Generative Artificial Intelligence with OpenAI

Bruno Gonçalves

data4sci.substack.com

https://github.com/DataForScience/OpenAI

Question

https://github.com/DataForScience/OpenAI

- What's your job title?
    - Data Scientist
    - Statistician
    - Data Engineer
    - Researcher
    - Business Analyst
    - Software Engineer
    - Other

Question

https://github.com/DataForScience/OpenAI

- How experienced are you in Python?
    - Beginner (<1 year)
    - Intermediate (1-5 years)
    - Expert (5+ years)

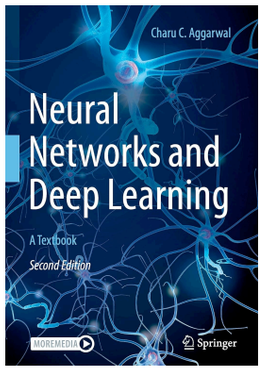
Question

https://github.com/DataForScience/OpenAI

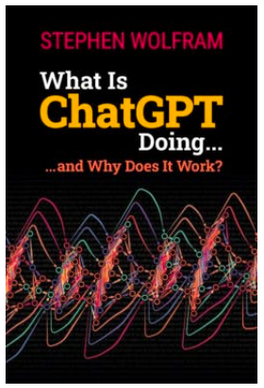
- How did you hear about this webinar?
    - O'Reilly Platform
    - Newsletter
    - data4sci.com Website
    - Previous event
    - Other?

References

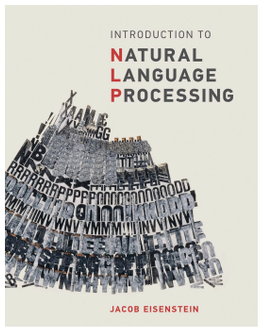
https://github.com/DataForScience/OpenAI



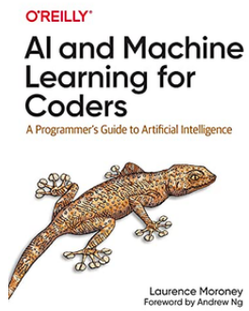
https://amzn.to/48rZn9X



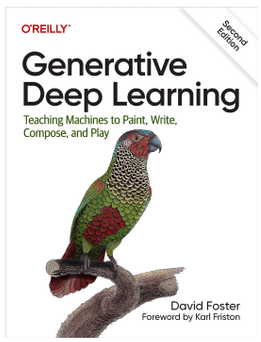
https://amzn.to/3LBRdBY



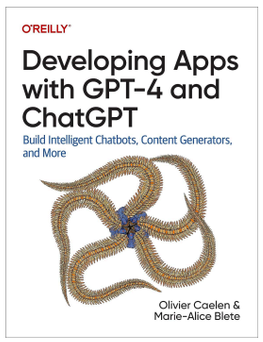
https://amzn.to/3ZMnTih



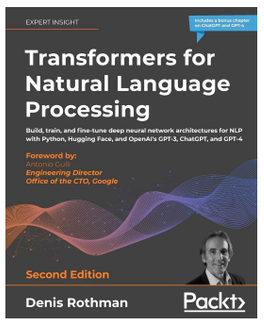
https://amzn.to/3KjB5W4



https://amzn.to/3t8PuxM



https://amzn.to/3RHRkQa



https://amzn.to/46kOo02

Table of Contents

1. Generative AI and OpenAI

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…

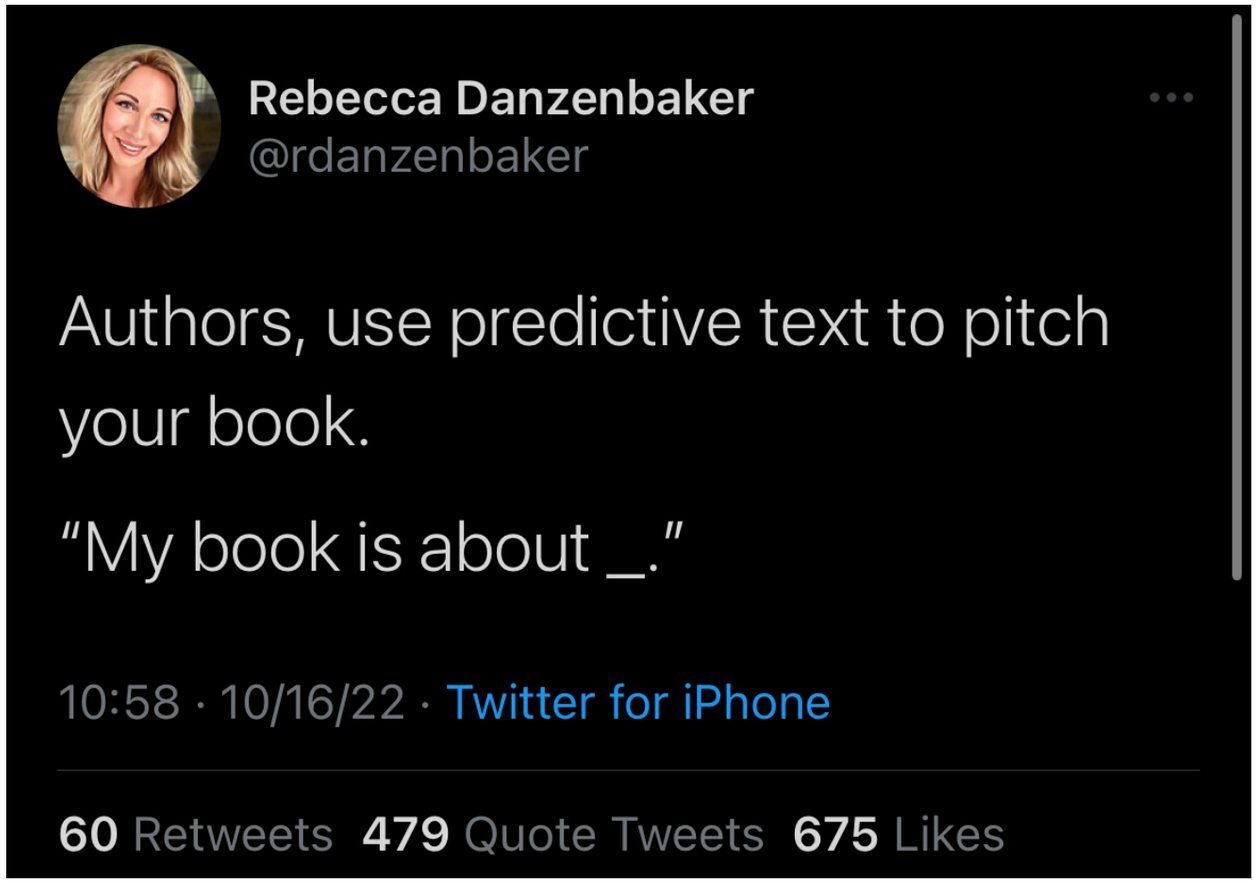
Language Models

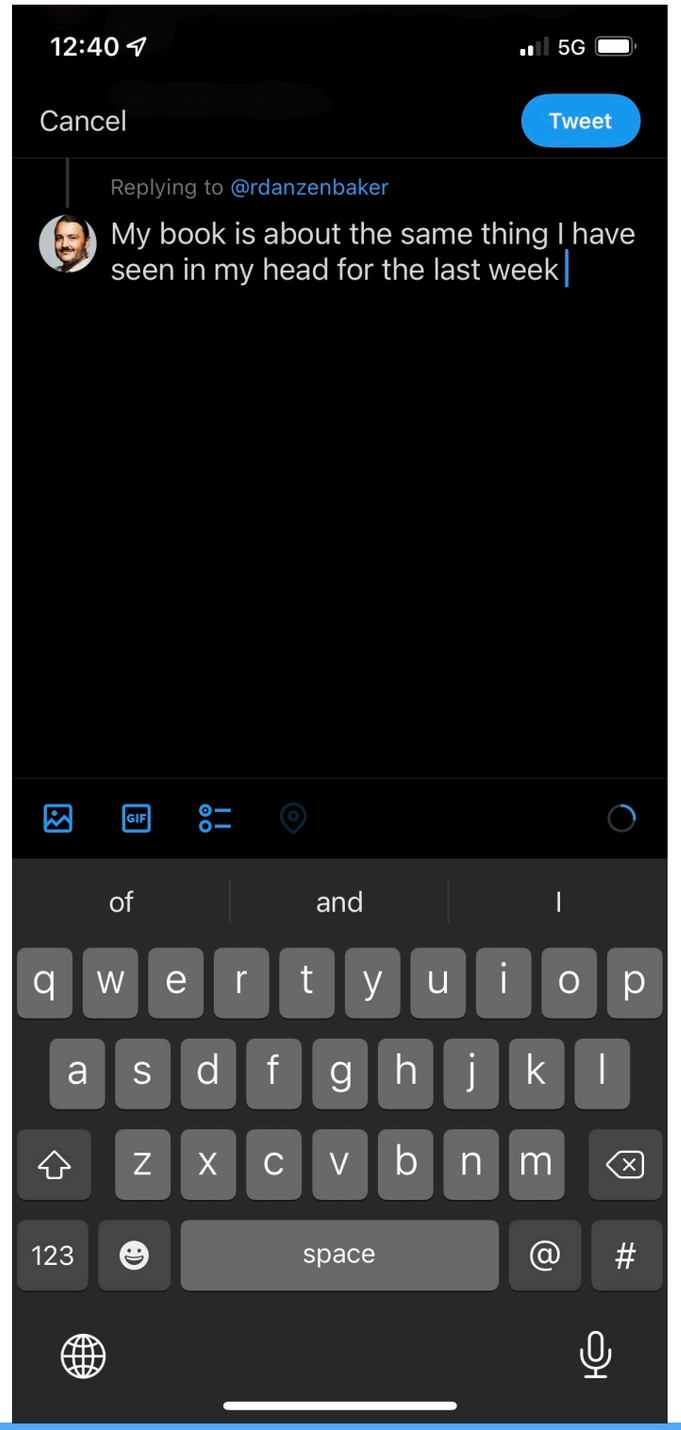
- If you the predictive text feature in your digital devices, you're already familiar with how language models work.

- It's just an endless game of "guess the next word"...

