Building Text-Based Applications with the ChatGPT API & LangChain

O'REILLY®

How to build LLM apps

Lucas Soares

21-08-2023

Intro

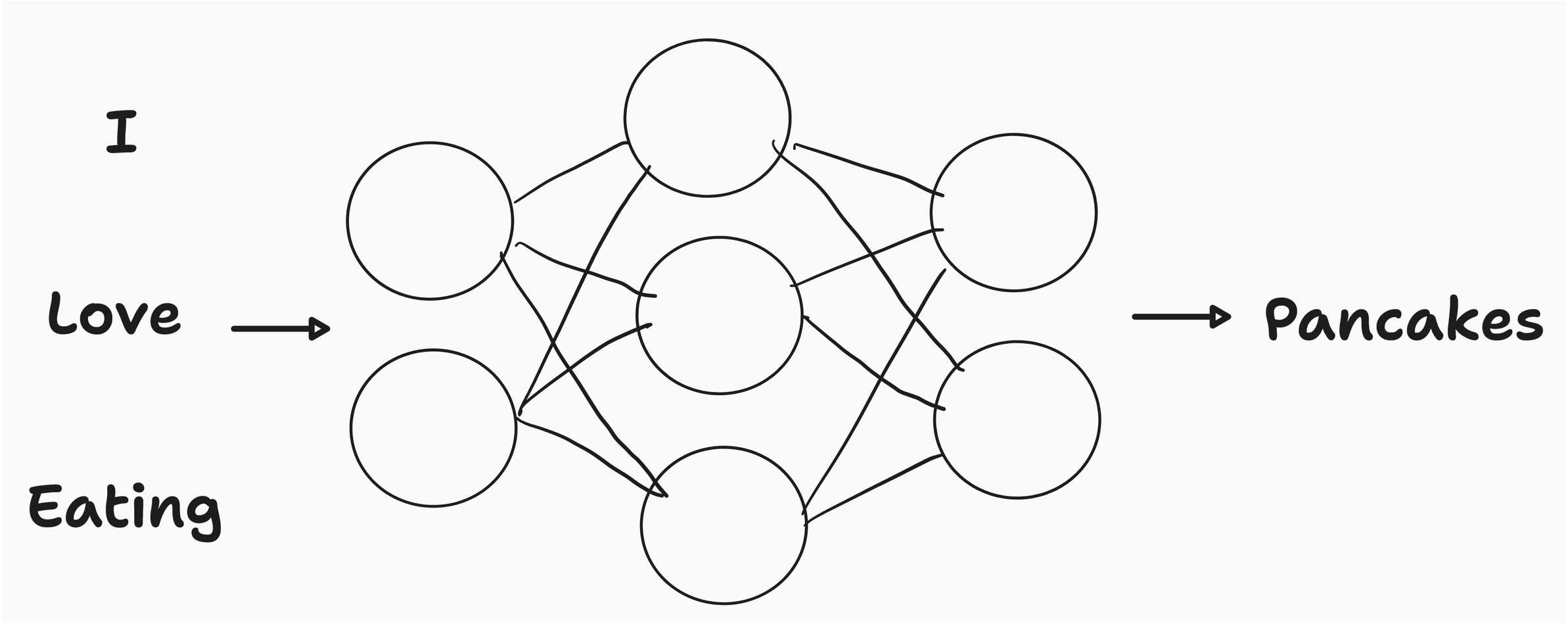
Hi!

- Me!
- Quick Survey to get to know everyone!

Quick Survey to get to know everyone!

