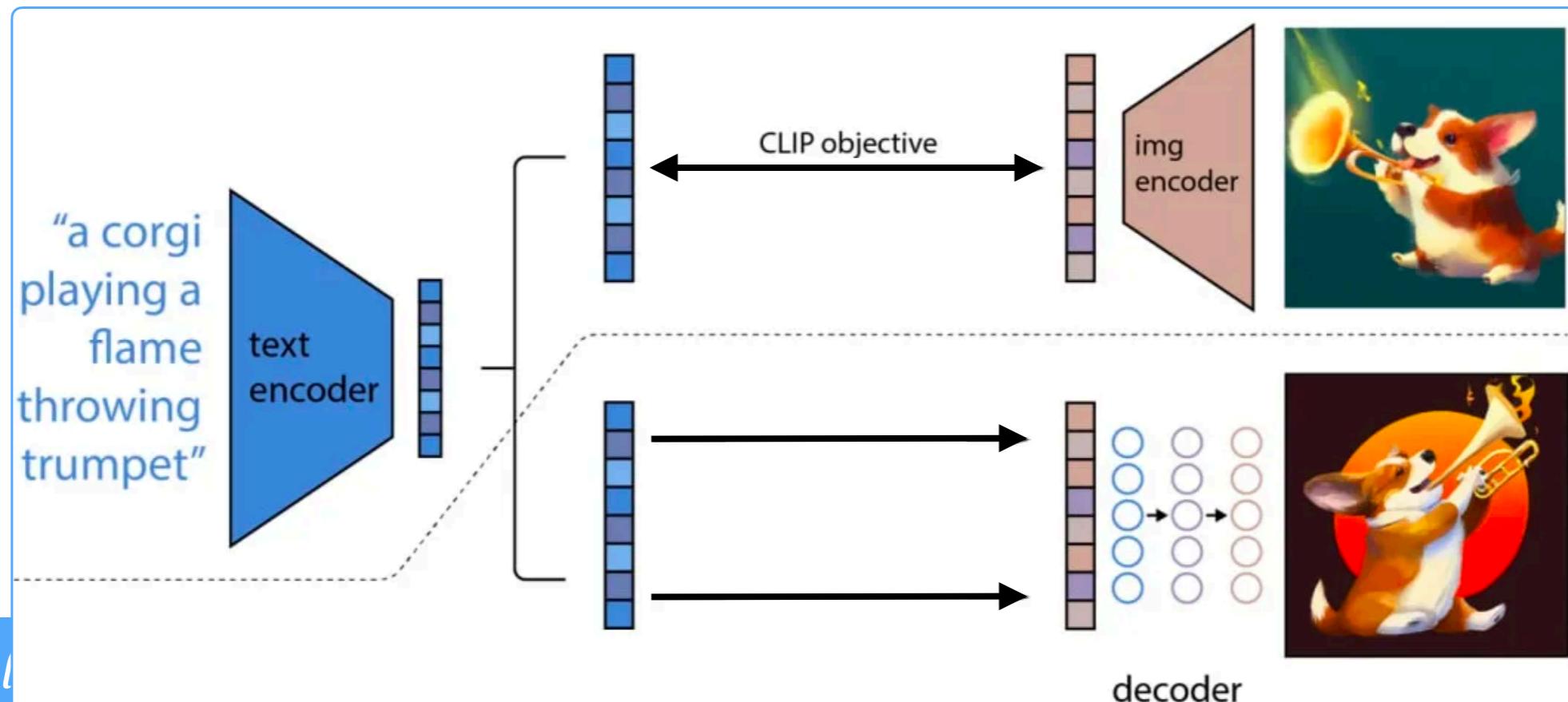
