LLMs for Data Science

Bruno Gonçalves

www.data4sci.com/newsletter

data4sci.substack.com

https://github.com/DataForScience/LLM4DS

Question

https://github.com/DataForScience/LLM4DS

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

Question

https://github.com/DataForScience/LLM4DS

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

Question

https://github.com/DataForScience/LLM4DS

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

Events

data4sci.substack.com

LangChain for Generative AI Pipelines

Sept 25, 2024 - 10am-2pm (PST)

Generative AI with the OpenAI API for Developers

Oct 9, 2024 - 10am-2pm (PST)

ChatGPT and Competing LLMs

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

Bruno Gonçalves

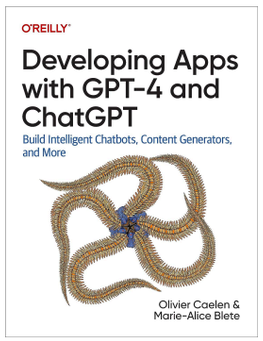
https://data4sci.com

info@data4sci.com

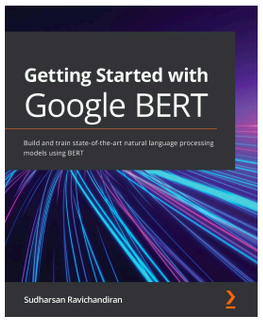
https://data4sci.com/call

References

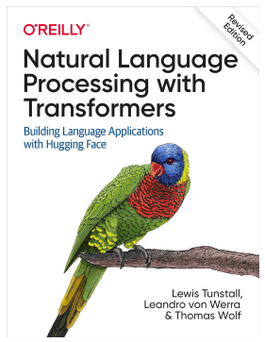
https://github.com/DataForScience/LLM4DS



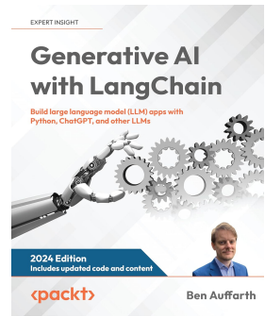
https://amzn.to/3RHRkQa



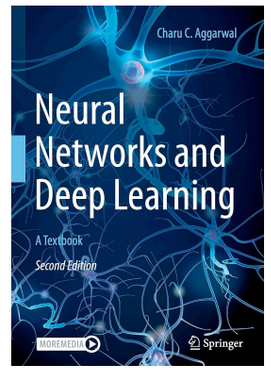
https://amzn.to/48u9qu5



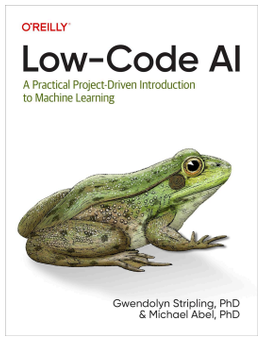
https://amzn.to/3UAaetS



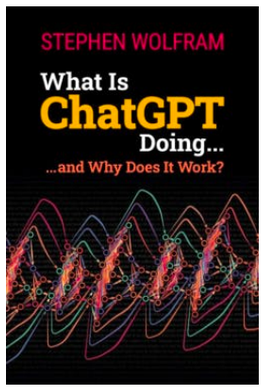
https://amzn.to/3VquiPj



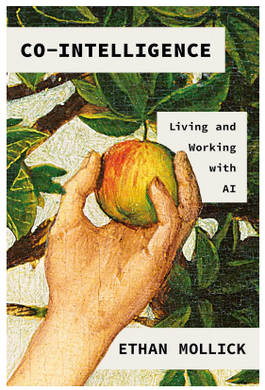
https://amzn.to/48rZn9X



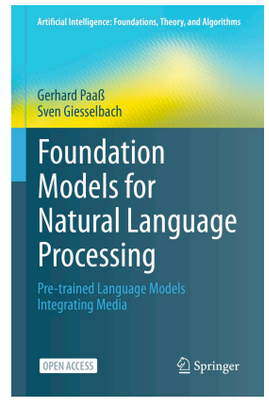
https://amzn.to/3UC7b4k



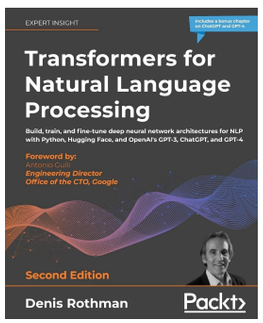
https://amzn.to/3LBRdBY



https://amzn.to/3KcLIsf



https://amzn.to/3P3qznx



https://amzn.to/46kOo02

Table of Contents

1. Generative AI for Data Science

Generative AI

- Any algorithm that can create new content (text, images, sound, code, etc) based on the training data.
- Common approaches:
    - Generative Adversarial Networks (GANs)
    - Variational Autoencoders (VAEs)
    - Transformers
- Applications:
    - Text Generation: ChatGPT, Claude, LLama, etc
    - Image Generation: DALL-E, Midjourney, etc
    - Audio: Whisper, Conformer, Wav2Vec, etc
    - Synthetic Data: ChatGPT, BERT, etc
    - Code Generation: GitHub Copilot, Cursor, 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:
  - Machine translation
  - Summarization
  - Sentiment analysis
  - Chatbots and Virtual Assistants
  - Content Generation
  - etc…

Large Language Models

- Large Language Models (LLMs) are Deep Learning models trained on vast amounts of text data that are able to learn the complexities of language, including syntax, semantics, and context.

- Architecture:

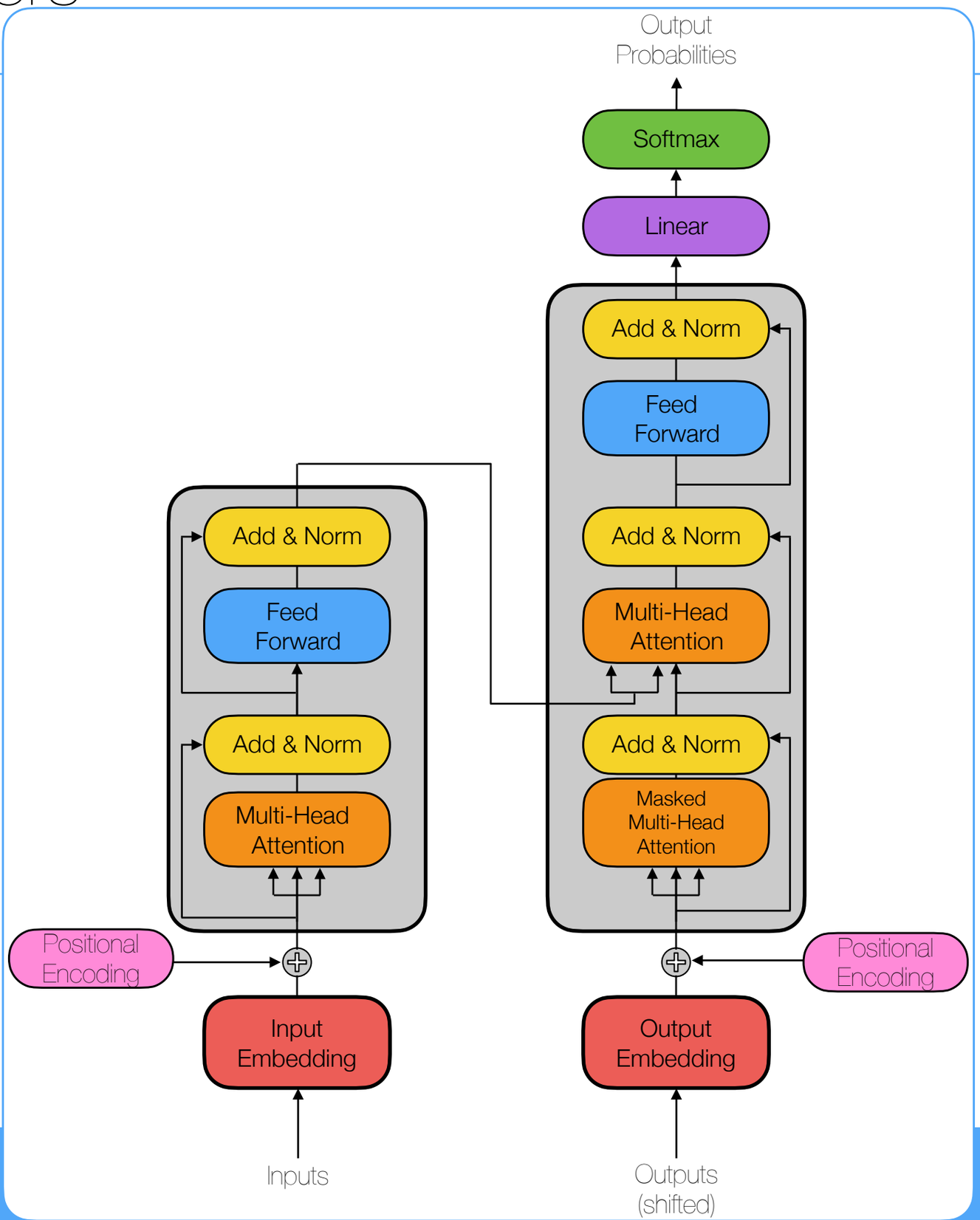
    - Transformers: The backbone architecture of most LLMs, which uses self-attention mechanisms to process and generate text efficiently. Key examples include GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers), and T5 (Text-to-Text Transfer Transformer).