Large Language Models

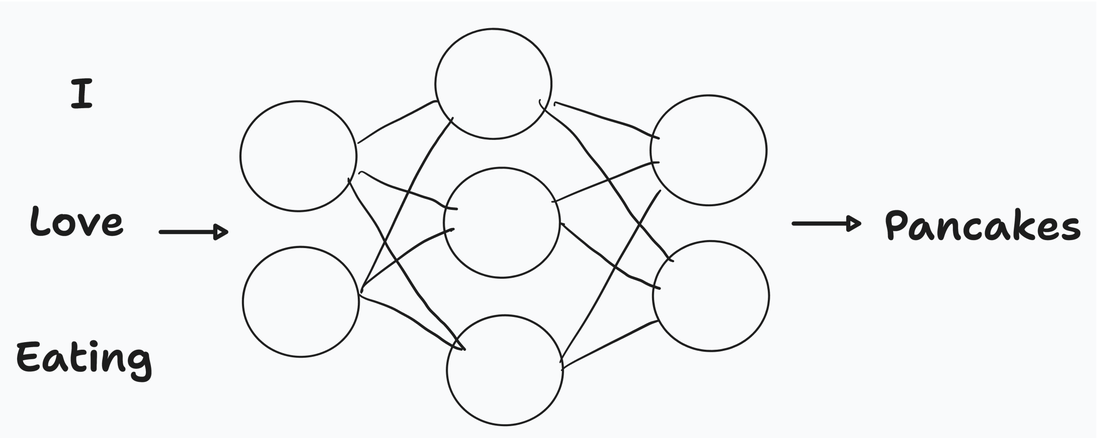
A definition



Large Language Models

As Probability Distributions

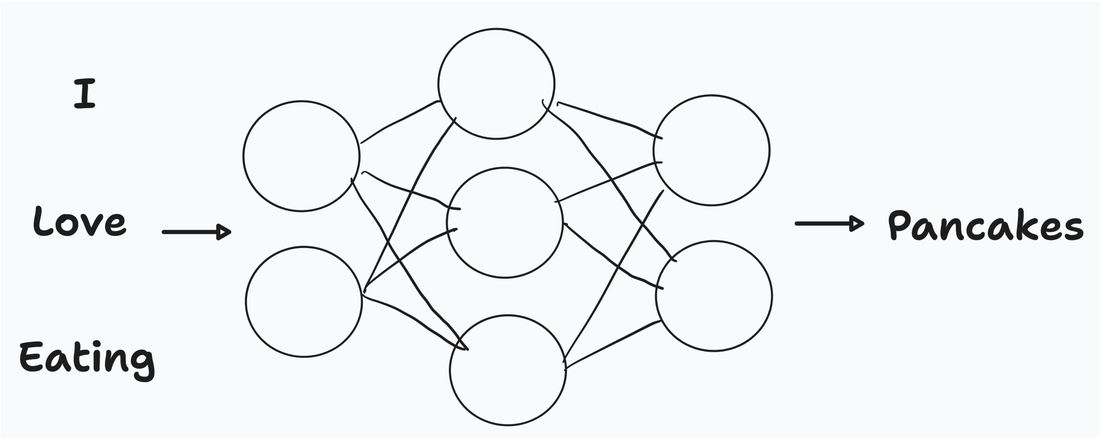
At their core, LLMs can be seen as distributions over words.



Large Language Models

As Probability Distributions

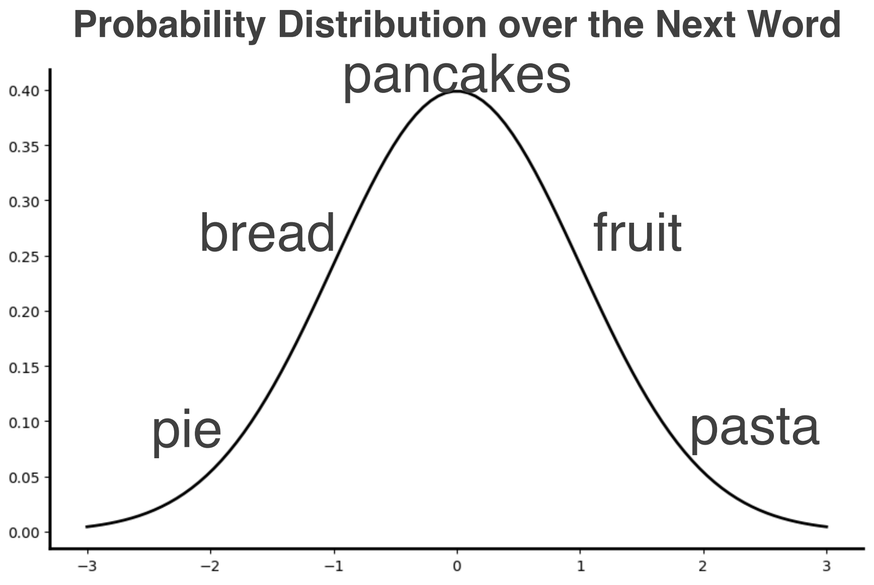
- At their core, LLMs can be seen as distributions over words.
- Use statistical models to capture patterns in text data.



Large Language Models

As Probability Distributions

- At their core, LLMs can be seen as distributions over words.
- Use statistical models to capture patterns in text data.
- They calculate the likelihood of each word occurring given the context.

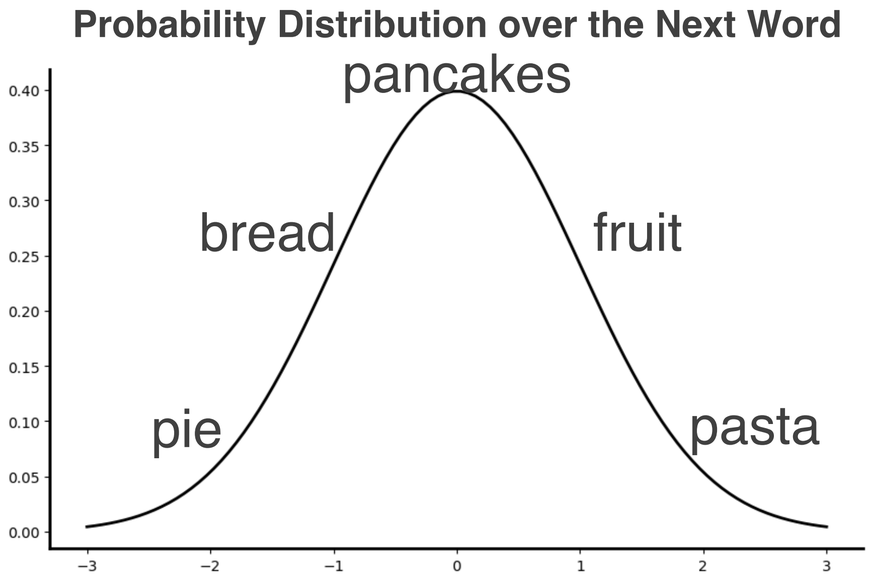


"I love eating…." → ?