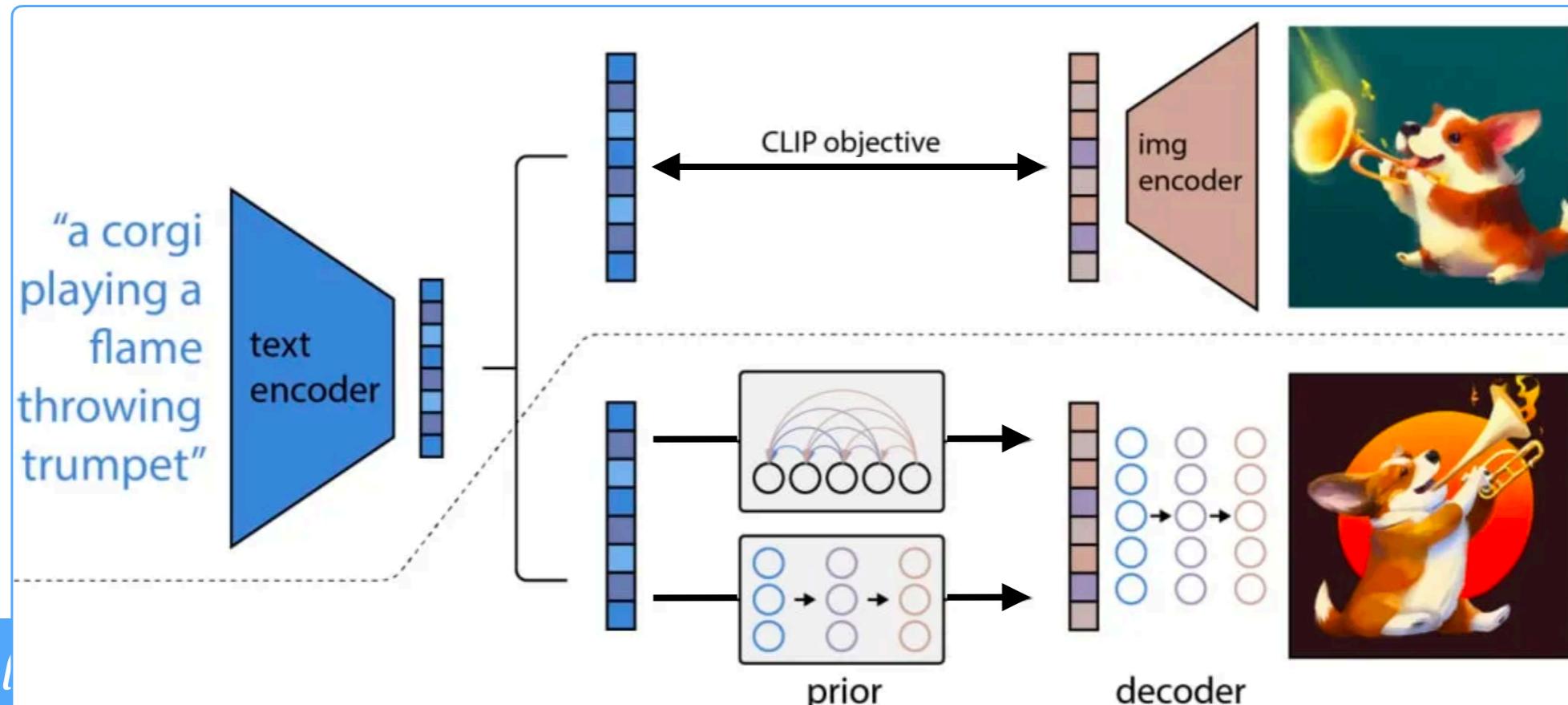