Language Models

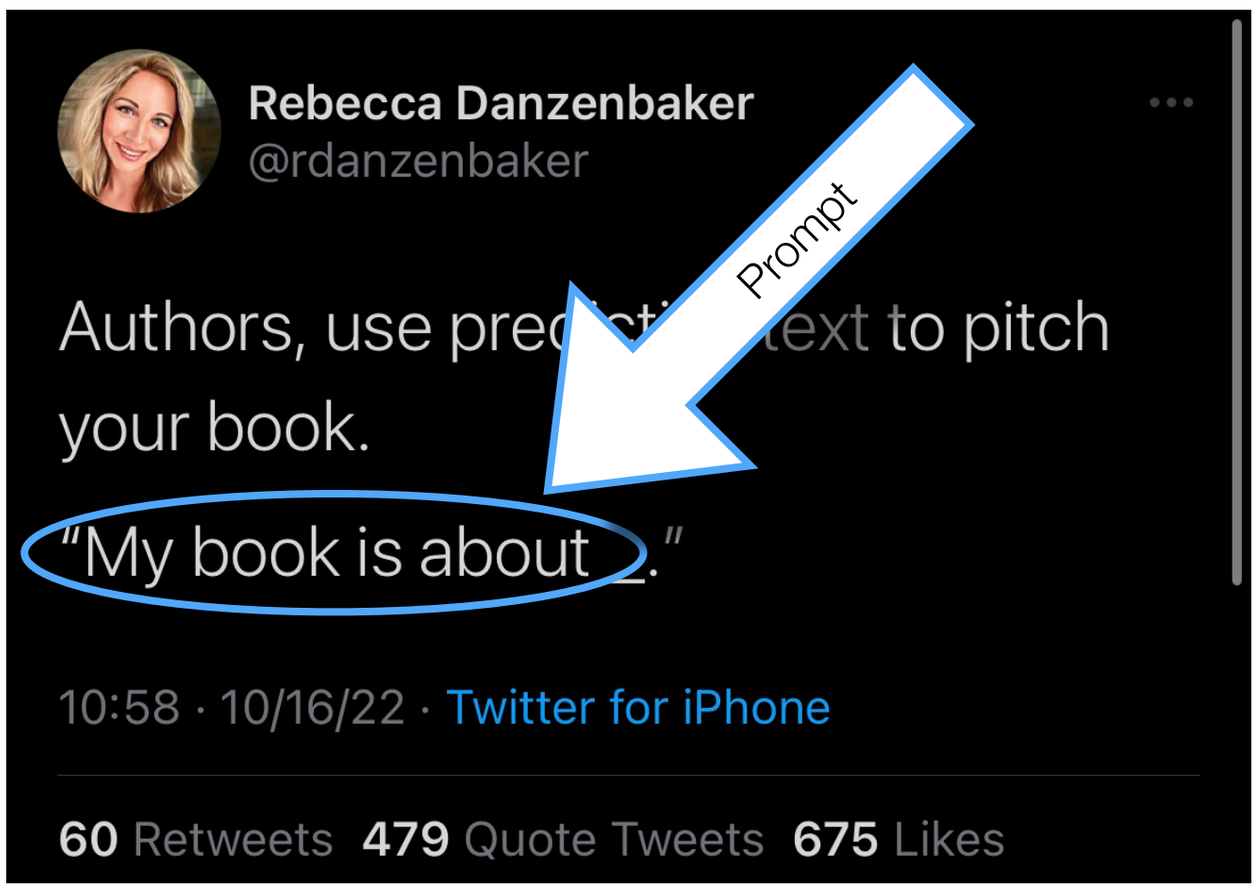
- If you the predictive text feature in your digital devices, you're already familiar with how language models work.
- It's just an endless game of "guess the next word"...



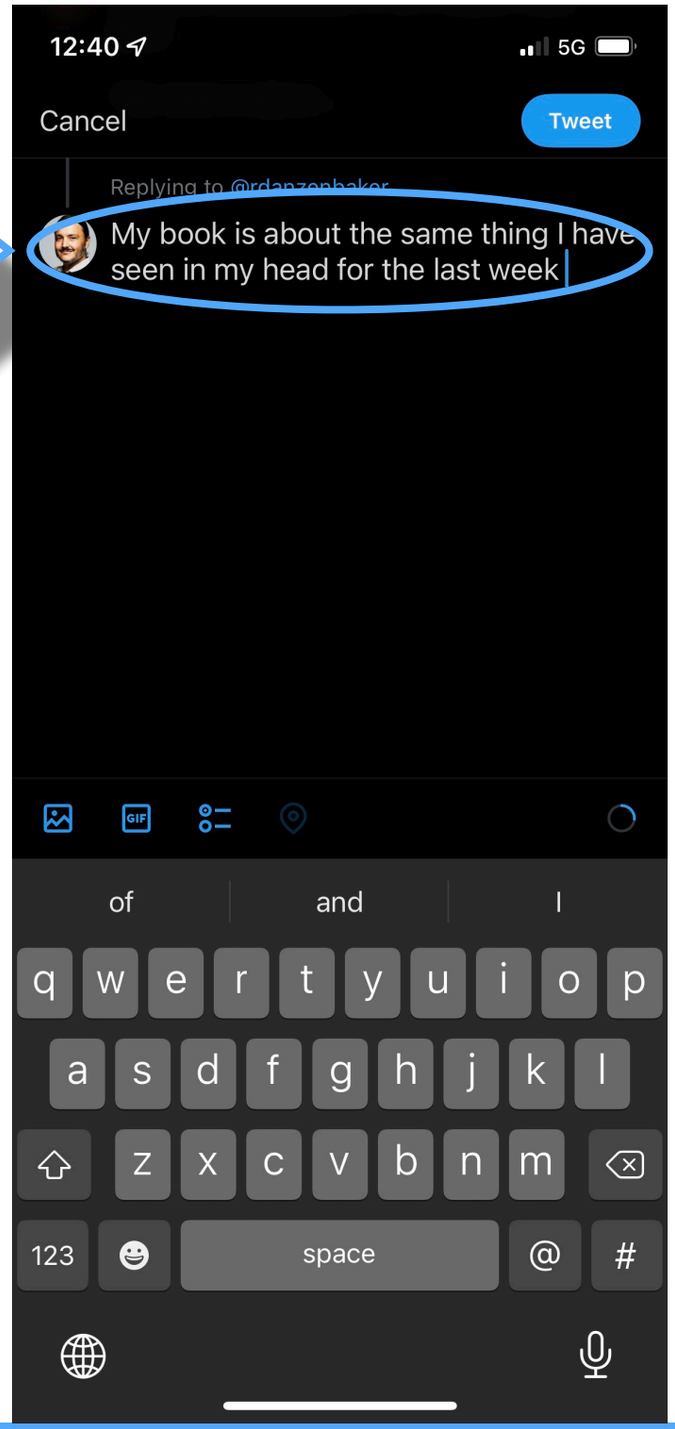


Language Models

- If you 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 nex



Output



Large Language Models

https://lifearchitect.ai/models/

- LLMs are… LARGE!

BILLBOARD CHART FOR LANGUAGE MODELS JUN/2024

MODELS

Now (Jun/2024)	6m ago (Dec/2023)	12m ago (Jun/2023)	ALScore	Model name Details	AI lab Openness
1	—	—	29.8	**Claude 3 Opus** 2T trained on 40T tokens*	◆ Anthropic API
2	1	—	22.4	**Gemini Ultra 1.0** 1.5T trained on 30T tokens*	◆ Google DM API
3	—	—	22.4	**Gemini 1.5 Pro** 1.5T trained on 30T tokens*	◆ Google DM API
4	—	—	21.1	**Yi-XLarge** 2T trained on 20T tokens*	◆ 01-ai API
5	—	—	16.3	**Inflection-2.5** 1.2T on 20T tokens*	◆ Inflection AI API
6	2	1	15.9	**GPT-4 (family)** 1.7T trained on 13T tokens*	◆ OpenAI API
7	3	—	14.9	**ERNIE 4.0** 1T trained on 20T tokens*	◆ Baidu API
8	—	—	8.2	**SenseNova 5.0** 600B on 10T tokens	◆ SenseTime API

DATASETS

Now (Jun/2024)	6m ago (Dec/2023)	12m ago (Jun/2023)	Size (TB)	Dataset name Details	AI lab Language
1	1	—	130	**Gemini** 30T tokens in 130TB*	◆ Google DM Multilingual
2	2	—	125	**RedPajama-Data-v2** 30T tokens in 125TB	◆ Together AI Multilingual
3	3	1	86	**Piper monorepo** 37.9T tokens in 86TB	◆ Google Code
4	4	—	40	**Massive Never-ending BT Vast Chinese corpus** 30T/40TB	◆ MNBVC Chinese
5	—	—	44	**FineWeb** 15T tokens in 44TB	◆ HF English
6	5	2	40	**GPT-4** 13T tokens in 40TB*	◆ OpenAI English
7	—	—	31.5	**FineWeb-Edu-score-2** 5.4T tokens in 31.5TB	◆ HF English
8	6	—	27	**CulturaX** 6.3T tokens in 27TB	◆ UOregon Multilingual

Selected highlights only, some older models disregarded. * = estimates and hypothesis only based on current information. Alan D. Thompson. June 2024. https://lifearchitect.ai/

Large Language Models

https://lifearchitect.ai/models/

- LLMs are… LARGE!

BILLBOARD CHART FOR LANGUAGE MODELS JUN/2024

MODELS

Now (Jun/2024)	6m ago (Dec/2023)	12m ago (Jun/2023)	ALScore	Model name Details	AI lab Openness
1	—	—	29.8	**Claude 3 Opus** 2T trained on 40T tokens*	◆ Anthropic API
2	1	—	22.4	**Gemini Ultra 1.0** 1.5T trained on 30T tokens*	◆ Google DM API
3	—	—	22.4	**Gemini 1.5 Pro** 1.5T trained on 30T tokens*	◆ Google DM API
4	—	—	21.1	**Yi-XLarge** 2T trained on 20T tokens*	◆ 01-ai API
5	—	—	16.3	**Inflection-2.5** 1.2T on 20T tokens*	◆ Inflection AI API
6	2	1	15.9	**GPT-4 (family)** 1.7T trained on 13T tokens*	◆ OpenAI API
7	3	—	14.9	**ERNIE 4.0** 1T trained on 20T tokens*	◆ Baidu API
8	—	—	8.2	**SenseNova 5.0** 600B on 10T tokens	◆ SenseTime API

DATASETS

Now (Jun/2024)	6m ago (Dec/2023)	12m ago (Jun/2023)	Size (TB)	Dataset name Details	AI lab Language
1	1	—	130	**Gemini** 30T tokens in 130TB*	◆ Google DM Multilingual
2	2	—	125	**RedPajama-Data-v2** 30T tokens in 125TB	◆ Together AI Multilingual
3	3	1	86	**Piper monorepo** 37.9T tokens in 86TB	◆ Google Code
4	4	—	40	**Massive Never-ending BT Vast Chinese corpus** 30T/40TB	◆ MNBVC Chinese
5	—	—	44	**FineWeb** 15T tokens in 44TB	◆ HF English
6	5	2	40	**GPT-4** 13T tokens in 40TB*	◆ OpenAI English
7	—	—	31.5	**FineWeb-Edu-score-2** 5.4T tokens in 31.5TB	◆ HF English
8	6	—	27	**CulturaX** 6.3T tokens in 27TB	◆ UOregon Multilingual

Selected highlights only, some older models disregarded. * = estimates and hypothesis only based on current information. Alan D. Thompson. June 2024. https://lifearchitect.ai/

Large Language Models

https://lifearchitect.ai/models/

- LLMs are… LARGE!