Large Language Models

As Probability Distributions

- At their core, LLMs can be seen as distributions over words.
- Use statistical models to capture patterns in text data.
- They calculate the likelihood of each word occurring given the context.



"I love eating…." → ?

Large Language Models

N-gram models

- **Unigram model**: each token/word is independently modeled.

Large Language Models

N-gram models

- **Unigram model**: each token/word is independently modeled.
- Sentence S = *"When the bough breaks, the cradle will fall."*

Unigram Word Probabilities

p = 0.86	p = 0.93	p = 0.22	p = 0.66	p = 0.66	p = 0.37	p = 0.15	p = 0.1
When	the	bough	breaks	the	cradle	will	fall

Large Language Models

N-gram models

```python
import random
sentence = "When the bough breaks the craddle will fall"

tokens = sentence.split(" ")
# First, the token_probs list is created with random values
token_probs = [random.random() for _ in tokens]
# Then, the probabilities dictionary is created
# The dictionary has keys of the form 'word1 word2', where 'word1' and
'word2' are consecutive words in the sentence
# The value for each key is the conditional probability of 'word2' given
'word1'
probabilities = {}
for i in range(len(tokens)):
    if tokens[i] in probabilities.keys():
        probabilities[f"{str(tokens[i])+'-2'}"] = round(token_probs[i],2)
    else:
        probabilities[f"{tokens[i]}"] = round(token_probs[i],2)
probabilities
```

Unigram Word Probabilities

p = 0.86	p = 0.93	p = 0.22	p = 0.66	p = 0.66	p = 0.37	p = 0.15	p = 0.1
When	the	bough	breaks	the	cradle	will	fall

$$P(S) = P(\text{"When"}) \times P(\text{"the"}) \times P(\text{"bough"}) \times P(\text{"breaks"}) \times P(\text{"the"}) \times P(\text{"cradle"}) \times P(\text{"will"}) \times P(\text{"fall"})$$

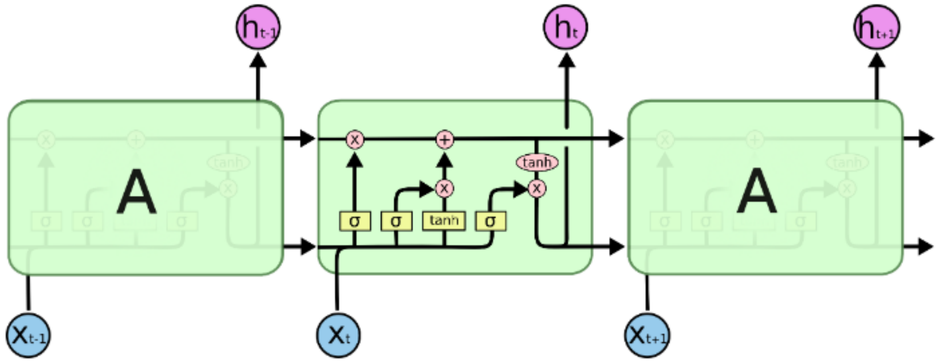
The probability of the sentence is the product of the individual probabilities:

Large Language Models

Sequence to Sequence Models: LSTMs

LSTMs, forgetting mechanism to model longer sentences and maintain context.

(Hochreiter and Jürgen Schmidhuber 1997)

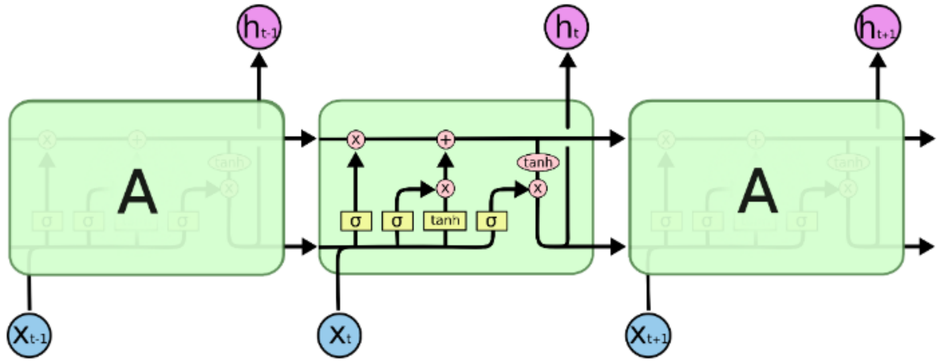


Understanding LSTM Networks by Christopher Olah

Large Language Models

Sequence to Sequence Models: LSTMs

- **LSTMs**, forgetting mechanism to model longer sentences and maintain context.
  (Hochreiter and Jürgen Schmidhuber 1997)
- The probability of the sentence now is influenced by context