Image Models

arXiv:2204.06125

- Image models like **DALL-E** build on top of the **LLM** approach outlined above
- Essentially, **DALL-E replaces the LLM decoder** with an image decoder
- The image decoder is trained using **Contrastive Training**:
 - A large list of images and textual descriptions is used to train the image decoder to generate an internal representation that matches the encoded description
 - Inverting the process produces a model that converts textual descriptions into images
- As a final improvement, we add a **prior** in between the output of the text encoder and the input of the image decoder.



Generating Images from a Prompt

- To generate images, the basic process is similar to what we've seen so far
- We use the Image API endpoint:

```
1 response = openai.Image.create(  
2     prompt=prompt,  
3     n=1,  
4     size="1024x1024",  
5     response_format="url",  
6 )
```

- Since DALL-E isn't a conversational model, there's no need to specify a list of **messages**, and we can provide just a single prompt
- The image size is specified by the **size** argument.

Image Variations

- Given a specific image, we can also use DALL-E to produce variations using the [Image.create_variation\(\)](#) endpoint.
- We upload the original image in the `image` argument

```
1 response = openai.Image.create_variation(  
2     image=open(image_filename, 'rb'),  
3     n=3,  
4     size="1024x1024",  
5     response_format="url",  
6 )
```

- The `n` argument allows us to request multiple variations

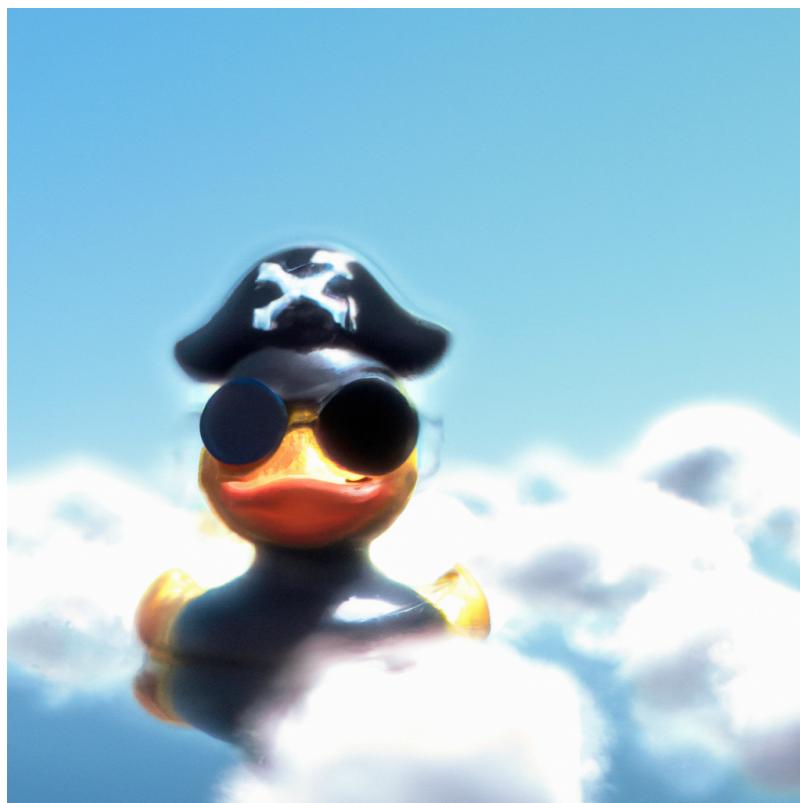
Image Edits

- DALL-E also has the functionality of editing existing images
- It works by generating just a portion of the image, while maintaining the rest of the image the same
- We specify which part of the image to regenerate by providing a mask file. The mask is simply an image where the portion we want to edit is set to be transparent

```
▼ 1 response = openai.Image.create_edit(  
 2     image=open(image_filename, "rb"),  
 3     mask=open(mask_filename, "rb"),  
 4     prompt=prompt,  
 5     n=1,  
 6     size="1024x1024",  
 7     response_format="url",  
 8 )
```