    - Self-Attention: Allows the model to weigh the importance of different words in a sentence, making it capable of understanding context and relationships within the text.

    - Tokenization: Text is split into smaller units (tokens), which can be words, characters, or subwords. The model predicts the next token in a sequence based on previous ones.

Transformers

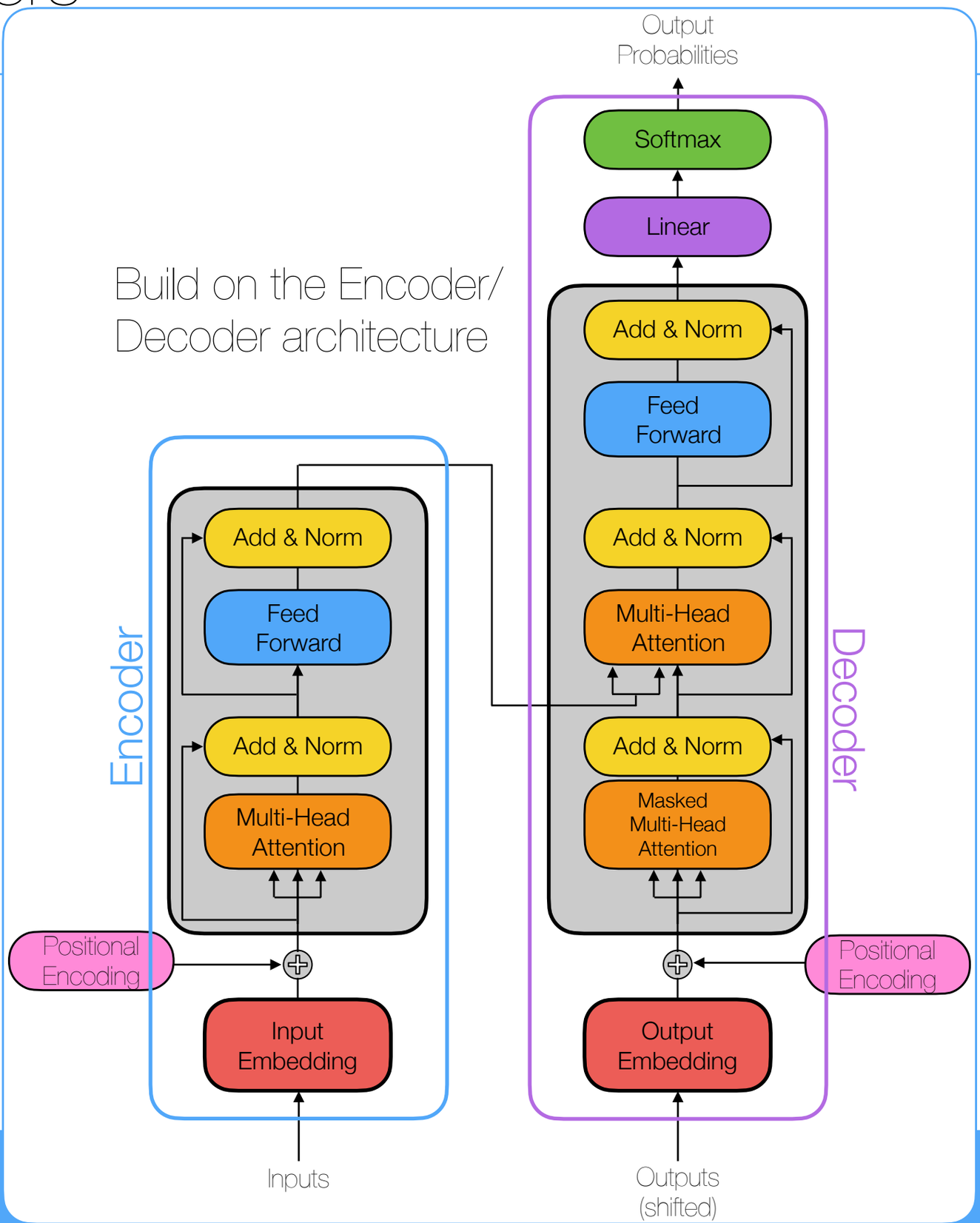
arXiv:1706.03762



Transformers

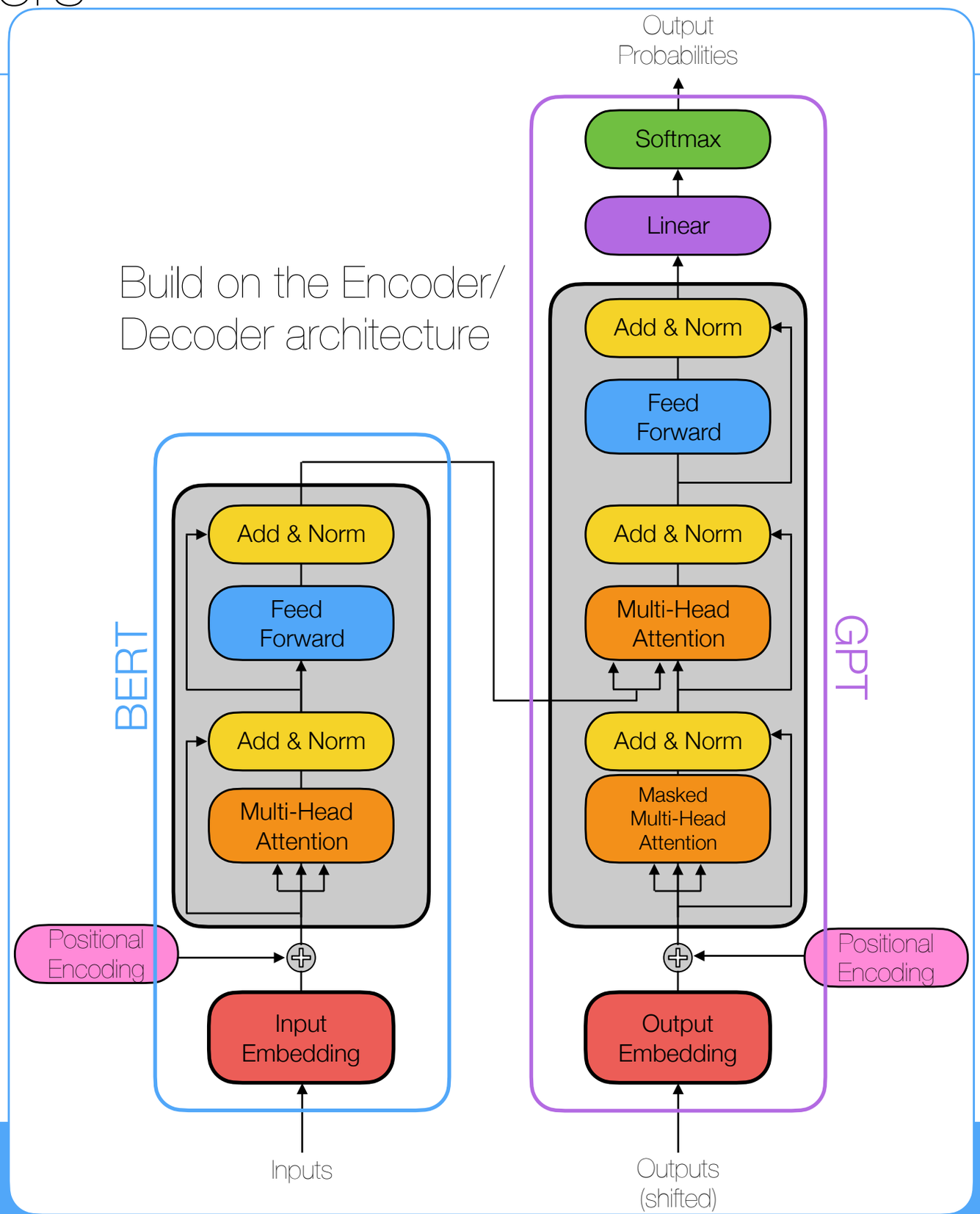
arXiv:1706.03762

Build on the Encoder/
Decoder architecture



Transformers

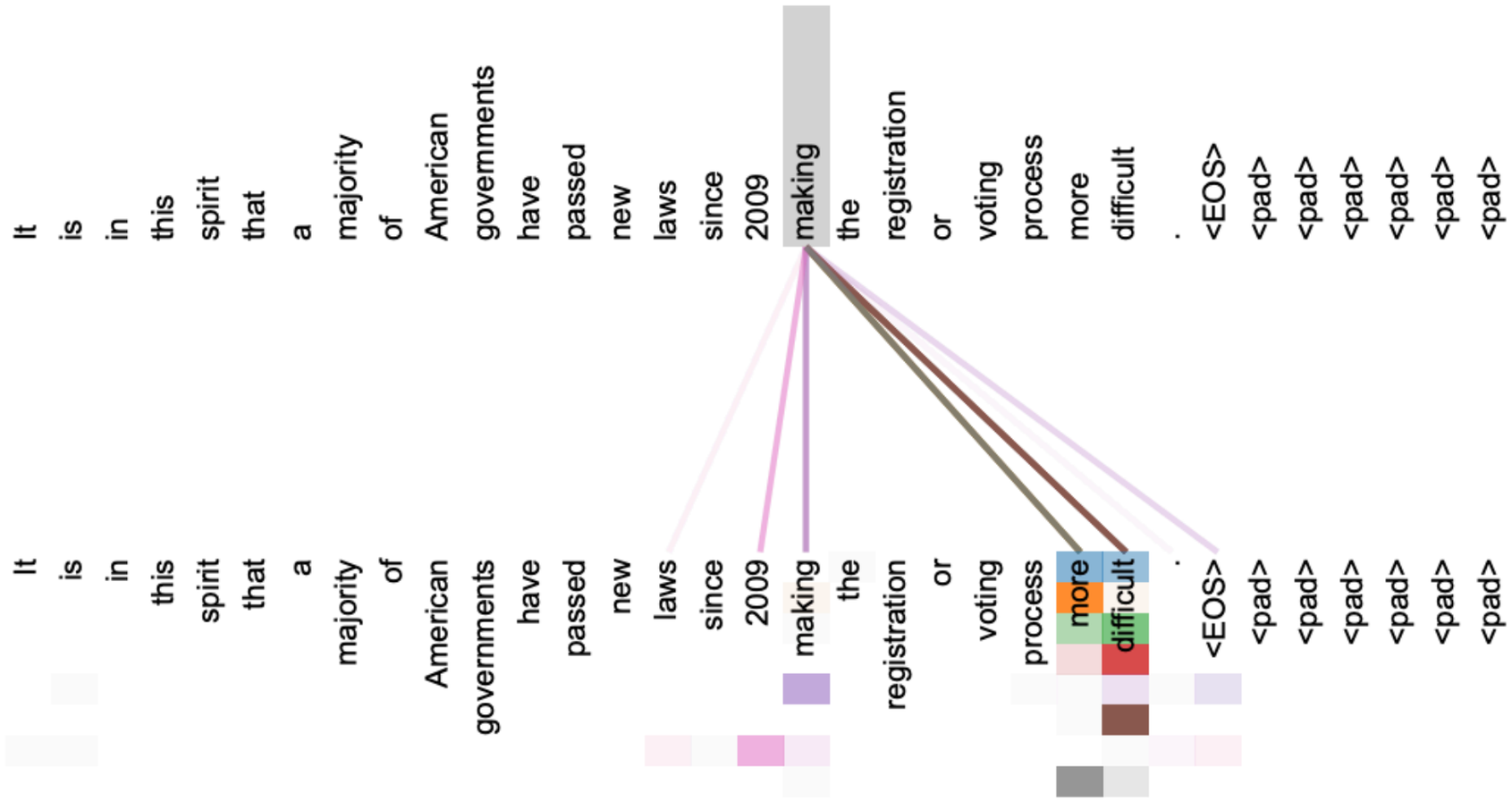
arXiv:1706.03762



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

Hallucinations

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

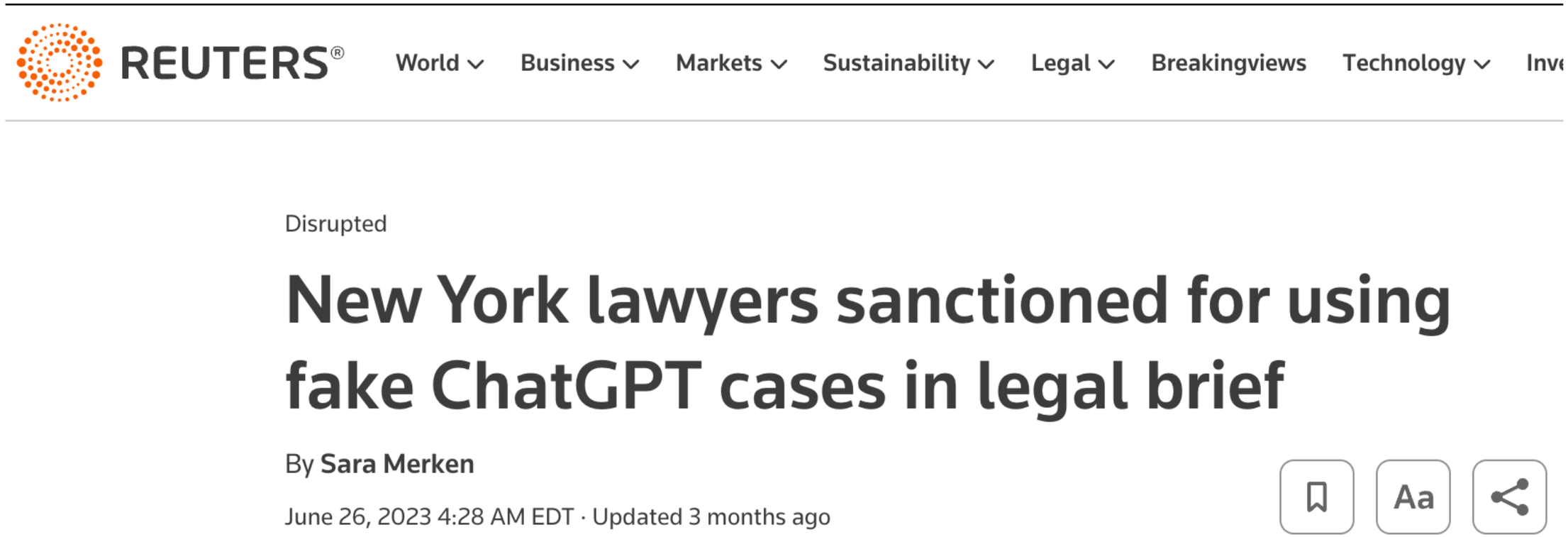
- LLMs are just trying to guess the next word with limited or no understanding of what they're "talking about"