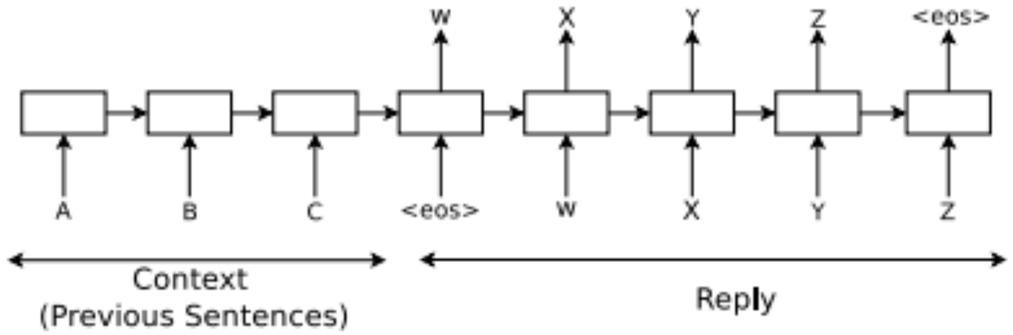


Understanding LSTM Networks by Christopher Olah

Large Language Models

Sequence to Sequence Models: LSTMs

Seq2Seq models require **processing input sequentially**

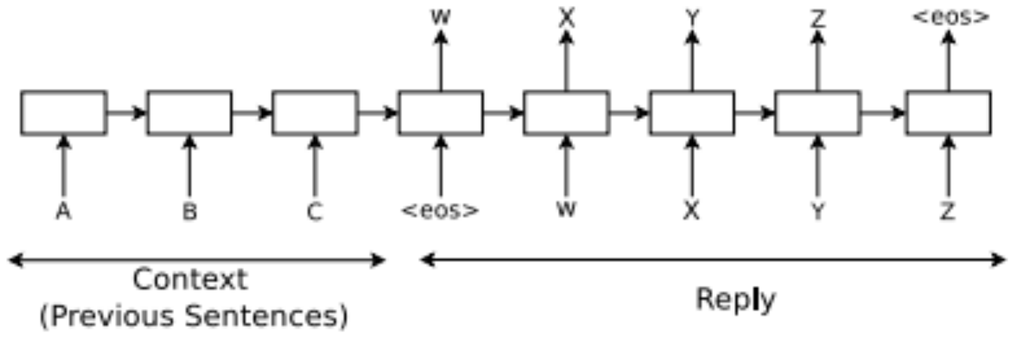


(Vinyals & Le, 2015)

Large Language Models

Sequence to Sequence Models: LSTMs

- **Seq2Seq** models require **processing input sequentially**
- Struggle with really long complex inputs

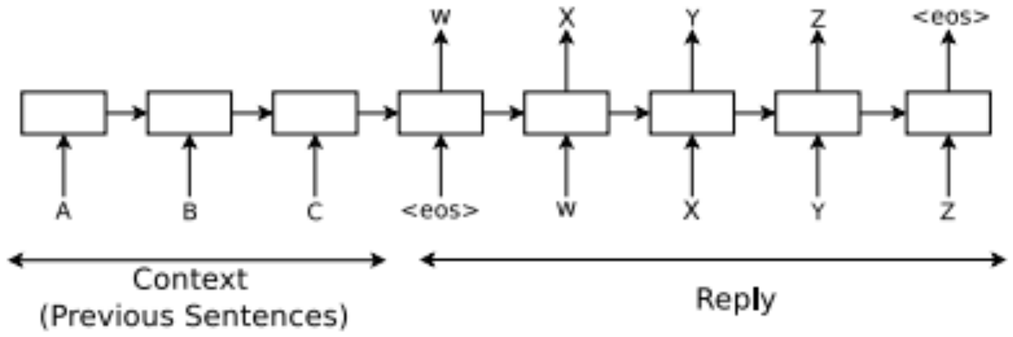


(Vinyals & Le, 2015)