Image Edits

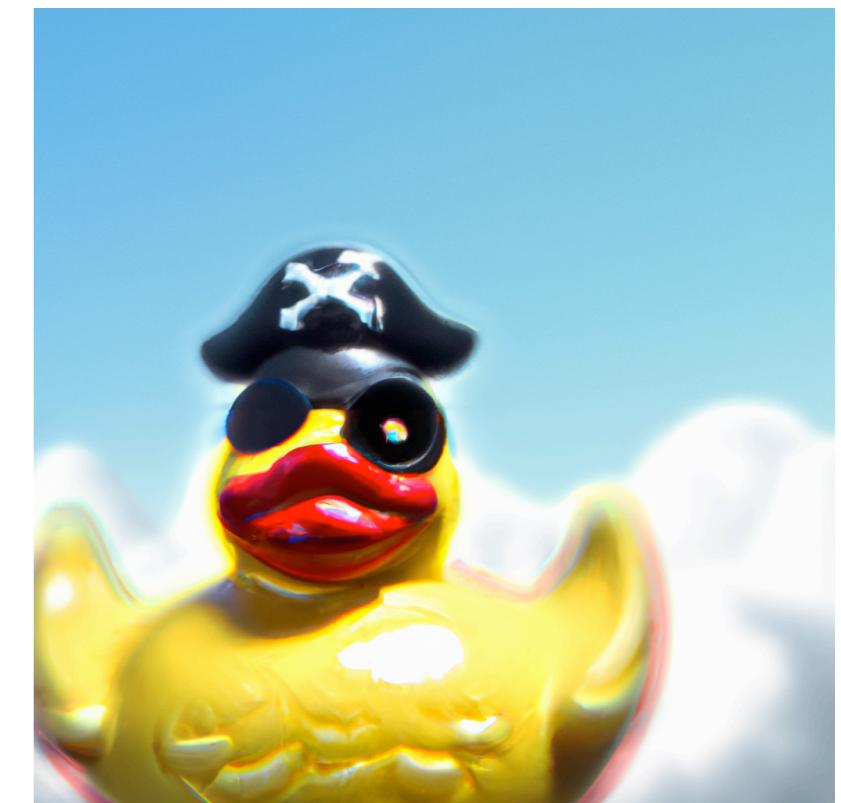
Original



Mask



Result



- We can see how the bottom half of the picture was completely replaced, while the top half remained the same

Prompt Expansion with GPT

- Due to its nature, DALL-E works best with relatively detailed prompts
- We can improve our prompts by enlisting the help of GPT, so now our image generation procedure requires two steps (and two API calls):
 - Use GPT to expand the prompt

```
▼ 1 response = openai.ChatCompletion.create(  
 2   model="gpt-3.5-turbo",  
 3   messages=[  
 4     {"role": "system", "content": """You're a brilliant graphic designer.  
 5      Generate an precise prompt for DALL-E that has at most 1000 characters."""}  
 6     },  
 7     {"role": "user", "content": prompt},  
 8   ]  
 9 )
```

- Use DALL-E to produce an image based on the expanded prompt

```
▼ 1 response = openai.Image.create(  
 2   prompt=expanded_prompt[:1000],  
 3   n=3,  
 4   size="1024x1024",  
 5   response_format="url",  
 6 )
```



Code - Image Generation
<https://github.com/DataForScience/OpenAI>



5. Code Generation and Explanation

CODEX Model

<https://openai.com/blog/openai-codex>

- OpenAI released the CODEX model in Aug 2021.
- It was a version of GPT-3 that was trained specifically in text and source code from GitHub.
- Designed to power the functionality of GitHub Copilot:<https://github.com/features/copilot> a digital assistant for programmers. In particular, CODEX was capable of
 - Produce code based on a prompt
 - Autocomplete your code as you're writing it
 - Suggest a useful library or API call for an application
 - Comment pre-existing code
 - Improve the efficiency of existing code
- In Mar 2023, CODEX was folded in to the general GPT-3.5-turbo model so all of this functionality is now available within the system we're already familiar with.