BILLBOARD CHART FOR LANGUAGE MODELS JUN/2024

MODELS

Now (Jun/2024)	6m ago (Dec/2023)	12m ago (Jun/2023)	ALScore	Model name Details	AI lab Openness
1	—	—	29.8	**Claude 3 Opus** 2T trained on 40T tokens*	◆ Anthropic API
2	1	—			
3	—	—			
4	—	—		2T trained on 20T tokens*	API
5	—	—	16.3	**Inflection-2.5** 1.2T on 20T tokens*	◆ Inflection AI API
6	2	1	15.9	**GPT-4 (family)** 1.7T trained on 13T tokens*	◆ OpenAI API
7	3	—	14.9	**ERNIE 4.0** 1T trained on 20T tokens*	◆ Baidu API
8	—	—	8.2	**SenseNova 5.0** 600B on 10T tokens	◆ SenseTime API

DATASETS

Now (Jun/2024)	6m ago (Dec/2023)	12m ago (Jun/2023)	Size (TB)	Dataset name Details	AI lab Language
1	1	—	130	**Gemini** 30T tokens in 130TB*	◆ Google DM Multilingual
				ata-v2 125TB	ether AI ultilingual
				bo 86TB	Google Code
				r-ending BT Vast **Chinese corpus** 30T/40TB	◆ MNBVC Chinese
5	—	—	44	**FineWeb** 15T tokens in 44TB	◆ HF English
6	5	2	40	**GPT-4** 13T tokens in 40TB*	◆ OpenAI English
7	—	—	31.5	**FineWeb-Edu-score-2** 5.4T tokens in 31.5TB	◆ HF English
8	6	—	27	**CulturaX** 6.3T tokens in 27TB	◆ UOregon Multilingual

Selected highlights only, some older models disregarded. * = estimates and hypothesis only based on current information. Alan D. Thompson. June 2024. https://lifearchitect.ai/

But they're still just playing a game of "Guess the next word"

Transformers

- More than meets the eye!

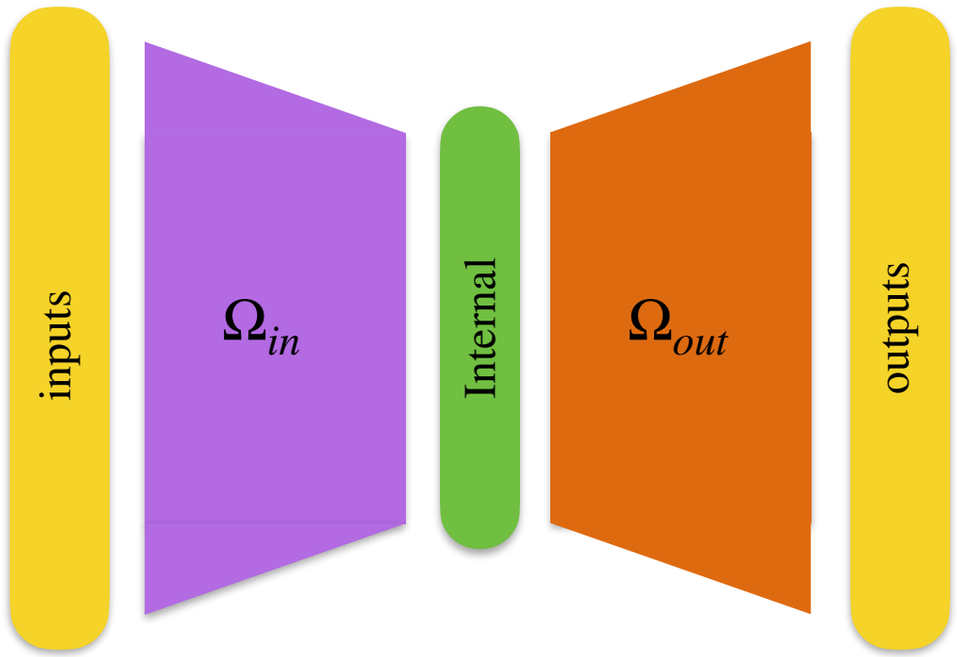
Transformers

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.

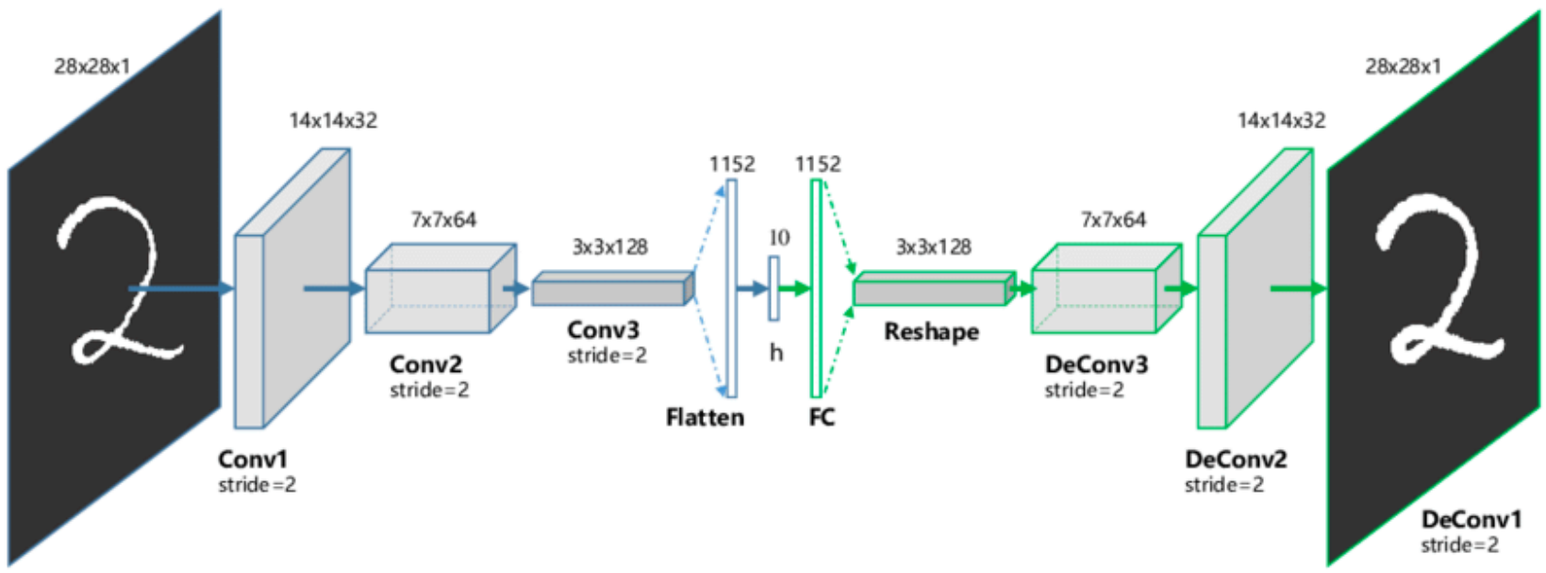


- Think of it as "zipping" and "unzipping" the input values. 