Large Language Models

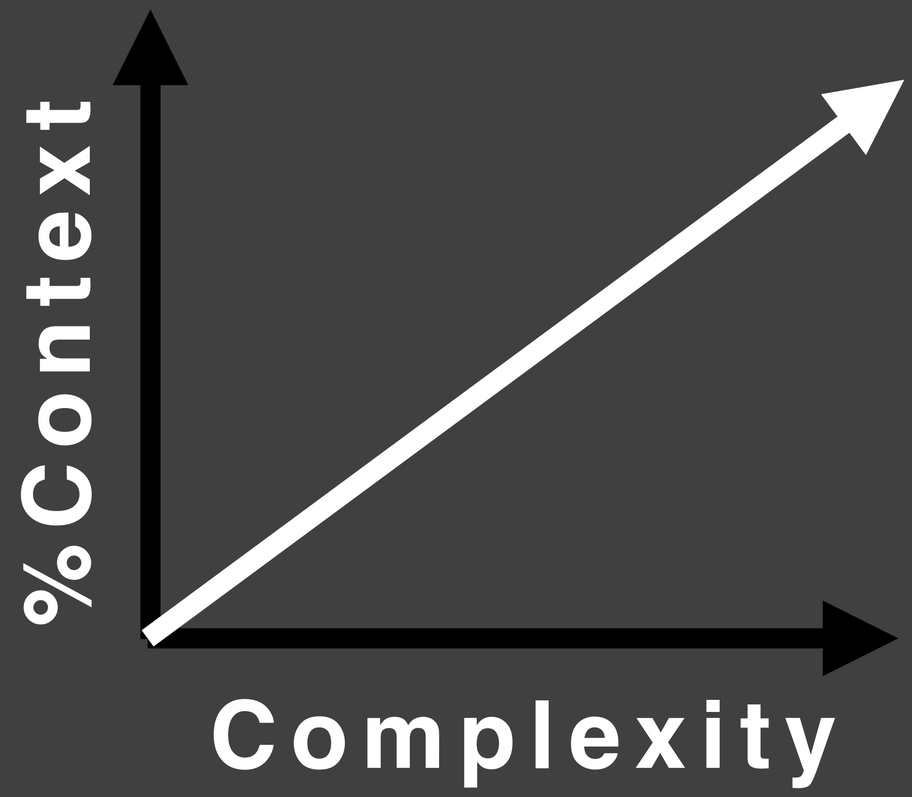
Sequence to Sequence Models: LSTMs

- **Seq2Seq** models require **processing input sequentially**
- Struggle with really long complex inputs



(Vinyals & Le, 2015)

Can't capture really long contexts!


%Context
Complexity

They capture context, understand dependencies, and predict text based on patterns learned during training.

Ok, but how?

LLMs as pattern matchers

LLMs are akin to probabilistic programs.

Ok, but how?

LLMs as pattern matchers

- LLMs are akin to probabilistic programs.
- They predict outcomes based on probabilities learned from training data

Ok, but how?

LLMs as pattern matchers

- LLMs are akin to probabilistic programs.
- They predict outcomes based on probabilities learned from training data

```python
import numpy as np

def llm_model(context, next_word):
    possible_next_words = ["text", "pie", "pizza", "motor", "cake"]
    next_word_probs = [0.2, 0.3, 0.4, 0.001, 0.6]
    return next_word_probs[possible_next_words.index(next_word)]

context  = ["This", "is", "a", "piece", "of"]
possible_next_words = ["text", "pie", "pizza", "motor", "cake"]
probs = []
for w in possible_next_words:
    probs.append(llm_model(context, w))

next_word = possible_next_words[np.argmax(probs)]
next_wor
```


Context/Past
LM
Future

How LLMs work

Transformers Architecture

- Traditional sequential models struggle with context (Vaswani et al 2017)

How LLMs work

Transformers Architecture

- Traditional sequential models struggle with context (Vaswani et al 2017)
- Transformers use attention mechanisms to capture global dependencies, enabling contextual understanding. (Vaswani et al 2017)

How LLMs work

Transformers Architecture

- Traditional sequential models struggle with context (Vaswani et al 2017)
- Transformers use attention mechanisms to capture global dependencies, enabling contextual understanding. (Vaswani et al 2017)
- The attention mechanism allows Transformers to focus on different parts of input simultaneously. (Vaswani et al 2017)