Generating Code from a Prompt

- To Generate code from a prompt, we must instruct our LLM on what code to produce. For example, these messages:

```
▼ 1 messages = [
  2   {"role": "system", "content": """You are a grumpy but expert Python programmer
  3   that interviewing for a job. Please be as concise with your answers as possible."""},
  4   {"role": "user", "content": """Create a recursive Python function to compute
  5   Fibonacci numbers. Don't provide any explanation, just the code"""},
  6 ]
```

- Produce the expected result

```
1 print(response["choices"][0]['message']['content'])

def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

- One important thing to remember is that we should provide as much information as a human requires.

Generating Comments

- Similarly, we can add comments to existing code by simply asking for them
- As we are looking for straightforward text, low **temperature** values (even **0**) tend to work best.

Interacting with a Database

- We can take advantage of GPT-3.5-turbo code generating functionality to easily explore a database
- Using detailed, sequential prompts we can have GPT-3.5-turbo generate the SQL code necessary to analyze the database
- Naturally, GPT needs to know the basics of the schema of each table before it can be made to answer our questions

```
 1 messages = [
 2   {"role": "system", "content": """You're a Database Administrator.
 3   Please generate SQL queries to answer the following questions.
 4   No comments are necessary."""},
 5   {"role": "user", "content": """
 6 # Table Employee, columns = [Id, LastName, First Name]
 7 # Table Shipper, columns = [Id, CompanyName, Phone]
 8 # Table OrderDetail, columns = [OrderId, ProductId, Quantity]
 9 # Table EmployeeTerritory, columns = [Id, EmployeeId, TerritoryId]
10   """},
11 ]
```



Code - Code Generation and
Explanation

<https://github.com/DataForScience/OpenAI>

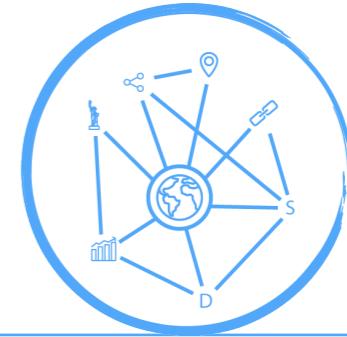
Question

- How was the technical level?
 - 1 — Too Low (too many details)
 - 2 — Low
 - 3 — Just Right
 - 4 — High
 - 5 — Too High (not enough details)

Question

- How was the level of Python code/explanations?
 - 1 — Too Low (too many details)
 - 2 — Low
 - 3 — Just Right
 - 4 — High
 - 5 — Too High (not enough details)

Events



graphs4sci.substack.com



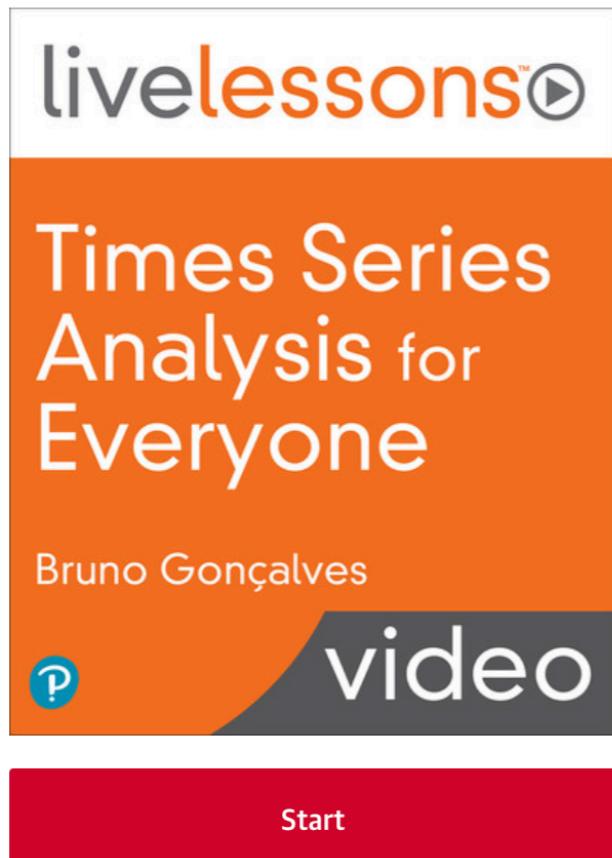
ChatGPT and Competing LLMs

Nov 29, 2023 - 9am-1pm (PST)

Times Series Analysis for Everyone

★★★★★ [1 review](#)

By [Bruno Gonçalves](#)



TIME TO COMPLETE:

6h

TOPICS:

[Time Series](#)

PUBLISHED BY:

[Pearson](#)

PUBLICATION DATE:

November 2021

https://bit.ly/Timeseries_LL

6 Hours of Video Instruction

The perfect introduction to time-based analytics

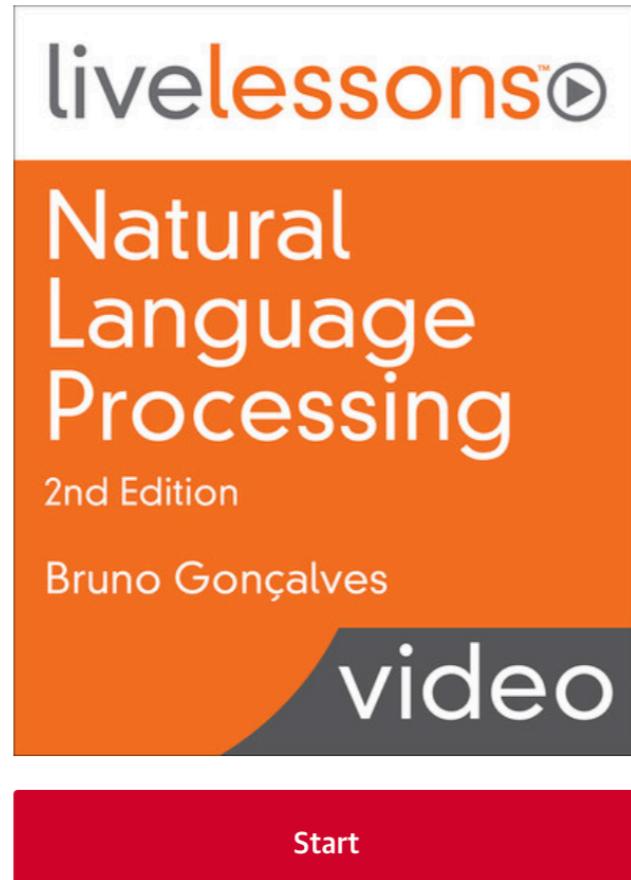
Overview