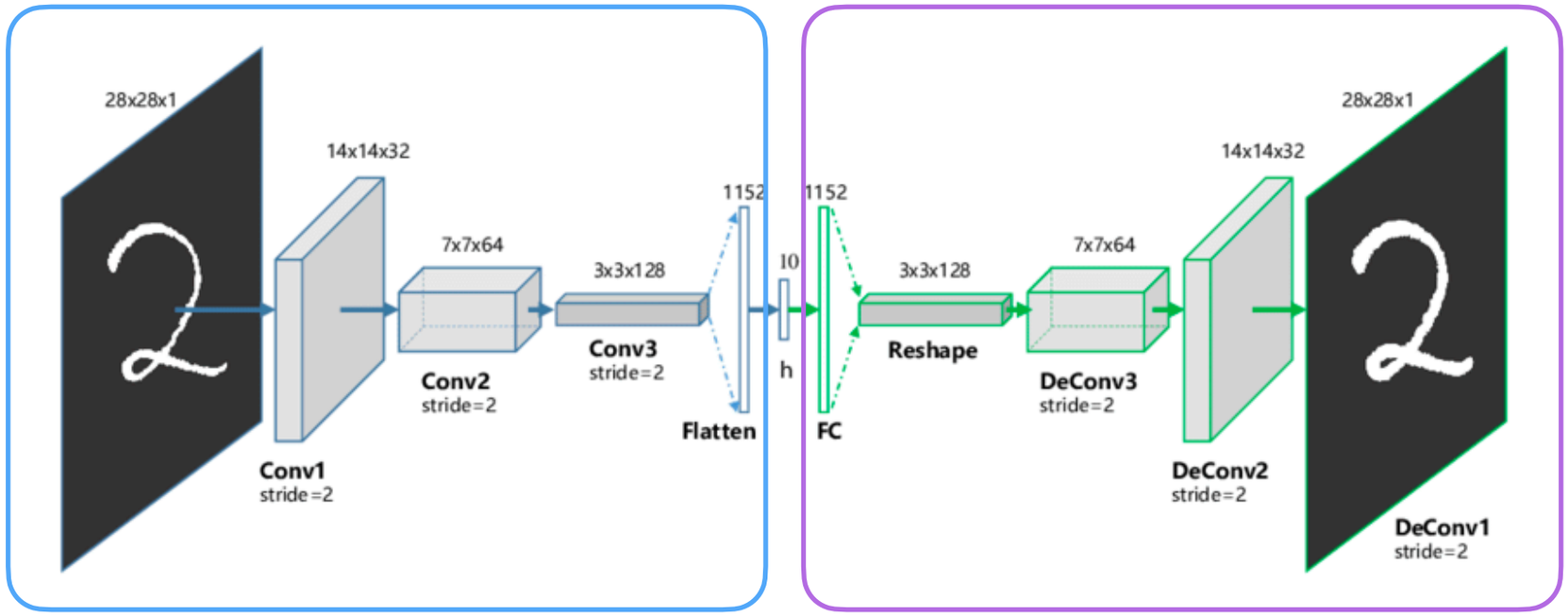
Auto-Encoders

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



Encoder

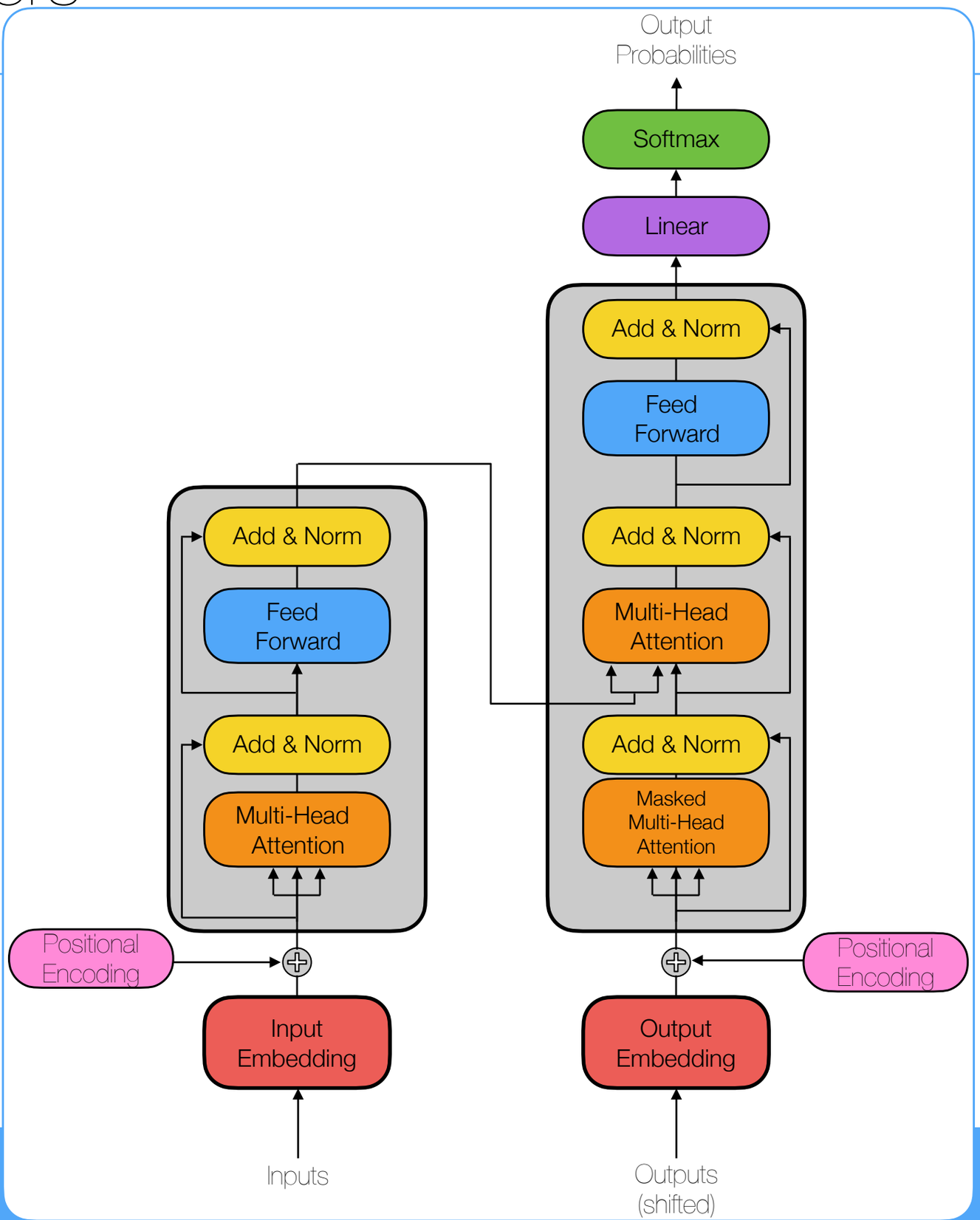
Input -> Internal Representation

Decoder

Internal Representation -> Output

Transformers

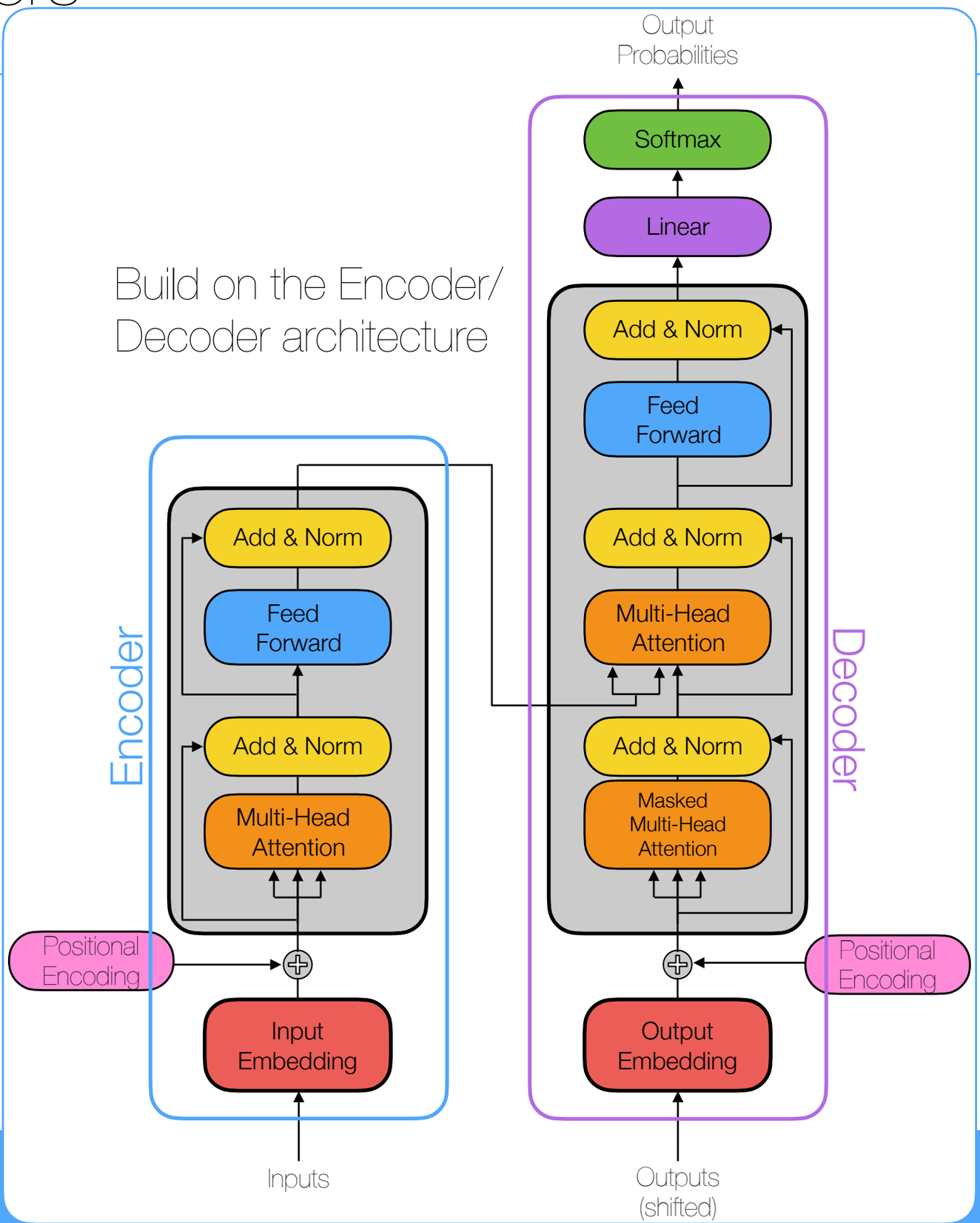
arXiv:1706.03762



Transformers

arXiv:1706.03762

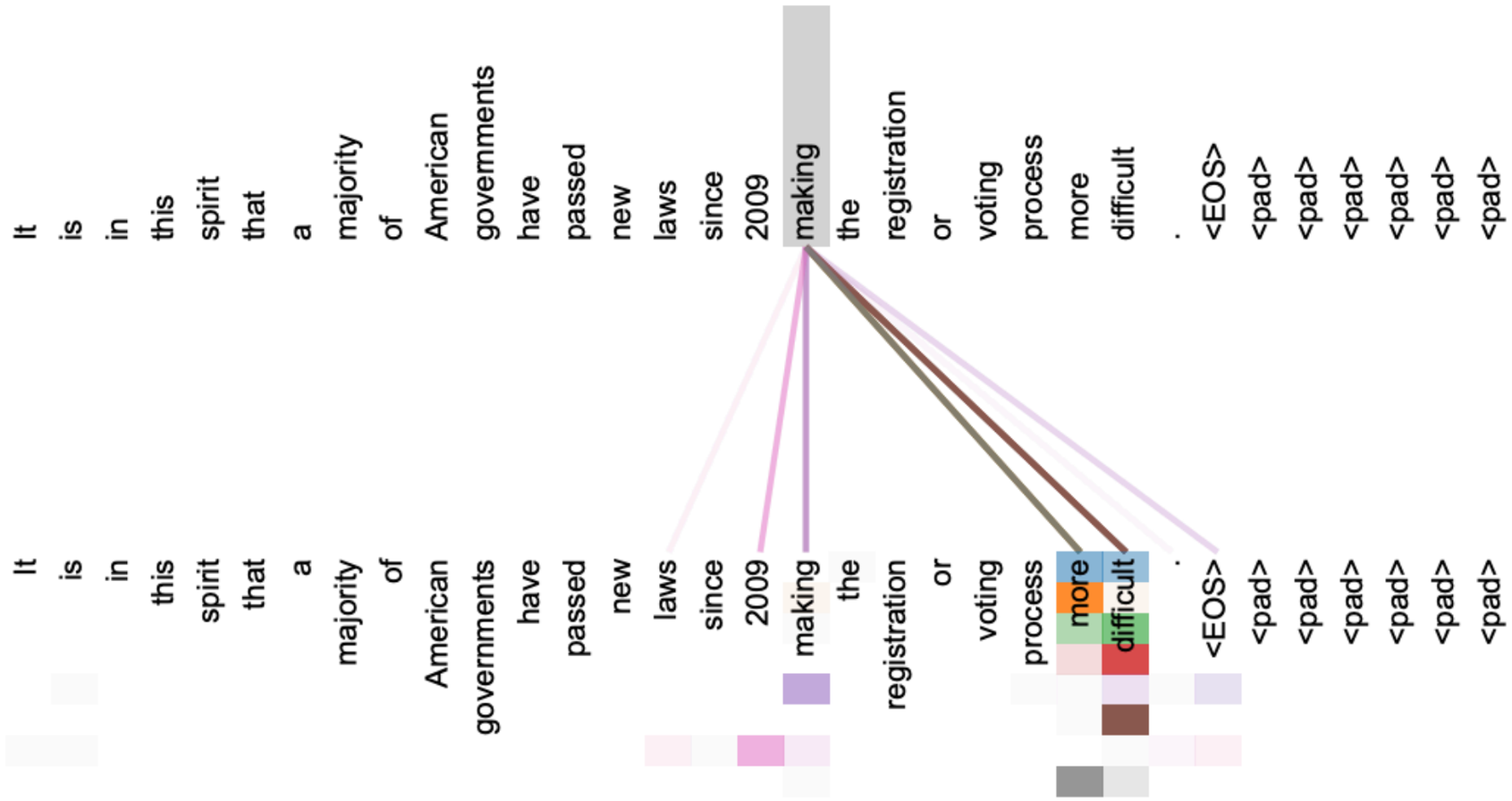
Build on the Encoder/
Decoder architecture