- 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

arXiv:2409.05746 (2024)

LLMs Will Always Hallucinate, and We Need to Live With This

Sourav Banerjee*
DataLabs
United We Care
sb@unitedwecare.com

Ayushi Agarwal
DataLabs
United We Care
ayushi@unitedwecare.com

Saloni Singla
DataLabs
United We Care
saloni@unitedwecare.com

September 10, 2024

Abstract

As Large Language Models become more ubiquitous across domains, it becomes important to examine their inherent limitations critically. This work argues that hallucinations in language models are not just occasional errors but an inevitable feature of these systems. We demonstrate that hallucinations stem from the fundamental mathematical and logical structure of LLMs. It is, therefore, impossible to eliminate them through architectural improvements, dataset enhancements, or fact-checking mechanisms. Our analysis draws on computational theory and Gödel's First Incompleteness Theorem, which references the undecidability of problems like the Halting, Emptiness, and Acceptance Problems. We demonstrate that every stage of the LLM process—from training data compilation to fact retrieval, intent classification, and text generation—will have a non-zero probability of producing hallucinations. This work introduces the concept of "Structural Hallucinations" as an intrinsic nature of these systems. By establishing the mathematical certainty of hallucinations, we challenge the prevailing notion that they can be fully mitigated.

arXiv:2409.05746 (2024)

LLMs Will Always Hallucinate, and We Need to Live With This

Sourav Banerjee*
DataLabs
United We Care
sb@unitedwecare.com

Ayushi Agarwal
DataLabs
United We Care
ayushi@unitedwecare.com

Saloni Singla
DataLabs
United We Care
saloni@unitedwecare.com

September 10, 2024

Abstract

As Large Language Models become more ubiquitous across domains, it becomes important to examine their inherent limitations critically. This work argues that hallucinations in language models are not just occasional errors but an inevitable feature of these systems. We demonstrate that hallucinations stem from the fundamental mathematical and logical structure of LLMs. It is, therefore, impossible to eliminate them through architectural improvements, dataset enhancements, or fact-checking mechanisms. Our analysis draws on computational theory and Gödel's First Incompleteness Theorem, which references the undecidability of problems like the Halting, Emptiness, and Acceptance Problems. We demonstrate that every stage of the LLM process—from training data compilation to fact retrieval, intent classification, and text generation—will have a non-zero probability of producing hallucinations. This work introduces the concept of "Structural Hallucinations" as an intrinsic nature of these systems. By establishing the mathematical certainty of hallucinations, we challenge the prevailing notion that they can be fully mitigated.

LLM 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

LLM Use Cases in Data Science

- Data Preprocessing and Cleaning: data cleaning, transformation, parsing and normalization
- Data Exploration and Analysis: Feature engineering and data summarization
- Code Generation and Optimization: Write new code or refactor/optimize existing code.
- Natural Language to SQL: Convert plain language queries into SQL code and automate complex database interactions without deep SQL knowledge.
- Text Data Processing: Sentiment analysis, information extraction, topic modeling, and named entity recognition on unstructured text data.
- Knowledge Extraction from Scientific Papers: Summarize academic papers
- Data Augmentation and Synthesis: Generate synthetic data for model training, and create realistic scenarios for testing model robustness.

LLM Use Cases in Data Science

- Data Preprocessing and Cleaning: data cleaning, transformation, parsing and normalization
- Data Exploration and Analysis: Feature engineering and data summarization
- Code Generation and Optimization: Write new code or refactor/optimize existing code.
- Natural Language to SQL: Convert plain language queries into SQL code and automate complex database interactions without deep SQL knowledge.
- Text Data Processing: Sentiment analysis, information extraction, topic modeling, and named entity recognition on unstructured text data.
- Knowledge Extraction from Scientific Papers: Summarize academic papers
- Data Augmentation and Synthesis: Generate synthetic data for model training, and create realistic scenarios for testing model robustness.

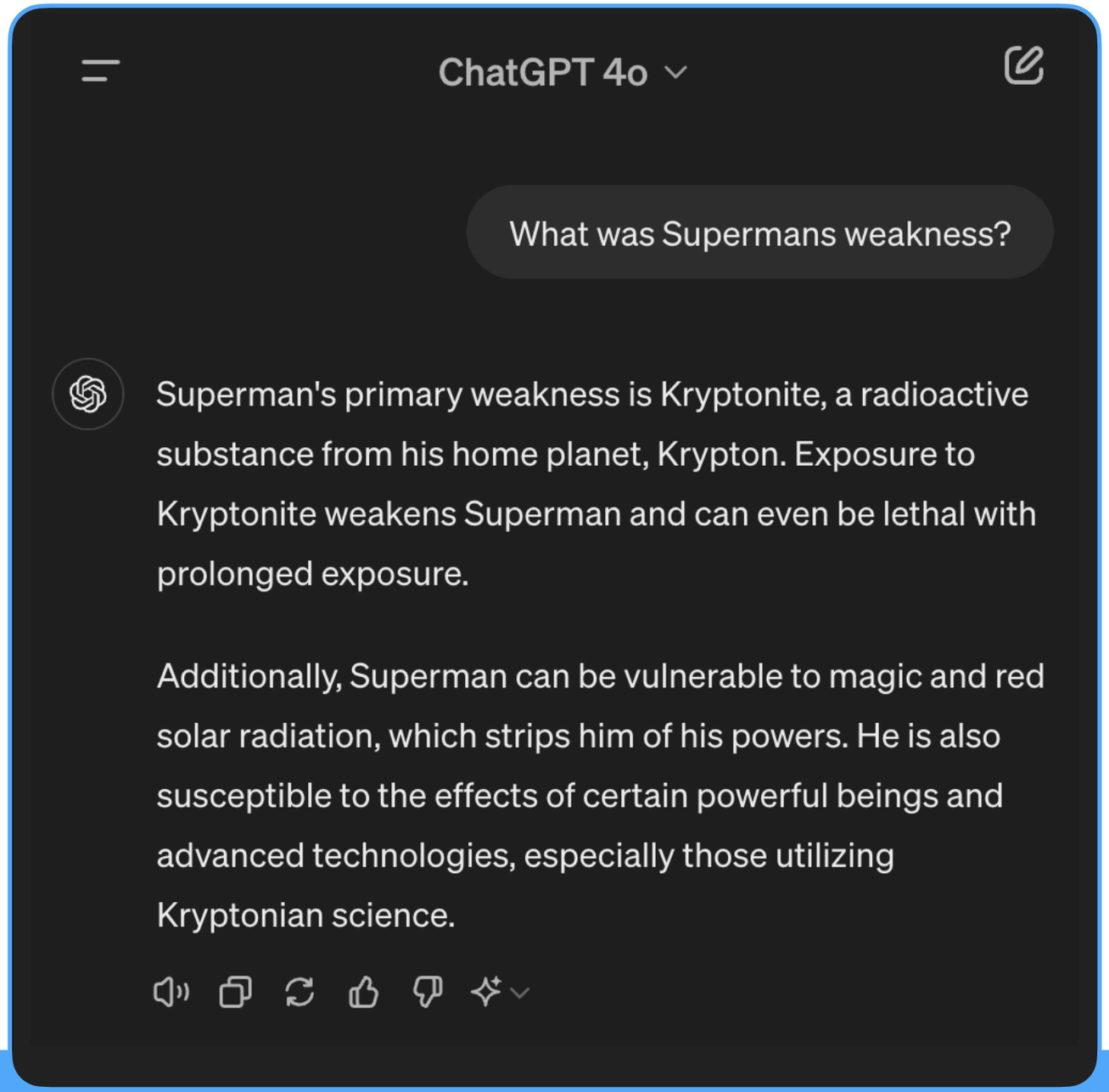
OpenAI

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

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


```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

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

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

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

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	

Hugging Face

- Founded in 2016 by French entrepreneurs in New York City
- Originally a chatbot aimed at teenagers, eventually pivoted to being a Machine Learning platform that allows users to share code and datasets for NLP based models, specially LLMs
- Creator of the transformers package and the BLOOM LLM

Basic Usage

Hugging Face

- The transformers package implements the pipeline() method
- Pipelines are objects that abstract away most of the complexity of interacting with each individual model behind a simple and consistent API
- The basic API call is pipeline() that takes two main arguments and returns a Pipeline object:
  - task - A string identifying the task we are trying to perform. Common examples:
    - 'fill-mask' - Identify possible values for the masked word
    - 'ner' - Named Entity Recognition
    - 'question-answering' - Answer questions about the input
    - 'translation_X_to_Y' - Translate from language X to language Y
    - 'text-generation' - Generate text based on a prompt
  - model [Optional] - the name of the model to use. Each pipeline has a reasonable default value
  - Each task can take further relevant arguments

Basic Usage

Hugging Face

- The Pipeline object is Callable as a function.