Times Series Analysis for Everyone LiveLessons covers the fundamental tools and techniques for the analysis of time series data. These lessons introduce you to the basic concepts, ideas, and algorithms necessary to develop your own time series applications in a step-by-step, intuitive fashion. The lessons follow a gradual progression, from the more specific to the more abstract, taking you from the very basics to some of the most recent and sophisticated algorithms by leveraging the statsmodels, arch, and Keras state-of-the-art models.

Natural Language Processing, 2nd Edition

Write the [first review](#)

By [Bruno Gonçalves](#)



TIME TO COMPLETE:

5h 23m

TOPICS:

[Natural Language Processing](#)

PUBLISHED BY:

[Addison-Wesley Professional](#)

PUBLICATION DATE:

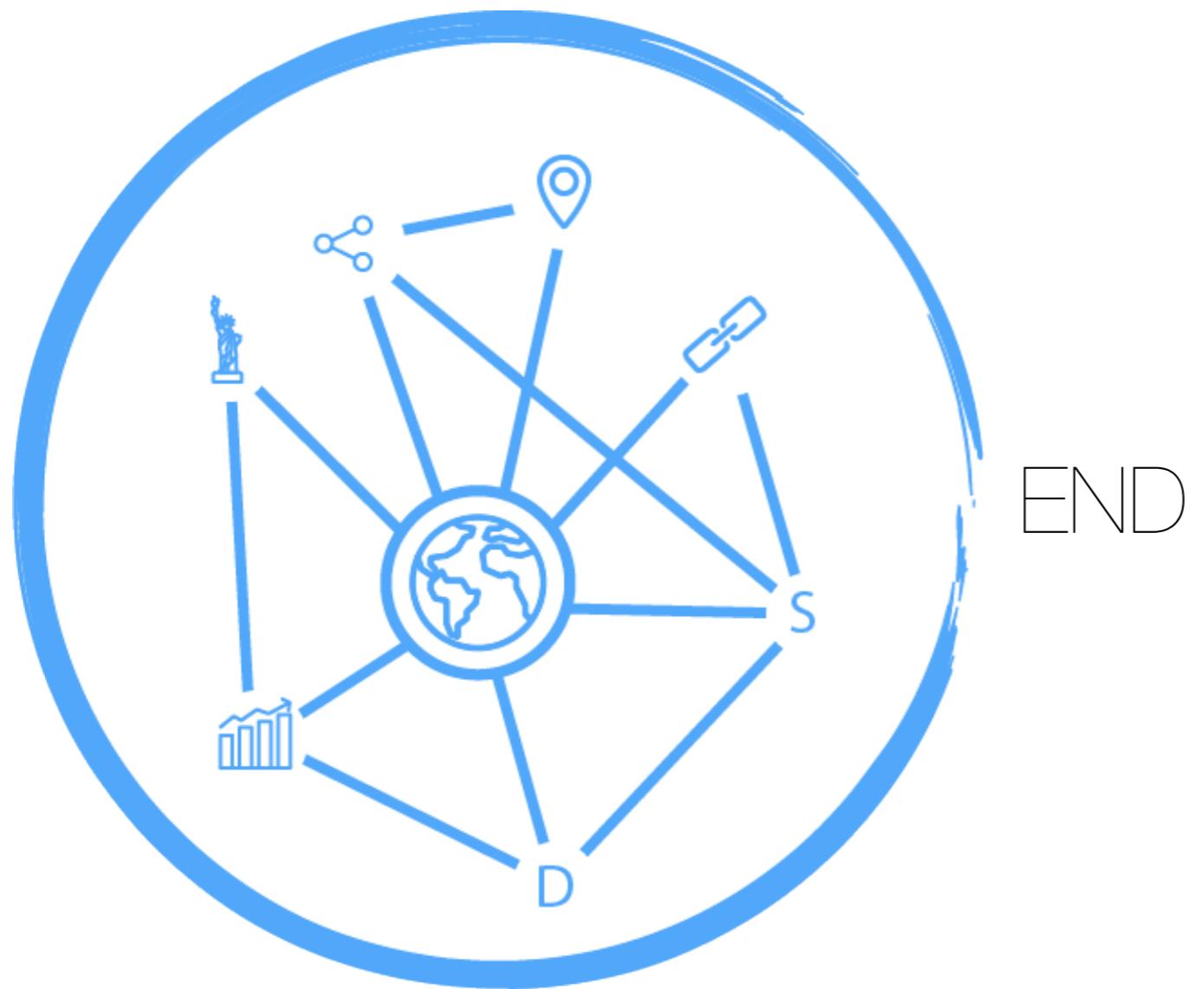
October 2021

https://bit.ly/NLP_LL

5 Hours of Video Instruction

Overview

Natural Language Processing LiveLessons covers the fundamentals of Natural Language Processing in a simple and intuitive way, empowering you to add NLP to your toolkit. Using the powerful NLTK package, it gradually moves from the basics of text representation, cleaning, topic detection, regular expressions, and sentiment analysis before moving on to the Keras deep learning framework to explore more advanced topics such as text classification and sequence-to-sequence models. After successfully completing these lessons you'll be equipped with a fundamental and practical understanding of state-of-the-art Natural Language Processing tools and algorithms.



END