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)



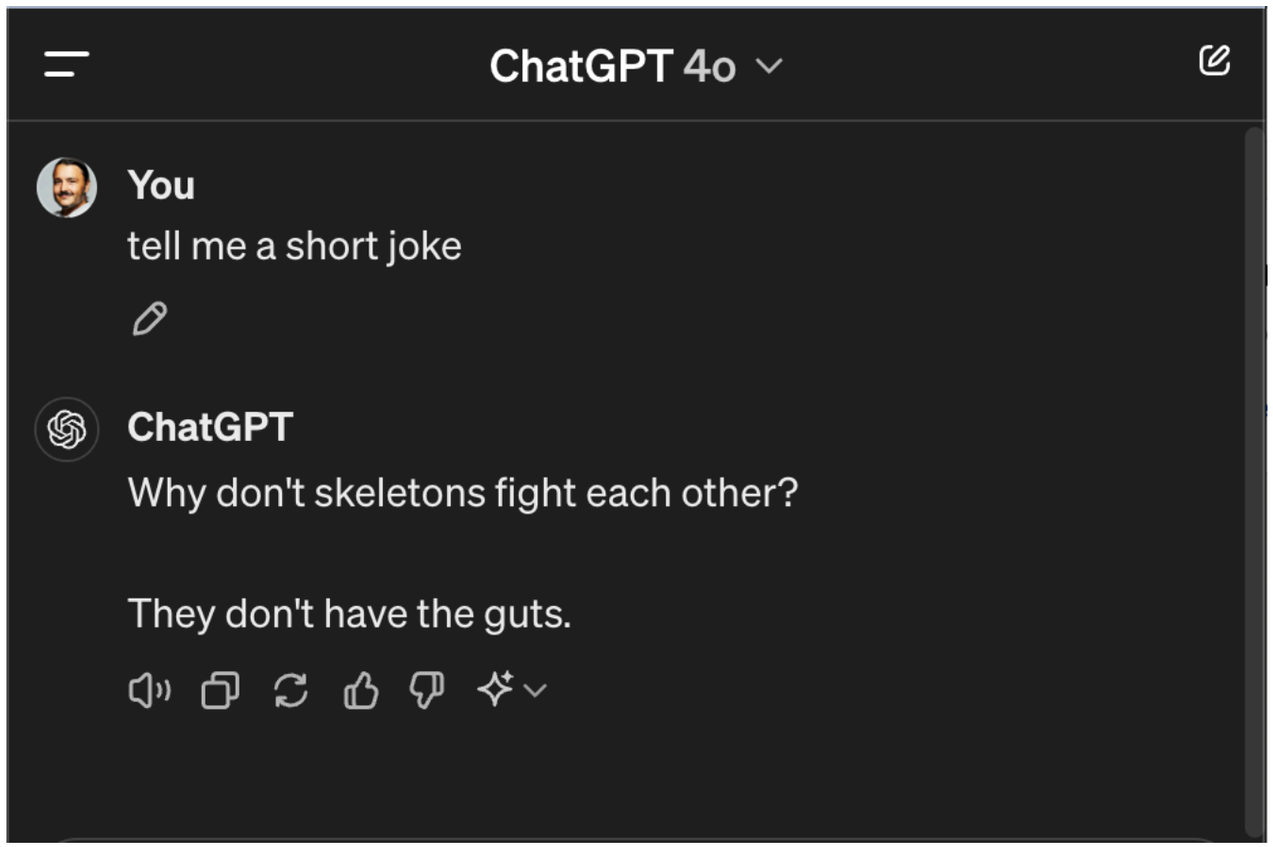
- Weights indicate the importance of each word relative to the current one

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 (~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

Hallucinations

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

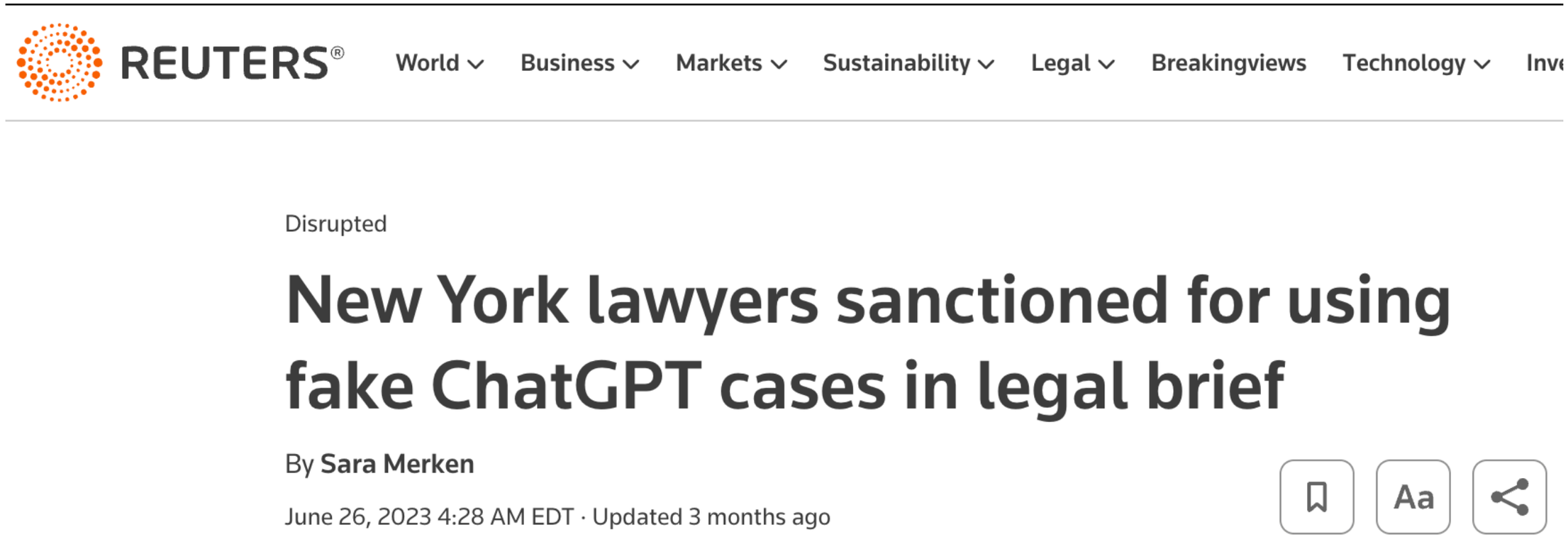
- LLMs are just trying to guess the next word with limited or no understanding of what they're "talking about"

- 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:



REUTERS® 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

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
  - May 2024 - GPT-4o

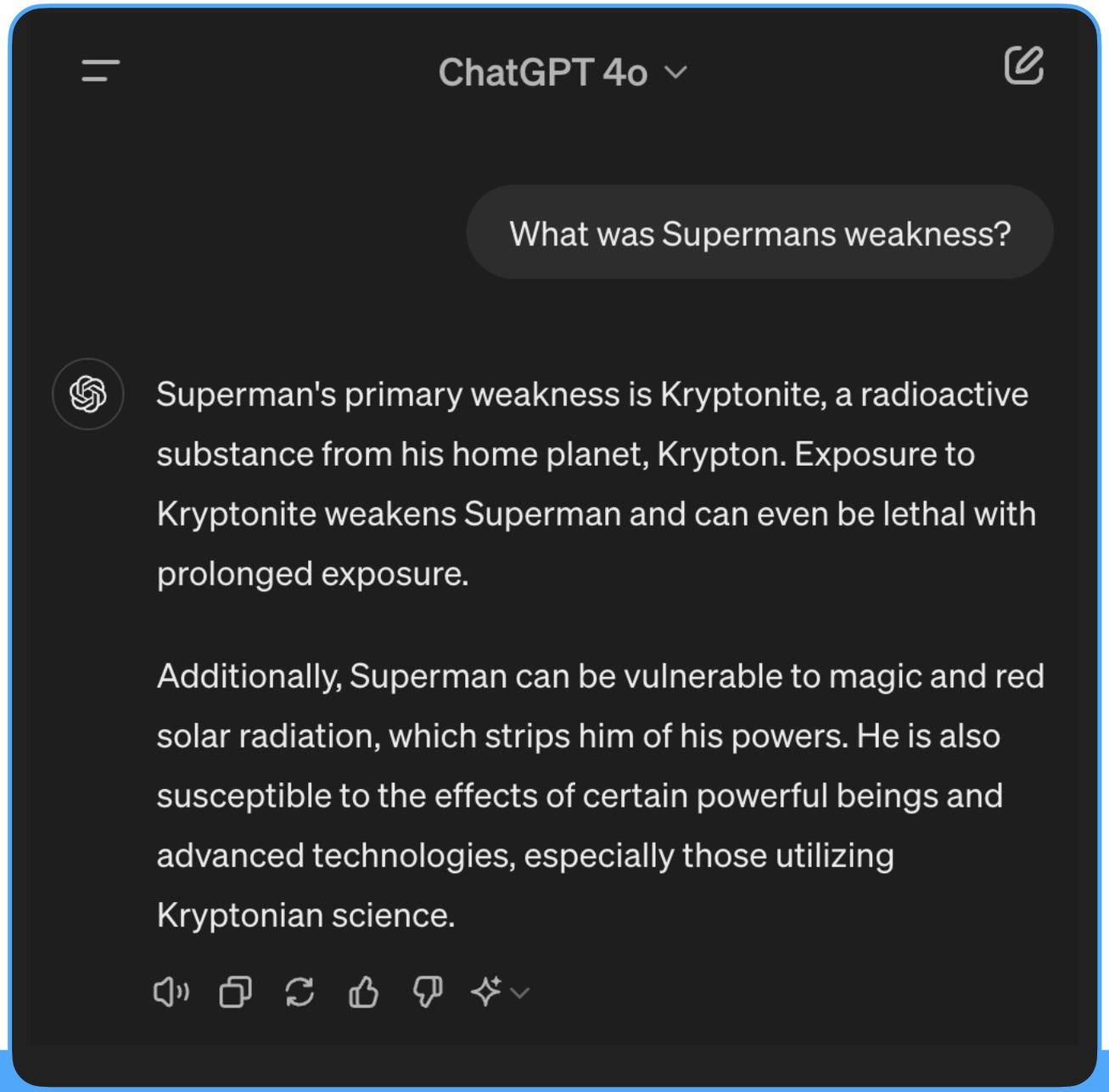
Generative AI and OpenAI

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:



Basic Usage

- Programmatically, things look a bit different
- The basic API call is chat.completions.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