```
1 unmasker = pipeline('fill-mask', model='bert-base-uncased')
2
3 unmasker("Artificial Intelligence [MASK] take over the world.")
```

- Any specific arguments passed in a call to the Pipeline object overrides the defaults defined when pipeline() was called, but only for that specific call.
- The exact format of the return value is task dependent but it is typically formed as a JSON object

🦜🔗 LangChain

- Launched in 2022 as an open source project by Harrison Chase
- Incorporated as a company in Jan 2023
- LangChain is a Swiss army knife of wrappers for a huge range of LLMs, database and storage solutions, data formats, etc.
- LangChain revolves around the concept of a pipeline (similar in spirit to the HuggingFace ones) that can be implemented using a wide range of modules to manage and process data from initial input all the way to final output

🦜🔗 LangChain

- LangChain is an open-source Python framework designed to simplify the development of applications using large language models (LLMs).
- Main features:
  - Model Abstraction: LangChain manages inputs and outputs seamlessly by abstracting away model details
  - Modular Structure: Allows developers to combine different prompts and even multiple LLMs within a single application.
  - Chains: Multiple components can be chained together to create complex applications
  - Data Integration: Allows LLMs to transform, store, and retrieve data from databases and online sources.
  - Memory Management: Provides utilities for adding memory to LLM systems, retaining conversation history or summaries for context.

🦜🔗 LangChain

- LangChain can be used to build a wide range of LLM-powered applications:
  - Chatbots
  - Coding assistants and code security analysis
  - Recommendation systems
  - Document analysis and summarization
  - Question-answering systems
  - Data augmentation
  - Text classification and summarization
  - Sentiment analysis
  - Machine translation

🦜🔗 LangChain

- A Chain is a sequence of calls to components, including other chains.
- Conceptually similar to sklearn Pipelines
- Component Examples:
    - LLMMath - For math related queries
    - SQLDatabaseChain - To query databases
    - OpenAIModerationChain - Check content against moderation rules
    - ConstitutionalChain - Legal applications
    - LLMCheckerChain - Prevent hallucinations

Generative AI for Data Science

https://github.com/DataForScience/LLM4DS

2. Prompt Engineering for Data Science

Output Formatting

- We can obtain structured output from LLMs by guiding the model to produce data in formats like JSON, XML, or CSV.

- Different Approaches:

    - Explicit Instructions: Clearly specify the desired format in your prompt.

    - Use Templates: Include a sample of the expected output structure

- Post-processing:

    - Parsing Libraries: Use JSON, XML, etc parsers to validate the output.

    - Error Correction Scripts: Automatically detect and correct common formatting errors.

Prompting

https://lilianweng.github.io/posts/2023-03-15-prompt-engineering/

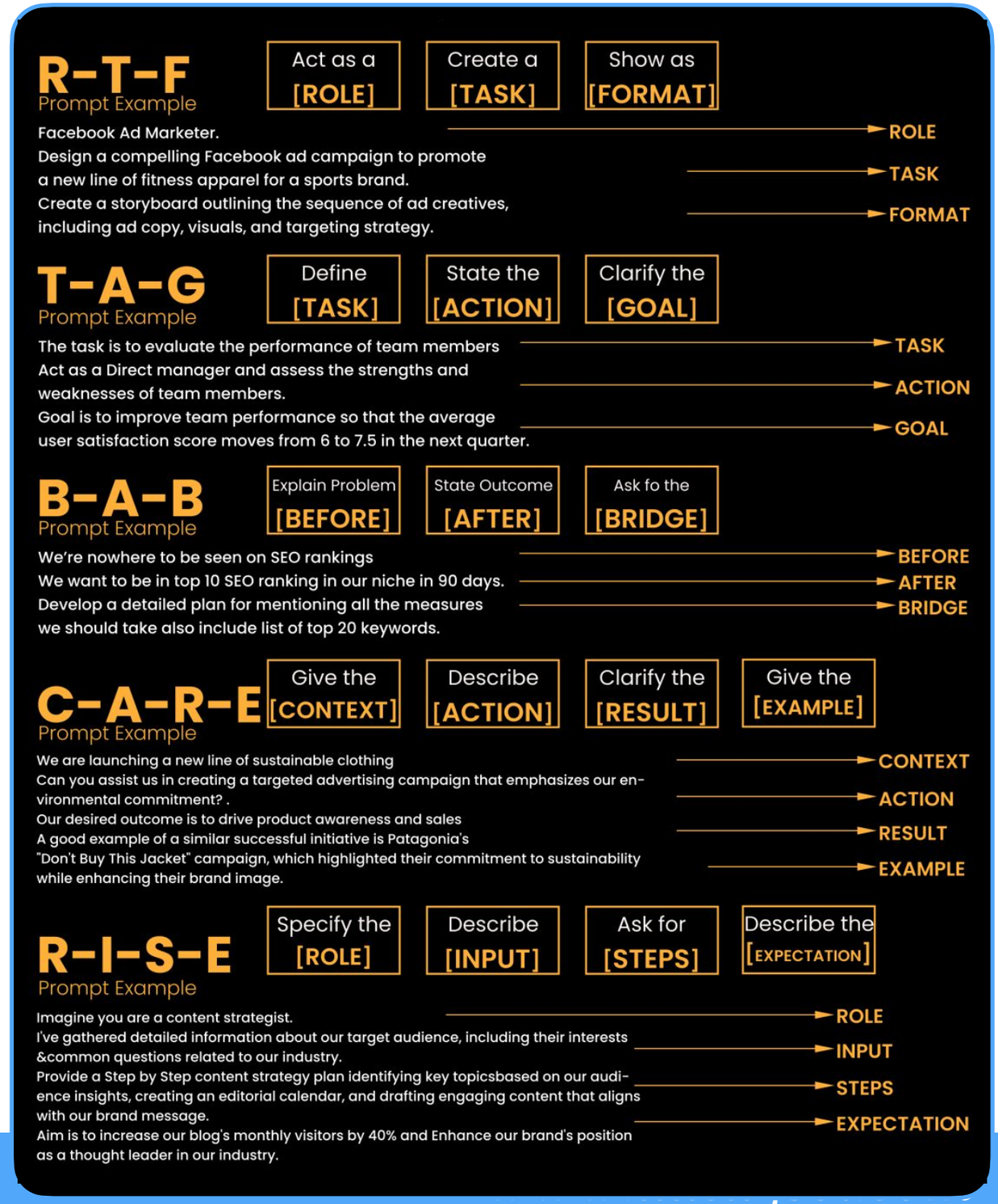
- Prompting is how we interact with LLMs
- Choosing the correct type of prompt can have a major effect on the results we obtain
- Common prompting approaches:
    - Zero-Shot - Explain the task and ask for the result
    - Few-Shot - Provide a few examples of both inputs and outputs that the model can generalize from
    - Chain of Thought - Ask the model to explain how it reached the result

Prompting Tips

- Start simple and integrate by gradually adding the details necessary to improve results
- Break down complex tasks into smaller, more manageable, subtasks
- Use action commands ("Write", "Classify", "Summarize", "Translate", "Order", etc.) to indicate what the goal is
- More descriptive and detailed the prompts produce better results.

Structured Prompts

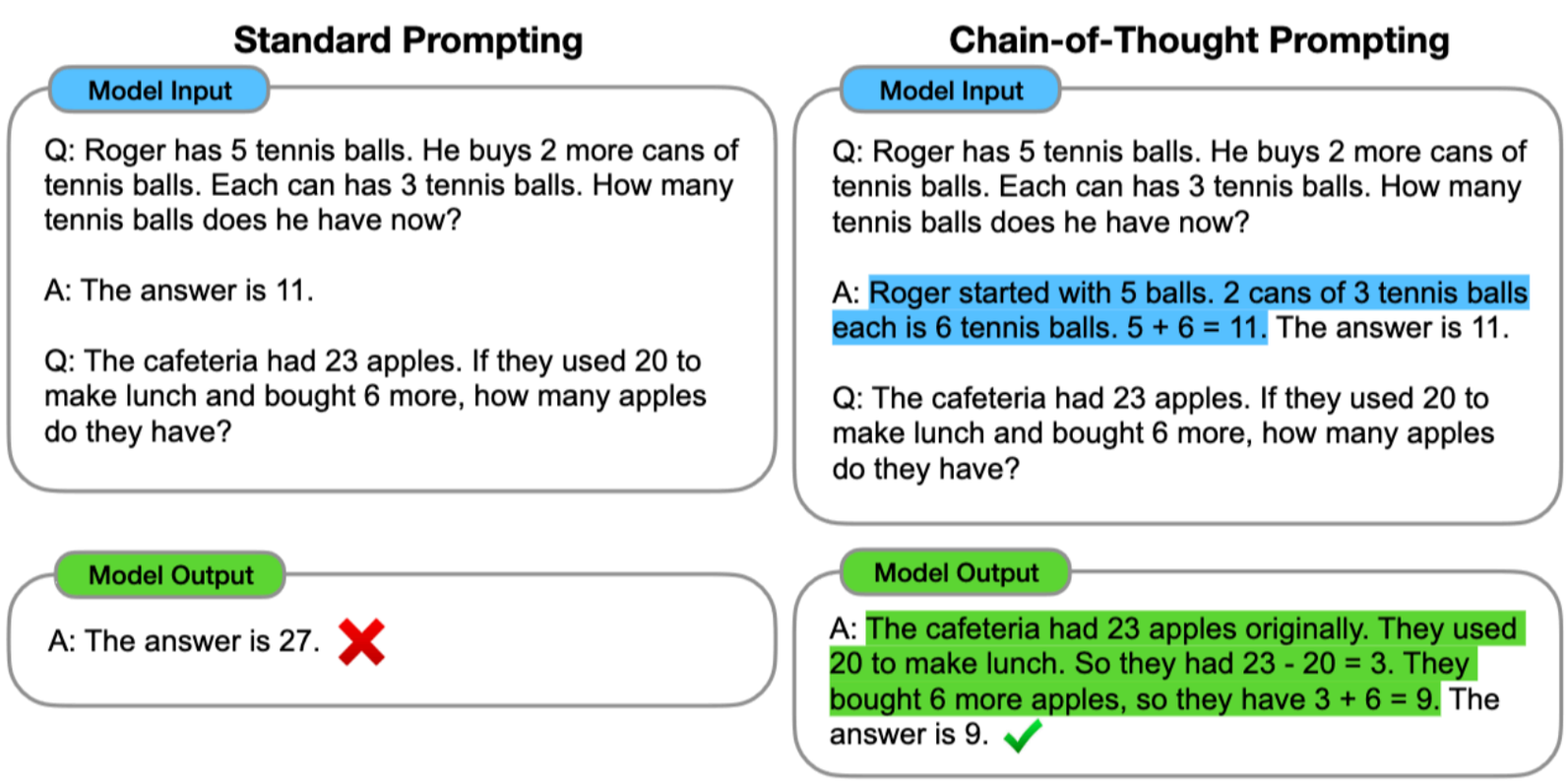
- Structured Prompts can significantly improve the quality of the output. They
  - Help the model concentrate on relevant information, improving the quality of responses.
  - Ensures the model understands the expected perspective and objective.
  - Reducing ambiguity in the model's response.
  - Reduce the likelihood of formatting errors



Chain-of-Thought prompts

arXiv: 2201.11903 (2022)

- Chain of Thought (CoT) prompts encourages LLMs to generate intermediate steps before providing the final solution to a problem.
- CoT breaks down complex problems into manageable, intermediate steps and mimic an human thought process when working through multi-step problems.
- Relatively small change to the prompt can have a profound effect