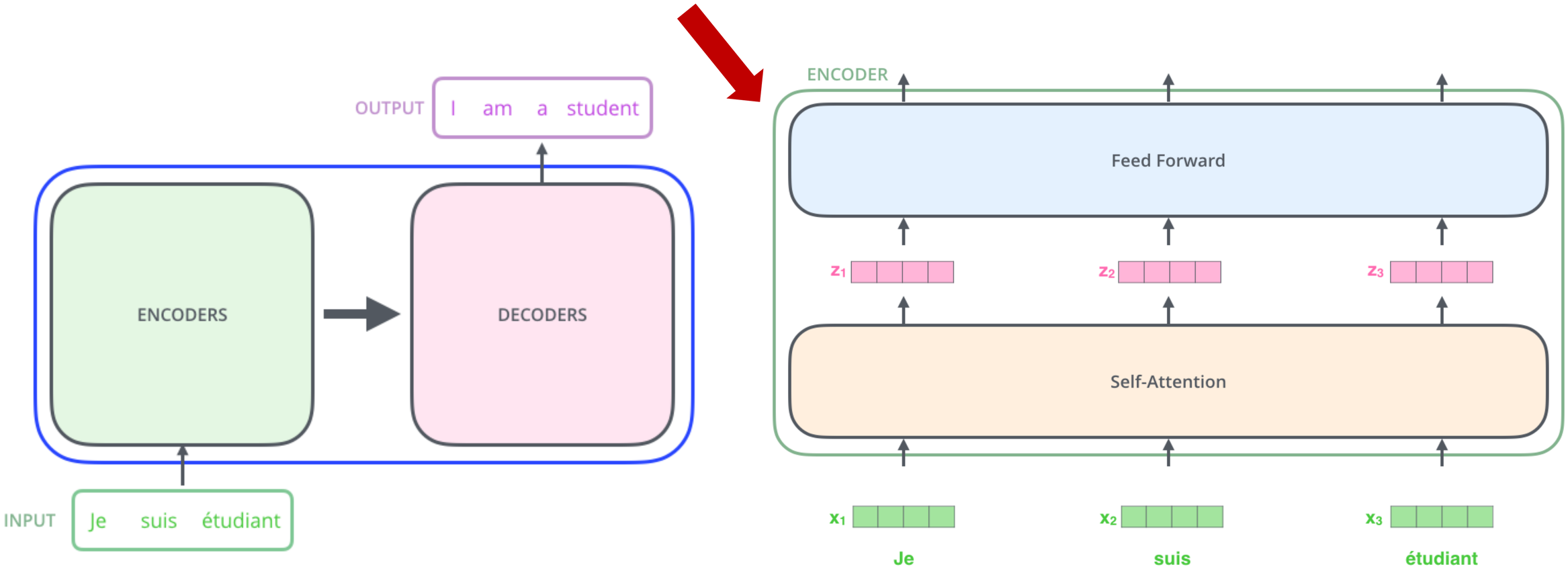
How LLMs work

Transformers Architecture

- Traditional sequential models struggle with context (Vaswani et al 2017)
- Transformers use attention mechanisms to capture global dependencies, enabling contextual understanding. (Vaswani et al 2017)
- The attention mechanism allows Transformers to focus on different parts of input simultaneously. (Vaswani et al 2017)
- Transformers can understand and predict based on long-range dependencies. (Vaswani et al 2017)