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
        {
            "role": "user",
            "content": "What was Superman's weakness?"
        },
    ]
)
```

- response is a openai.types.chat.chat_completion.ChatCompletion object with several relevant fields:
  - model - the model used

Basic Usage

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
        {
            "role": "user",
            "content": "What was Superman's weakness?"
        },
    ]
)
```

- response is a openai.types.chat.chat_completion.ChatCompletion object with several relevant fields:
  - model - the model used
  - choices - a list of Choice objects that you can treat as named tuples

Basic Usage

```python
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
        {
            "role": "user",
            "content": "What was Superman's weakness?"
        },
    ]
)
```

- response is a openai.types.chat.chat_completion.ChatCompletion object with several relevant fields:
  - model - the model used
  - choices - a list of Choice objects that you can treat as named tuples
    - message - A named tuple containing the output
      - content - The text of the output
      - finish_reason - The reason text generation stopped
      - role - The role corresponding to the output

The meat of the answer generated by the model

Basic Usage

```python
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
        {
            "role": "user",
            "content": "What was Superman's weakness?"
        },
    ]
)
```

- response is a openai.types.chat.chat_completion.ChatCompletion object with several relevant fields:
  - model - the model used
  - choices - a list of Choice objects that you can treat as named tuples
    - message - A named tuple containing the output
      - content - The text of the output
      - finish_reason - The reason text generation stopped
      - role - The role corresponding to the output

The output generated, formatted as the prompt message

Basic Usage

```python
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
        {
            "role": "user",
            "content": "What was Superman's weakness?"
        },
    ]
)
```

- response is a openai.types.chat.chat_completion.ChatCompletion object with several relevant fields:
  - model - the model used
  - choices - a list of Choice objects that you can treat as named tuples
    - message - A named tuple containing the output
      - content - The text of the output
      - finish_reason - The reason text generation stopped
      - role - The role corresponding to the output

Basic Usage

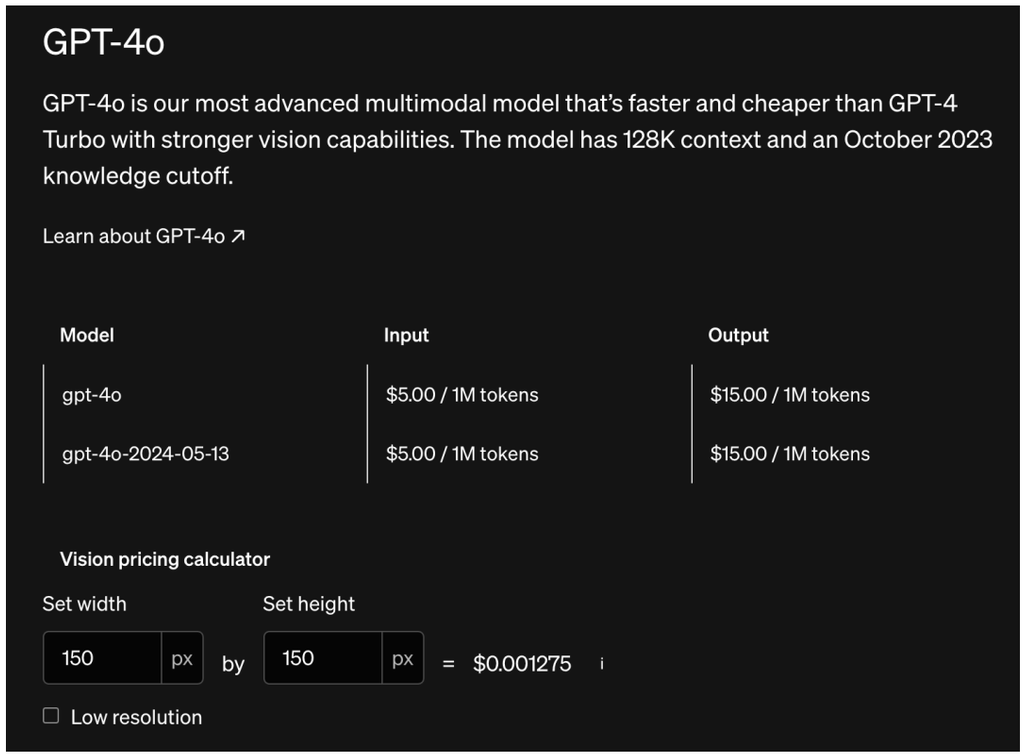
```python
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
        {
            "role": "user",
            "content": "What was Superman's weakness?"
        },
    ]
)
```

- response is a openai.types.chat.chat_completion.ChatCompletion object with several relevant fields:
  - model - the model used
  - choices - a list of Choice objects that you can treat as named tuples
    - message - A named tuple containing the output
      - content - The text of the output
      - finish_reason - The reason text generation stopped
      - role - The role corresponding to the output
  - usage - a CompletionUsage object containing the number of tokens used

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



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 10x from one model to another

GPT-4o

GPT-4o is our most advanced multimodal model that's faster and cheaper than GPT-4 Turbo with stronger vision capabilities. The model has 128K context and an October 2023 knowledge cutoff.

Learn about GPT-4o ↗

Model	Input	Output
gpt-4o	$5.00 / 1M tokens	$15.00 / 1M tokens
gpt-4o-2024-05-13	$5.00 / 1M tokens	$15.00 / 1M tokens

Vision pricing calculator

Set width: 150 px by Set height: 150 px = $0.001275

Low resolution

GPT-3.5 Turbo

GPT-3.5 Turbo is our fast and inexpensive model for simpler tasks.
`gpt-3.5-turbo-0125` is the flagship model of this family, supports a 16K context window 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
gpt-3.5-turbo-0125	$0.50 / 1M tokens	$1.50 / 1M tokens
gpt-3.5-turbo-instruct	$1.50 / 1M tokens	$2.00 / 1M 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 10x from one model to another

GPT-4o

GPT-4o is our most advanced multimodal model that's faster and cheaper than GPT-4 Turbo with stronger vision capabilities. The model has 128K context and an October 2023 knowledge cutoff.

Learn about GPT-4o ↗

Model	Input	Output
gpt-4o	$5.00 / 1M tokens	$15.00 / 1M tokens
gpt-4o-2024-05-13	$5.00 / 1M tokens	$15.00 / 1M tokens

Vision pricing calculator

Set width 150 px by Set height 150 px = $0.001275

Low resolution

Input and output tokens are also priced differently

GPT-3.5 Turbo

GPT-3.5 Turbo is our fast and inexpensive model for simpler tasks.
`gpt-3.5-turbo-0125` is the flagship model of this family, supports a 16K context window 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
gpt-3.5-turbo-0125	$0.50 / 1M tokens	$1.50 / 1M tokens
gpt-3.5-turbo-instruct	$1.50 / 1M tokens	$2.00 / 1M tokens

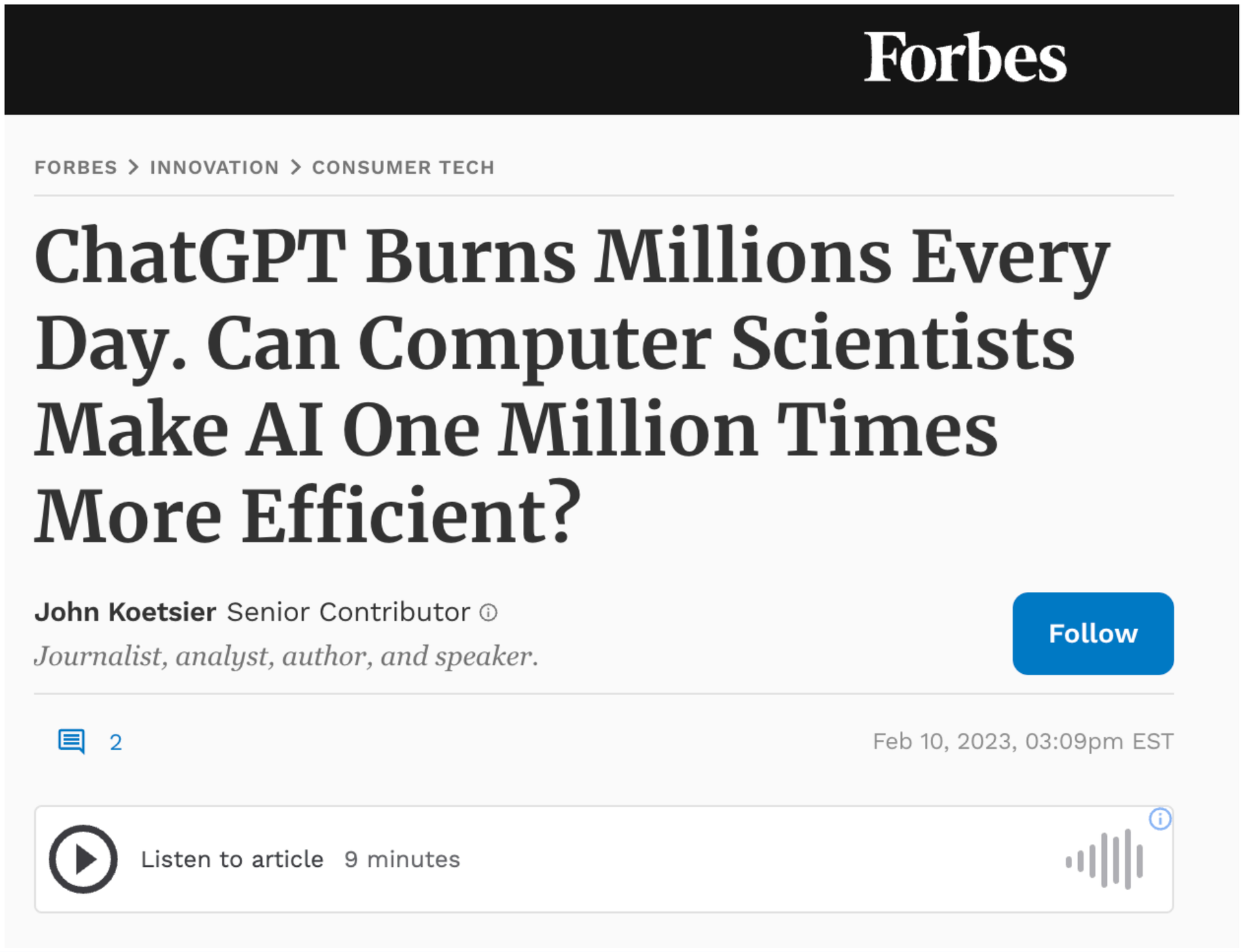
Rate Limits

platform.openai.com/account/rate-limits

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

MODEL	TOKEN LIMITS	REQUEST AND OTHER LIMITS	BATCH QUEUE LIMITS
gpt-4o	450,000 TPM	5,000 RPM	1,350,000 TPD
gpt-3.5-turbo	80,000 TPM	5,000 RPM	400,000 TPD
gpt-4	40,000 TPM	5,000 RPM	200,000 TPD
gpt-4-turbo	450,000 TPM	500 RPM	1,350,000 TPD
text-embedding-3-small	1,000,000 TPM	5,000 RPM	20,000,000 TPD
dall-e-3		7 images per minute	
tts-1		50 RPM	
whisper-1		50 RPM	

ChatGPT infrastructure is expensive



Forbes

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

```python
response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "user", "content": "What are the different kinds of Kryptonite?"},
    ],
    n=3
)
```

- We can also request multiple outputs simultaneously using the n argument to chat.completions.create().