Chain-of-Thought prompts

arXiv: 2201.11903 (2022)

- Chain of Thought (CoT) prompts encourages LLMs to generate intermediate steps before providing the final solution to a problem.
- CoT breaks down complex problems into manageable, intermediate steps and mimic an human thought process when working through multi-step problems.
- Relatively small change to the prompt can have a profound effect

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Add an example to break down the problem

Chain-of-Thought prompts

arXiv: 2201.11903 (2022)

- Chain of Thought (CoT) prompts encourages LLMs to generate intermediate steps before providing the final solution to a problem.
- CoT breaks down complex problems into manageable, intermediate steps and mimic an human thought process when working through multi-step problems.
- Relatively small change to the prompt can have a profound effect

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Add an example to break down the problem

LLM follows similar detailed approach to reach a better solution

Chain-of-Thought prompts

arXiv: 2201.11903 (2022)

Math Word Problems (free response)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Math Word Problems (multiple choice)

Q: How many keystrokes are needed to type the numbers from 1 to 500?
Answer Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788

A: There are 9 one-digit numbers from 1 to 9. There are 90 two-digit numbers from 10 to 99. There are 401 three-digit numbers from 100 to 500. 9 + 90(2) + 401(3) = 1392. The answer is (b).

CSQA (commonsense)

Q: Sammy wanted to go to where the people were. Where might he go?
Options: (a) race track (b) populated areas (c) desert (d) apartment (e) roadblock

A: The answer must be a place with a lot of people. Race tracks, desert, apartments, and roadblocks don't have a lot of people, but populated areas do. So the answer is (b).

StrategyQA

Q: Yes or no: Would a pear sink in water?

A: The density of a pear is about 0.6 g/cm^3, which is less than water. Thus, a pear would float. So the answer is no.

Date Understanding

Q: The concert was scheduled to be on 06/01/1943, but was delayed by one day to today. What is the date 10 days ago in MM/DD/YYYY?

A: One day after 06/01/1943 is 06/02/1943, so today is 06/02/1943. 10 days before today is 05/23/1943. So the answer is 05/23/1943.

Sports Understanding

Q: Is the following sentence plausible? "Joao Moutinho caught the screen pass in the NFC championship."

A: Joao Moutinho is a soccer player. The NFC championship is part of American football, not soccer. So the answer is no.

SayCan (Instructing a robot)

Human: How would you bring me something that isn't a fruit?

Explanation: the user wants something to eat that isn't a fruit. An energy bar is not a fruit, so I will bring the user an energy bar.
Plan: 1. find(energy bar) 2. pick(energy bar) 3. find(user) 4. put(energy bar) 5. done().

Last Letter Concatenation

Q: Take the last letters of the words in "Lady Gaga" and concatenate them.

A: The last letter of "Lady" is "y". The last letter of "Gaga" is "a". Concatenating them is "ya". So the answer is ya.

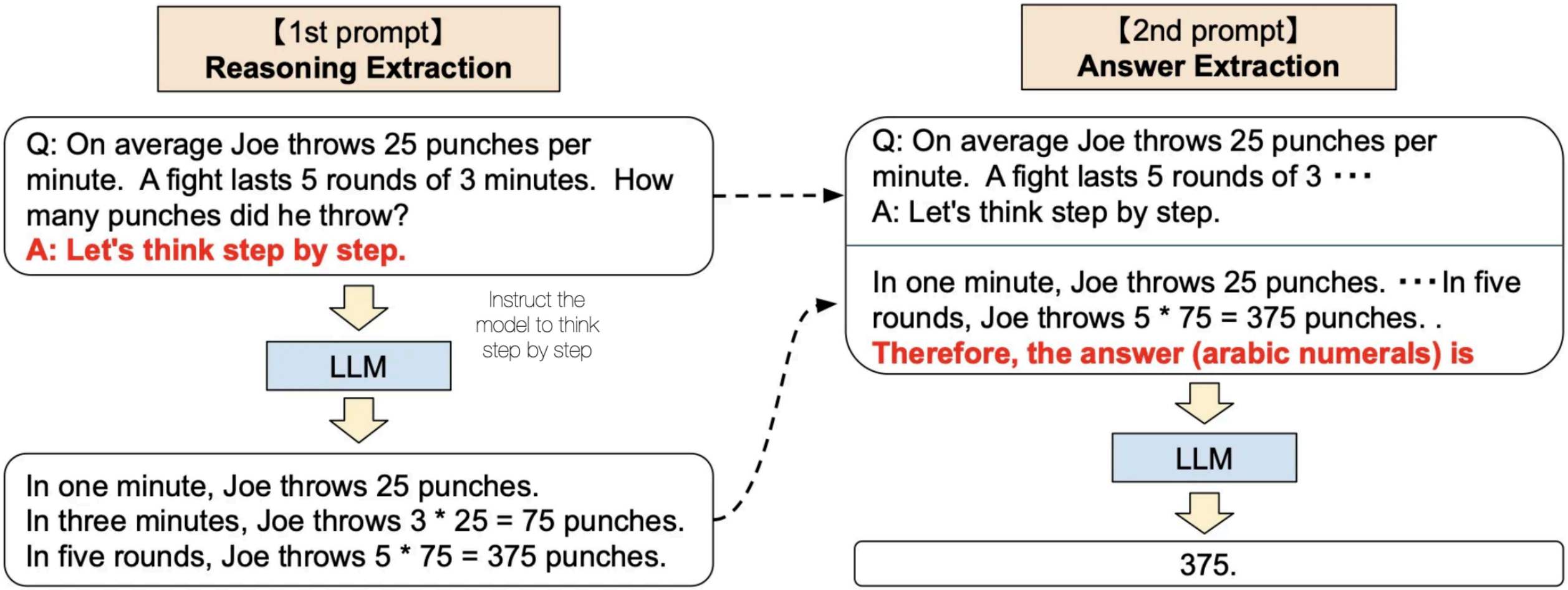
Coin Flip (state tracking)

Q: A coin is heads up. Maybelle flips the coin. Shalonda does not flip the coin. Is the coin still heads up?

A: The coin was flipped by Maybelle. So the coin was flipped 1 time, which is an odd number. The coin started heads up, so after an odd number of flips, it will be tails up. So the answer is no.

Chain-of-Thought - Zero Shot

arXiv: 2201.11903 (2022)



Prompt Engineering for DS

https://github.com/DataForScience/LLM4DS

3. NLP with HuggingFace

Advantages of LLMs for NLP Tasks

- LLMs excel in various traditional NLP tasks by leveraging their ability to understand context, nuance, and syntax in text
- Advantages:
  - Versatility - The same model can handle a large number of tasks
  - Minimal or no task specific training - Using One-Shot or Few-Shot prompting techniques
  - Ease of Integration - Tools like the OpenAI, Hugging Face, or LangChain facilitate the integration of LLMs into new or pre-existing data processing pipelines

Disadvantages of LLMs for NLP Tasks

- LLMs excel in various traditional NLP tasks by leveraging their ability to understand context, nuance, and syntax in text

- Caveats:

    - Computationally Intensive - Training and Inference require significant computational resources

    - Bias and Fairness - LLMs reflect any biases that were present in their training data

    - Hallucinations - LLMs can generate plausible-sounding outputs that are completely incorrect.