How LLMs work

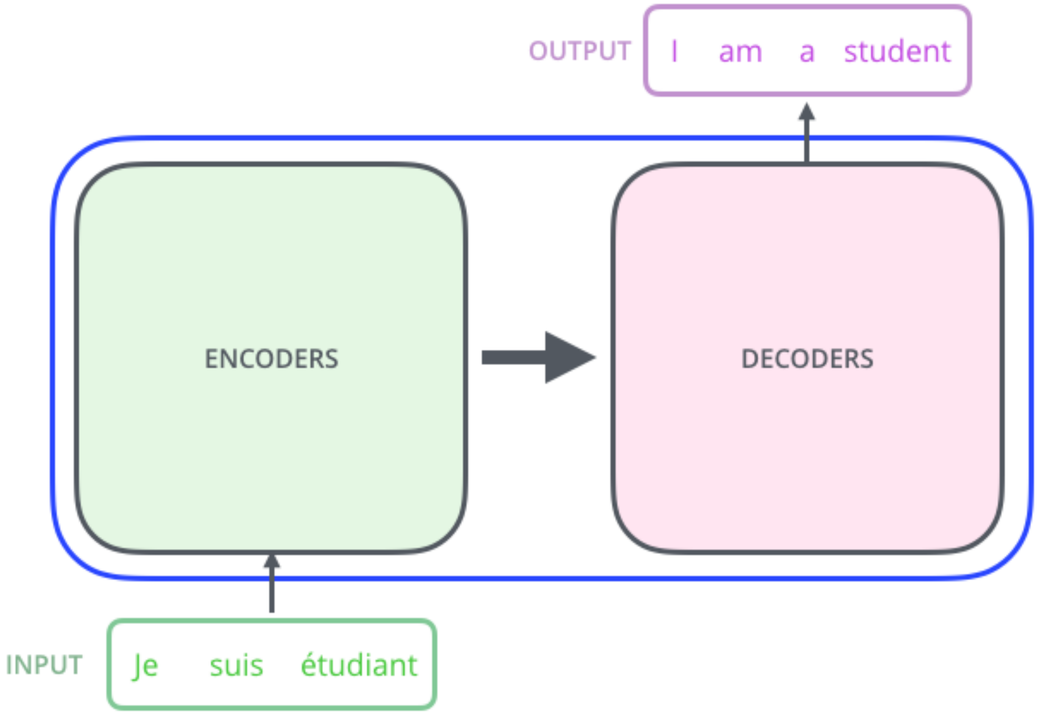
Transformers Architecture

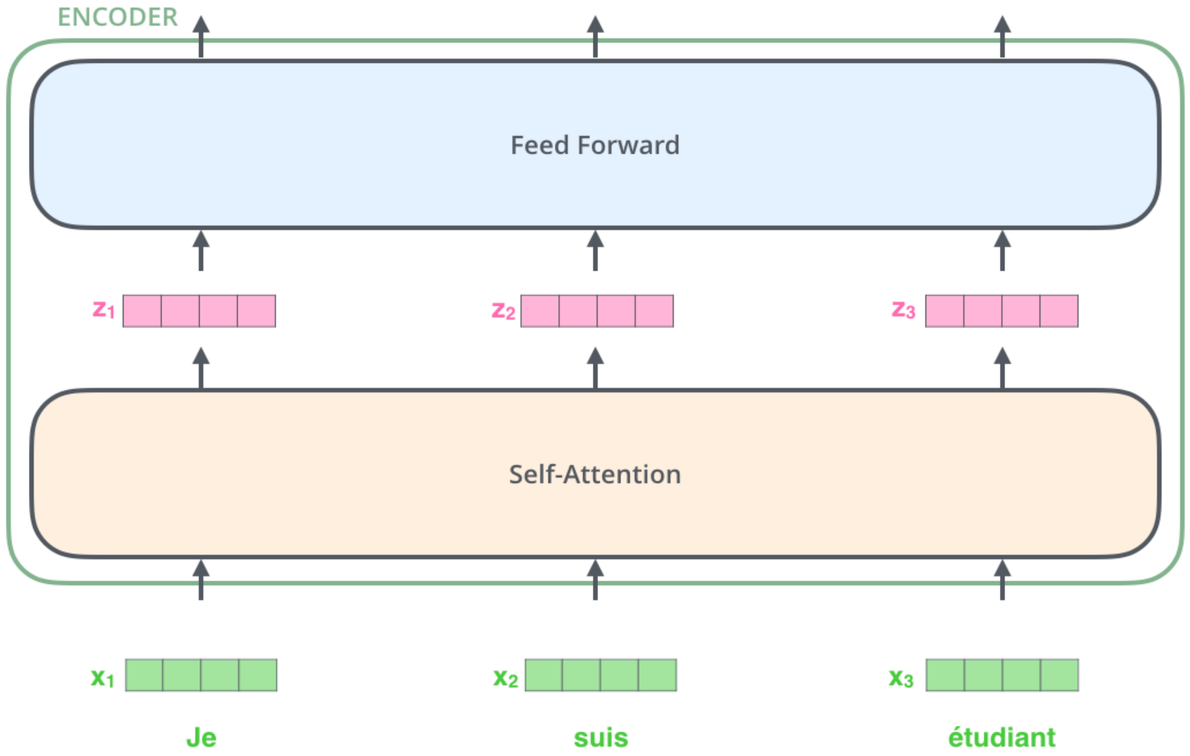


Inputs are processed in parallel!

How LLMs work

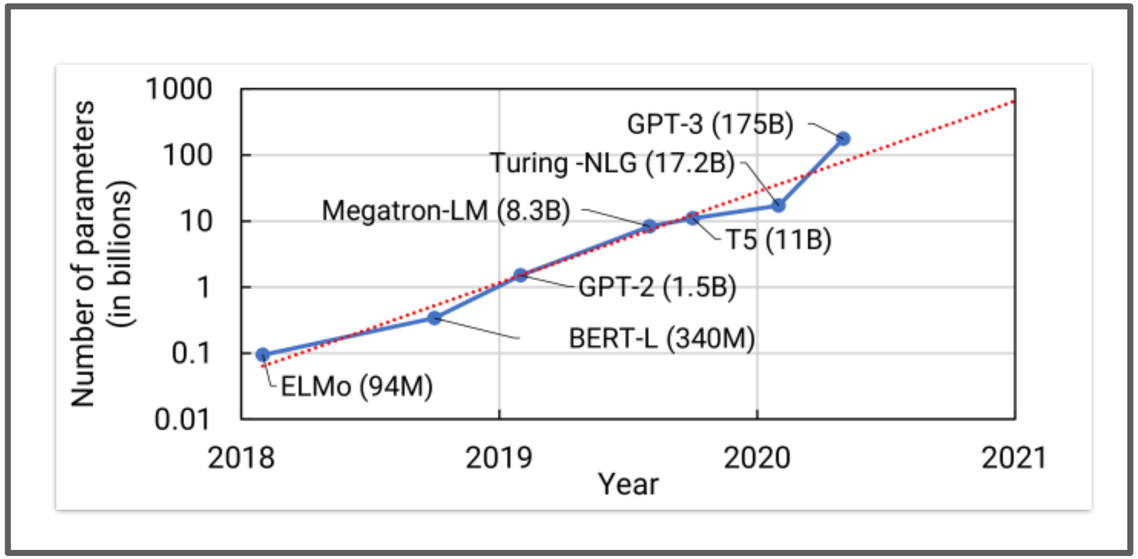
Transformers Architecture





Why “Large” Language Models?

- Large → Big number of parameters



Benefits of Large Language Models

Multi-task, fine tuning, scalability

- **Multi-task Capability:** LLMs find applications in content generation, question answering, translation, tutoring, and personal assistants.

Benefits of Large Language Models

Multi-task, fine tuning, scalability

- **Multi-task Capability:** LLMs find applications in content generation, question answering, translation, tutoring, and personal assistants.
- **Fine-tunning:** LLMs can usually be fine tuned with a relatively small amount of data, making them adaptable to a wide range of tasks.

Benefits of Large Language Models

Multi-task, fine tuning, scalability

- **Multi-task Capability:** LLMs find applications in content generation, question answering, translation, tutoring, and personal assistants.
- **Fine-tunning:** LLMs can usually be fine tuned with a relatively small amount of data, making them adaptable to a wide range of tasks.
- **Scalability**: LLMs demonstrate excellent scalability to very large capacity networks and huge datasets.

Applications of Large Language Models

- Content generation
- Q&A
- Translation
- Tutoring
- Personal Assistants

Applications of LLMs

Content generation Example - DEMO

Applications of LLMs

Translation Example - DEMO

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit:** LLMs have a cutoff point for their knowledge.

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit:** LLMs have a cutoff point for their knowledge.
- **Understanding Limit**: LLMs do not understand text in the same way humans do. They don't have beliefs or desires; they simply predict what comes next based on their training.