- Each completion is listed in the responses choices array with a sequential index

Code - GPT Models

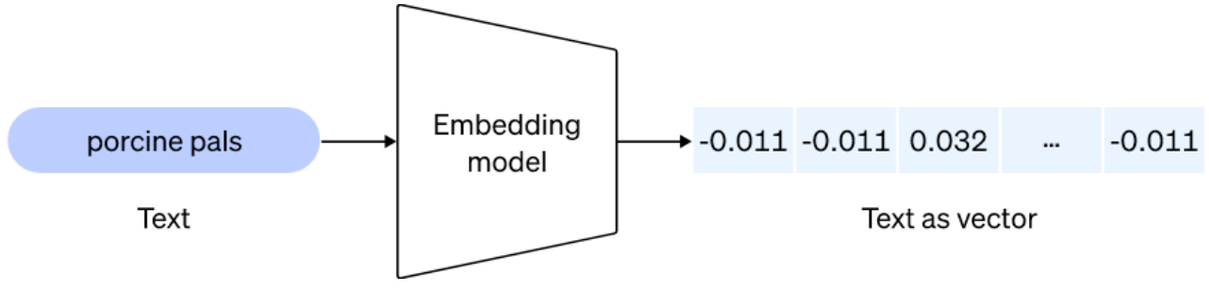
https://github.com/DataForScience/OpenAI

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:

```
response = client.embeddings.create(
    model="text-embedding-ada-002",
    input="Your text goes here",
    )
```

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

```
response.data[0].embedding
```

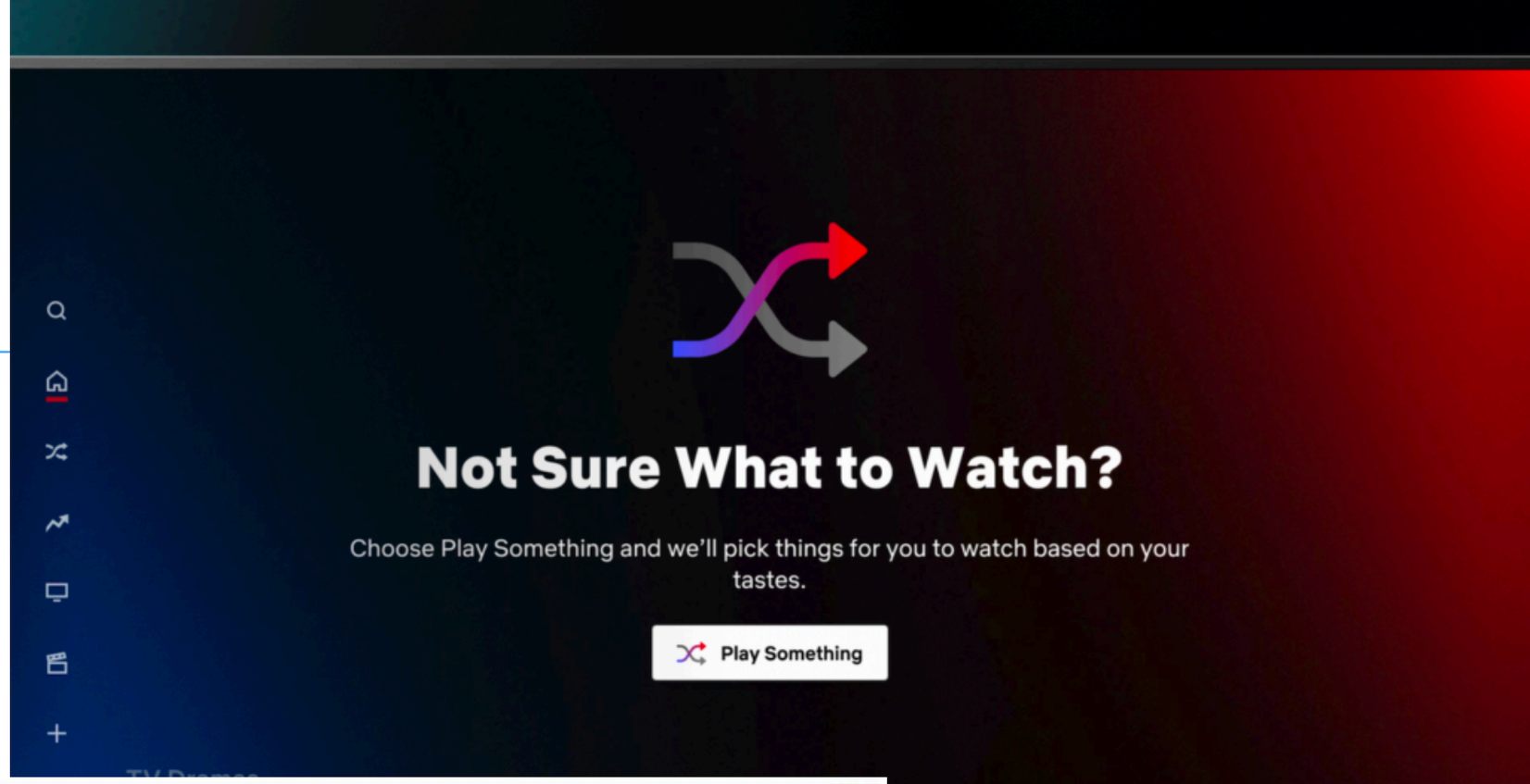
Embeddings and Encodings

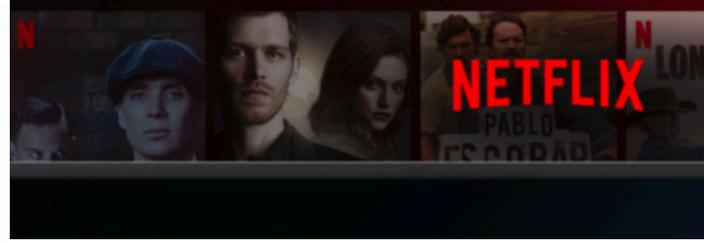
- Embedding models work at the token level
- Embedding models are trained on a specific encoding of the tokens generated by a given tokenizer, so we must be careful to use the correct encoding
- To get the encoding associated with a specific embedding model, we use:

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

- And we must pass this encoding as an argument to the get_embedding function

Recommendations





amazon.com

Recommended for You

Amazon.com has new recommendations for you based on items you purchased or told us you own.