Named Entity Recognition

- Named Entity Recognition (NER) is a traditional NLP technique used to classify key entities within text into predefined categories. Entities cam be names of people, organizations, locations, dates, numerical expressions, and other domain-specific terms.

- Applications:

    - Information Extraction: Extract key facts from documents

    - Search and Information Retrieval: Help search engines provide more precise search results by indexing and recognizing named entities

    - Customer Support Automation

    - Document Summarization.

    - Data Enrichment: Connect entities to external data sources

- LLMs are a significant improvement over previous approaches due to their ability to leverage context to best identify entities

Part-of-Speech Tagging

- Part of Speech (POS) Tagging labels words in a snippet of text with their corresponding part of speech (e.g., Noun, Verb, Adjective) to better understand the syntactic structure of sentences, which is crucial for downstream tasks like parsing, named entity recognition, and text generation.

- Applications:

    - Syntax Analysis: Understand the grammatical structure of sentences

    - Disambiguation: Remove ambiguities in words whose meaning is context dependent

    - Feature Extraction: Generate linguistic features for others tasks like NER and sentiment analysis.

- LLMs use attention mechanisms to capture complex dependencies and contextual information

Summarization

- Summarization is the process of generating a shorter version of a longer piece of text while preserving its main ideas and essential information.

- Two main approaches to Summarization:

    - Extractive - Selects key sentences, phrases, or segments directly from the original text to identify and extract the most informative parts of the text.

    - Abstractive - Generates a summary by generating novel sentences that capture the essence of the document.

- LLMs generally produce abstractive summarization but can also be fine-tuned to produce extractive summarization.

Question Answering

- Question Answering (QA) techniques aim to provide answers to user queries based on a given context or knowledge base.
- Many points in common to Summarization
- Two main approaches of QA:
    - Extractive - Selects key sentences, phrases, or segments directly from the original text that contain the answer.
    - Abstractive - Generates an answer by generating novel sentences that answer the query based on the context provided
- Two main classes of QA:
    - Closed-Domain - Limited to well defined domains
    - Open-Domain - No predefined constraints

Sentiment Analysis

- Question Answering (QA) techniques aim to provide answers to user queries based on a given context or knowledge base.
- Many points in common to Summarization
- Two main approaches of QA:
    - Extractive - Selects key sentences, phrases, or segments directly from the original text that contain the answer.
    - Abstractive - Generates an answer by generating novel sentences that answer the query based on the context provided
- Two main classes of QA:
    - Closed-Domain - Limited to well defined domains
    - Open-Domain - No predefined constraints

NLP with HuggingFace

https://github.com/DataForScience/LLM4DS

4. Text to Speech with OpenAI

The Whisper model

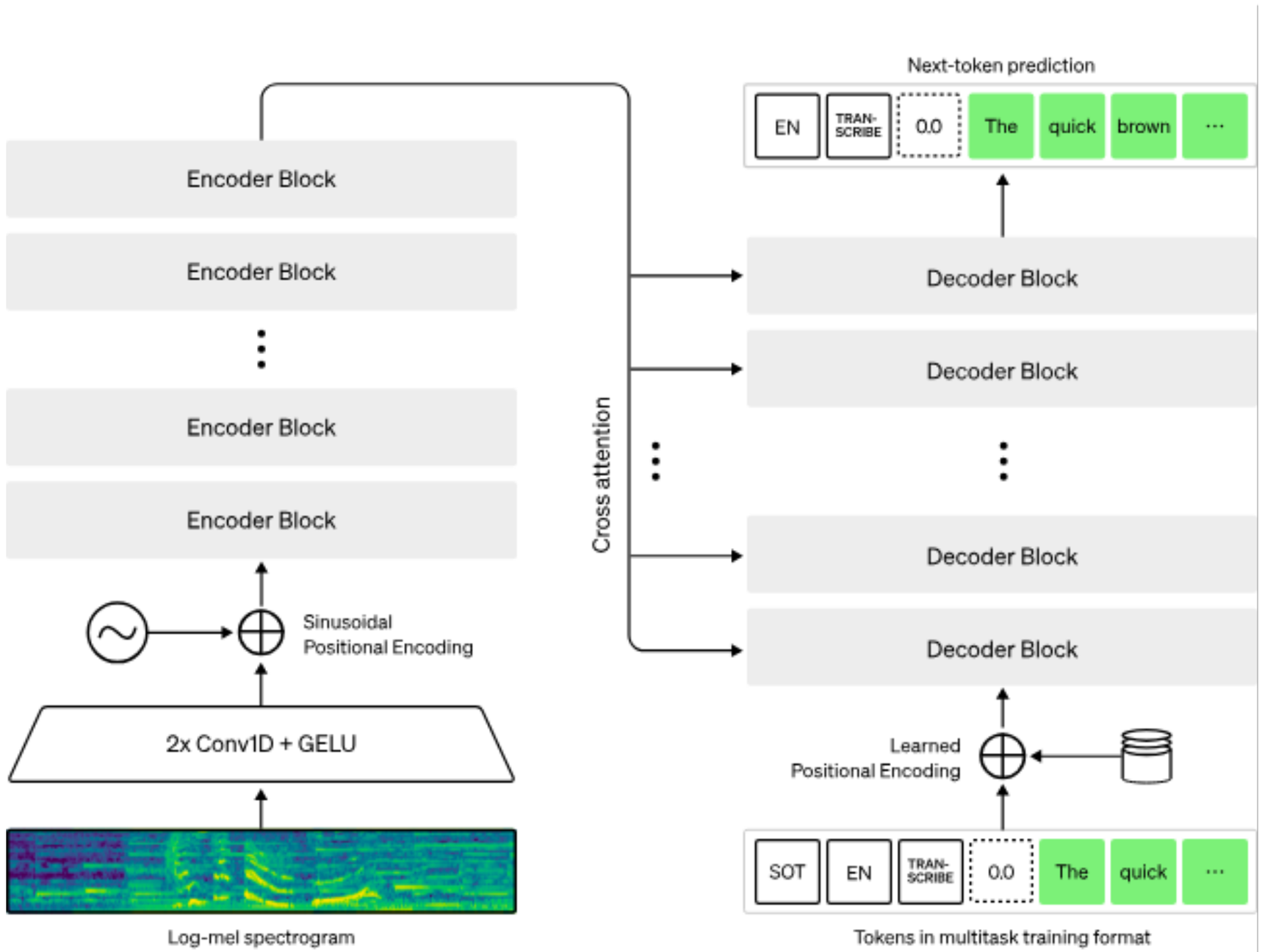
https://openai.com/index/whisper/

- Whisper is an Automatic Speech Recognition (ASR) system trained on multilingual and multitask supervised data collected from the web.
- Originally introduced in Sep. 2022, the most recent version is from Nov. 2023
- Features
    - Text to Speech
    - Audio Transcription
    - Automatic Translation

Whisper Architecture

https://openai.com/index/whisper/

- Basically a modified transformer!



Generating audio from text

- The basic API call is audio.speech.create()
- It takes several required arguments:
  - model - The model to use, tts-1
  - input - The text to use,
  - voice - The voice the audio is going to use to “read” the text
  - response_format - What format to return the transcription, “mp3”, “opus”, “aac”, “flac”, “wav”, “pcm”

Audio transcription

- The basic API call is audio.transcriptions.create()
- It takes several required arguments:
    - model - The model to use, whisper-1
    - file - The path to the audio file to use,
    - response_format - What format to return the transcription, “json”, “text”, “srt"
    - language - The language to produce the text in
- ”name” [optional] - An optional name field to be used to identify the participants in the conversation

Audio transcription

```python
with open("data/gettysburg10.wav", "rb") as audio_file:
    transcript = client.audio.transcriptions.create(
        file = audio_file,
        model = "whisper-1",
        response_format="text",
        language="es"
    )
```

- The response is a string of text containing the transcription

Text to Speech with OpenAI

https://github.com/DataForScience/LLM4DS

5. Pandas AI

PandasAI

https://pandas-ai.com/

- Python library that makes it easy to use natural language to ask questions of your data
- Features:
  - Natural Language Querying
  - Data Visualization
  - Data Cleaning: Fix missing values.
  - Feature Extraction
  - Data Connectors: Supports CSV, XLSX, PostgreSQL, MySQL, BigQuery, Databricks, Snowflake, etc.
  - LLM Support: Use your favorite LLM in the background

PandasAI

https://pandas-ai.com/

- Supports a large number of LLMs
  - BambooLLM - PandasAI own LLM and the default unless another LLM is explicitly selected. Obtain free API key from https://pandabi.ai
  - OpenAI
  - PaLM
  - VertexAI
  - HuggingFace
  - LangChain
  - etc…
- API keys can be provided explicitly or made available in the OS as an environment variable

Data Structures

PandasAI

- Two main data structures:
  - SmartDataFrame - The main data structure. Represents an individual database table or pandas DataFrame