Google Apps Deciphered: Compute in the Cloud to Streamline Your Desktop

Google Apps Administrator Guide: A Private-Label Web Workspace

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

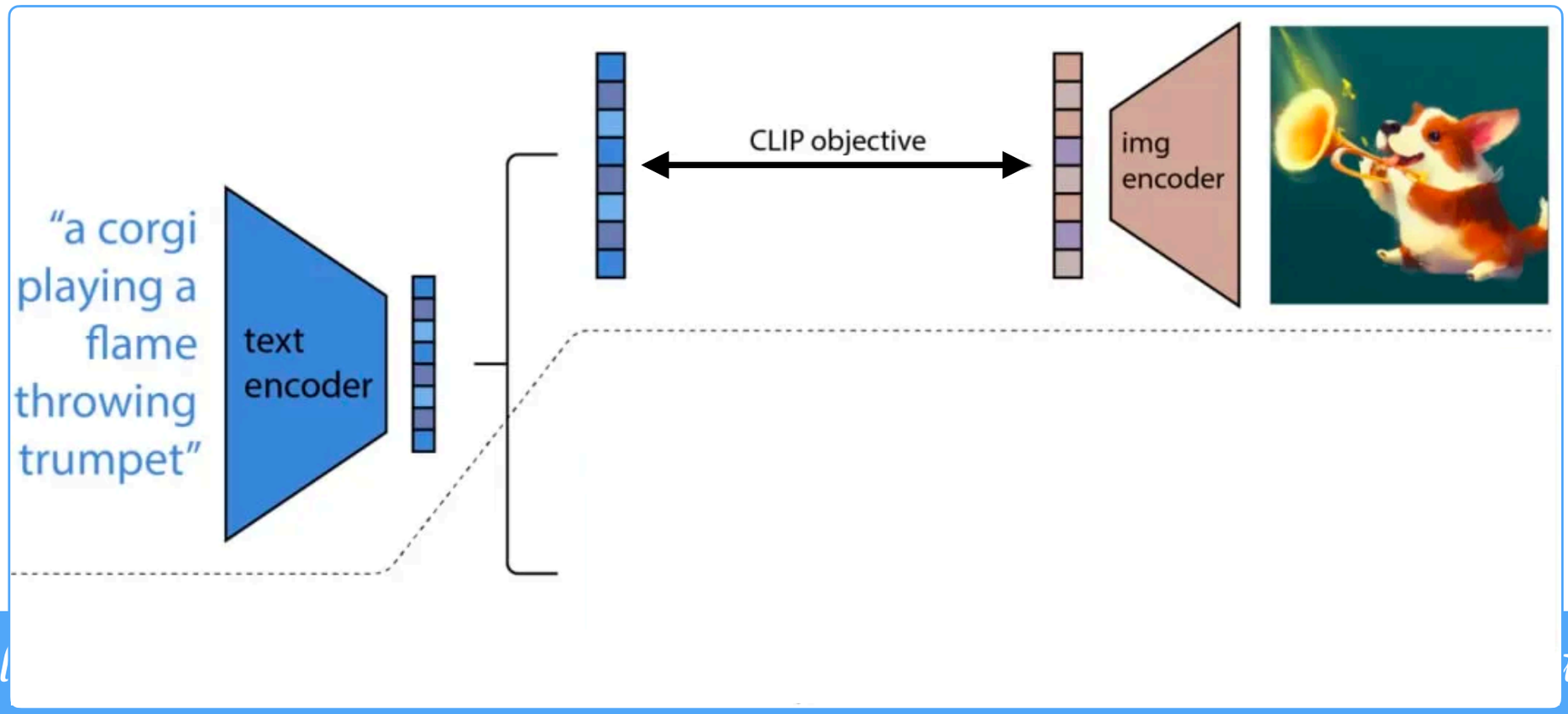


Image Models

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

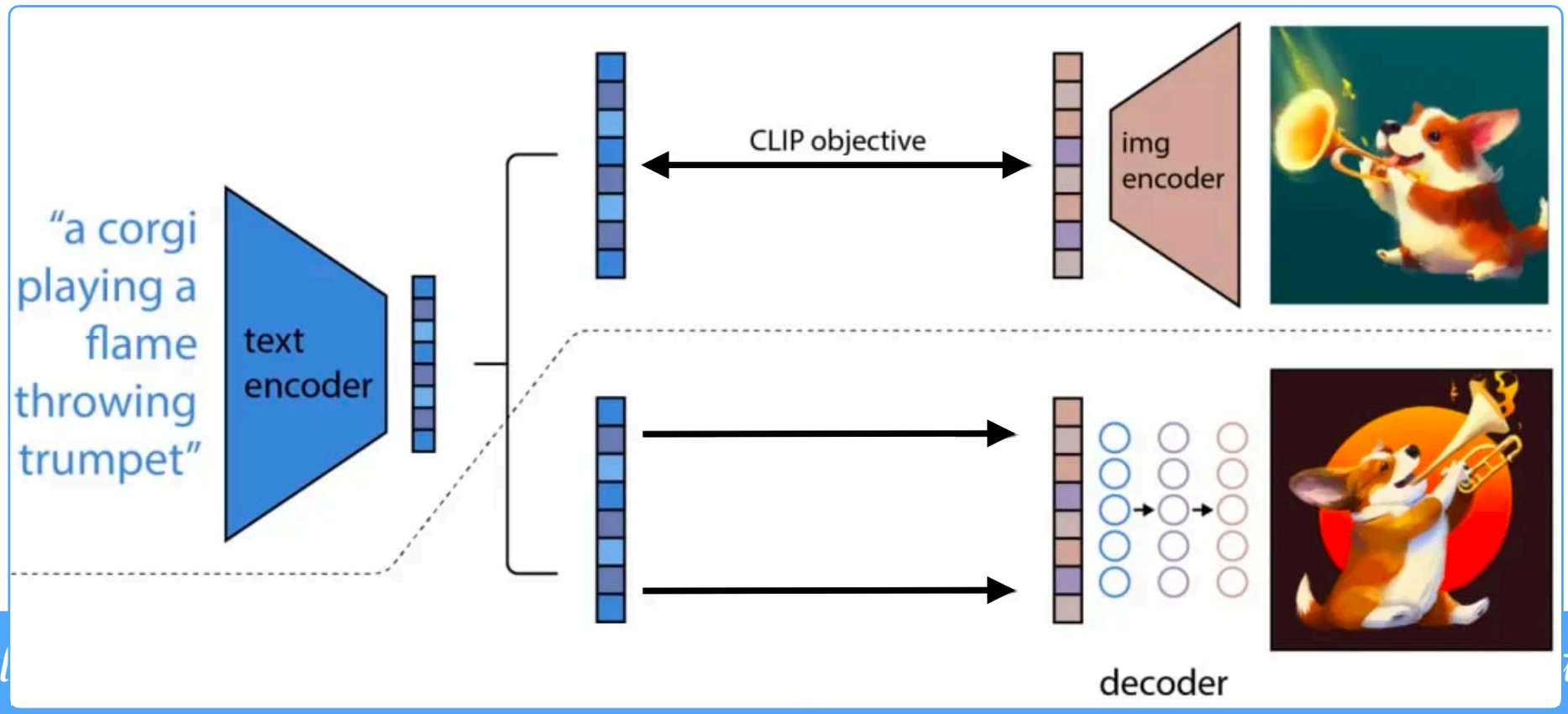
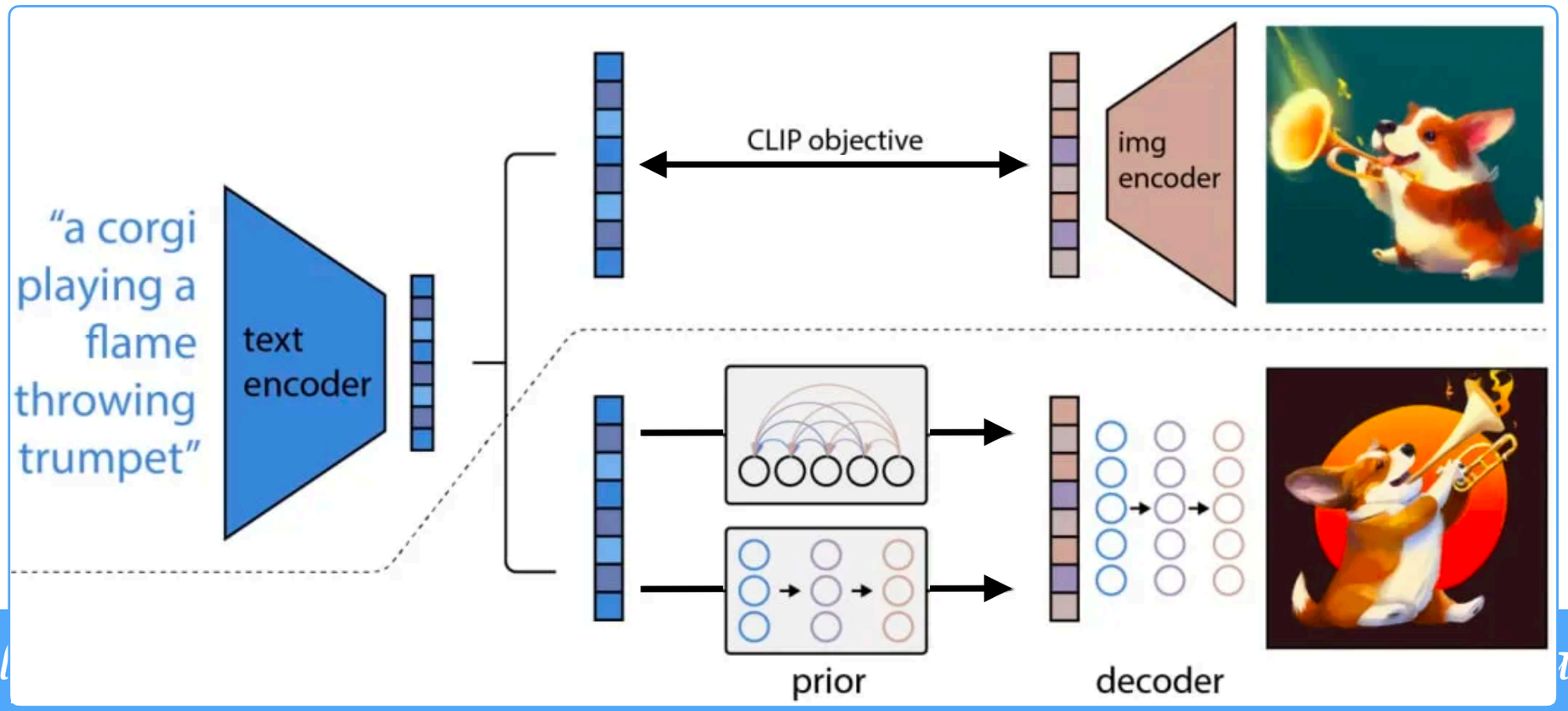


Image Models

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:

```
response = client.images.generate(
    model='dall-e-3',
    prompt=prompt,
    n=1,
    size="1024x1024",
    response_format="url",
)
```

- 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

```python
response = client.images.create_variation(
    image=open('images/rainbow.png', 'rb'),
    n=3,
    size="1024x1024",
    response_format="url",
)
```

- 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

```python
response = client.images.edit(
    image=open(image_filename, "rb"),
    mask=open(mask_filename, "rb"),
    prompt=prompt,
    n=1,
    size="1024x1024",
    response_format="url",
)
```

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

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": """You're a brilliant graphic designer.
        Generate an precise prompt for DALL-E that has at most 1000 characters."""
        },
        {"role": "user", "content": prompt},
    ]
)
```

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

```
response = client.images.generate(
    prompt=expanded_prompt[:1000],
    n=3,
    size="1024x1024",
    response_format="url",
)
```

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 your 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:

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

- Produce the expected result

```
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

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

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

data4sci.substack.com

Interactive Data Visualization with Python

Aug 7, 2024 - 10am-2pm (PST)

ChatGPT and Competing LLMs

Aug 13, 2024 - 10am-2pm (PST)

LLMs for Data Science

Sept 19, 2024 - 10am-2pm (PST)

Bruno Gonçalves

https://data4sci.com

info@data4sci.com

https://data4sci.com/call

VIDEO

Times Series Analysis for Everyone

★★★★★ 1 review

By Bruno Goncalves

https://bit.ly/Timeseries_LL

TIME TO COMPLETE:
6h

TOPICS:
Time Series

PUBLISHED BY:
Pearson

PUBLICATION DATE:
November 2021

Start

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.

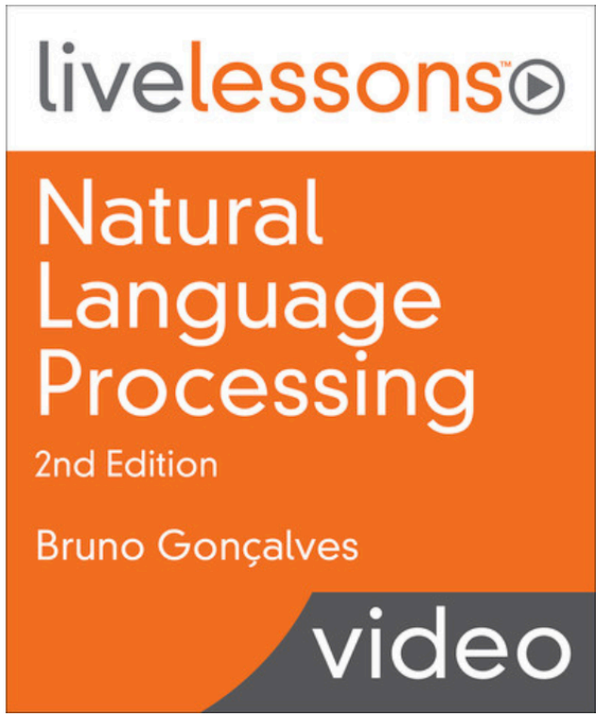
VIDEO

Natural Language Processing, 2nd Edition

Write the first review

By Bruno Goncalves

https://bit.ly/NLP_LL



TIME TO COMPLETE:
5h 23m

TOPICS:
Natural Language Processing

PUBLISHED BY:
Addison-Wesley Professional

PUBLICATION DATE:
October 2021

Start

5 Hours of Video Instruction

Overview

Natural Language Processing LiveLessons covers thefundamentals of Natural Language Processing in a simple and intuitive way,empowering you to add NLP to your toolkit. Using the powerful NLTK package, itgradually moves from the basics of text representation, cleaning, topicdetection, regular expressions, and sentiment analysis before moving on to theKeras deep learning framework to explore more advanced topics such as textclassification and sequence-to-sequence models. After successfully completingthese lessons you'll be equipped with a fundamental and practical understandingof state-of-the-art Natural Language Processing tools and algorithms.

END