```python
from pandasai import SmartDataframe
from pandasai.llm import OpenAI

llm = OpenAI()
pandas_ai = SmartDataframe("data.csv", config={"llm": llm})
```

  - SmartDataLake - Data structure to support queries across multiple DataFrames

```python
from pandasai import SmartDataframe, SmartDatalake
from pandasai.llm import OpenAI

employees_df = SmartDataframe(employees_data)
salaries_df = SmartDataframe(salaries_data)

llm = OpenAI()
lake = SmartDatalake([employees_df, salaries_df], config={"llm": llm})
lake.chat("Who gets paid the most?")
```

Data Structures

PandasAI

- Two main data structures:
  - SmartDataFrame - The main data structure. Represents an individual database table or pandas DataFrame

```python
from pandasai import SmartDataframe
from pandasai.llm import OpenAI

llm = OpenAI()
pandas_ai = SmartDataframe("data.csv", config={"llm": llm})
```

  - SmartDataLake - Data structure to support queries across multiple DataFrames

```python
from pandasai import SmartDataframe, SmartDatalake
from pandasai.llm import OpenAI

employees_df = SmartDataframe(employees_data)
salaries_df = SmartDataframe(salaries_data)

llm = OpenAI()
lake = SmartDatalake([employees_df, salaries_df], config={"llm": llm})
lake.chat("Who gets paid the most?")
```

SmartDataLake takes a list of DataFrames

Connectors

PandasAI

- Connectors facilitate interaction with external databases:
  - PostgreSQLConnector
  - MySQLConnector
  - SqliteConnector
  - SnowFlakeConnector
  - GoogleBigQueryConnector
  - etc
- Each specific connector instance handles only a single table (like a SmartDataframe)
- The arguments necessary for each connector depended on the underlying database, but they all require the name of the table to connect to.

Agents

PandasAI

- An Agent is the only PandasAI data structure that is able to maintain context throughout the conversation
- An Agent is a drop-in replacement for SmartDatalake
- The clarification_questions() method can be used to request clarification on any aspect of the conversation
- The explain() method provides detailed explanations of the agent "thought" process that led to the output observed

Pandas AI

https://github.com/DataForScience/LLM4DS

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

LangChain for Generative AI Pipelines

Sept 25, 2024 - 10am-2pm (PST)

Generative AI with the OpenAI API for Developers

Oct 9, 2024 - 10am-2pm (PST)

ChatGPT and Competing LLMs

Nov 13, 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.

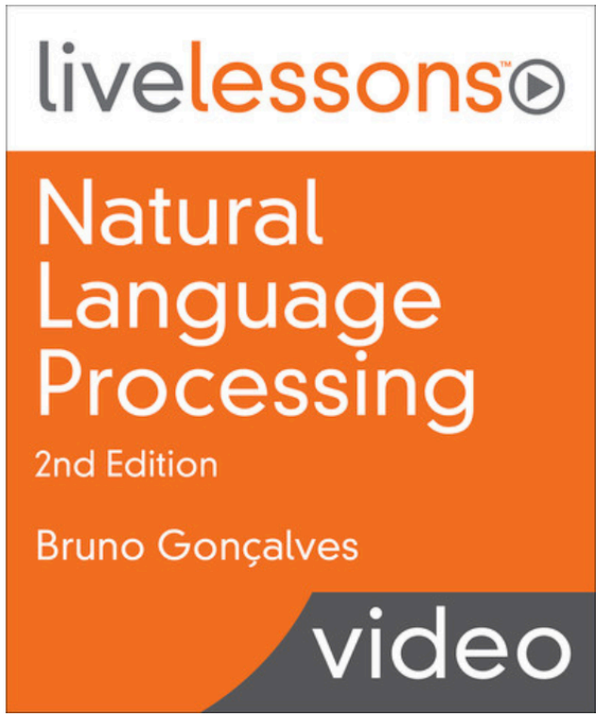
VIDEO

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