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit:** LLMs have a cutoff point for their knowledge.
- **Understanding Limit**: LLMs do not understand text in the same way humans do. They don't have beliefs or desires; they simply predict what comes next based on their training.
- **Misuse:** LLMs can hallucinate and produce false or harmful content

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit:** LLMs have a cutoff point for their knowledge.
- **Understanding Limit**: LLMs do not understand text in the same way humans do. They don't have beliefs or desires; they simply predict what comes next based on their training.
- **Misuse:** LLMs can hallucinate and produce false or harmful content
- **Reproducibility**: Unpredictability of LLM behavior. Watkins 2023

Limitations and Ethical Considerations

LLMs are far from perfect

- **Knowledge Limit:** LLMs have a cutoff point for their knowledge.
- **Understanding Limit**: LLMs do not understand text in the same way humans do. They don't have beliefs or desires; they simply predict what comes next based on their training.
- **Misuse:** LLMs can hallucinate and produce false or harmful content
- **Reproducibility**: Unpredictability of LLM behavior. Watkins 2023
- **Data Privacy and Bias:** Ethical considerations should extend to the acquisition of data for training additional models. Models may have biases; their use should be transparent and biases mitigated. Watkins 2023

Q & A

Introduction to prompt engineering and the ChatGPT API

1. Prompt basics
2. Introduction to the ChatGPT API
3. Prompt engineering guide
4. A Framework for Building Good Prompts

1

Prompt Basics

Introduction to prompts

A prompt is a piece of text that conveys to the LLM the user’s intention.

1

Prompt Basics

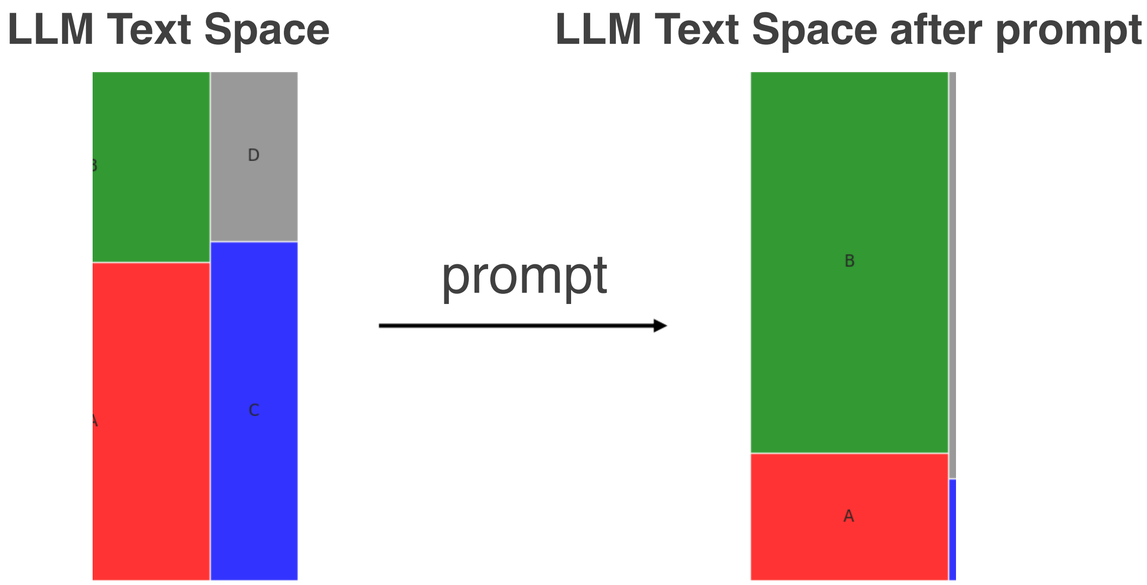
Introduction to prompts

- A prompt is a piece of text that conveys to the LLM the user’s intention.
- Question → Instruction → Behavior

Prompt Basics

Introduction to prompts

- A prompt is a piece of text that conveys to the LLM the user’s intention.
- Question → Instruction → Behavior
- It **constrains the space of possibilities** in the LLM's text space



1

Prompt Basics

Components of a prompt: task, input, context, style

- **Task description:** where you describe what you want

1

Prompt Basics

Components of a prompt: task, input, context, style

- **Task description:** where you describe what you want

- **Input data:** data the model has not seen to illustrate what you need

Prompt Basics

Components of a prompt: task, input, context, style

- **Task description:** where you describe what you want
- **Input data:** data the model has not seen to illustrate what you need
- **Context information:** background info on what you are requesting, the data you are providing etc

Prompt Basics

Components of a prompt: task, input, context, style

- **Task description:** where you describe what you want
- **Context information:** background info on what you are requesting, the data you are providing etc
- **Input data:** data the model has not seen to illustrate what you need
- **Prompt style:** Its about how you ask the thing you want to the model

Prompt Basics

Components of a prompt: task, input, context, style

- **Task description:** where you describe what you want
- **Input data:** data the model has not seen to illustrate what you need
- **Context information:** background info on what you are requesting, the data you are providing etc
- **Prompt style:** Its about how you ask the thing you want to the model

Notebook Demo

Introduction to the ChatGPT API

Where does ChatGPT fit in this chaotic LLM universe?

- The ChatGPT API allows us to use OpenAI's models to generate dynamic, contextually-aware responses.
- Required parameters: model, messages.

```python
import openai

openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
        {"role": "user", "content": "Where was it played?"}
    ]
```

Introduction to the ChatGPT API

Where does ChatGPT fit in this chaotic LLM universe?

```
!pip install openai

openai.api_key = os.getenv("OPENAI_API_KEY")

import openai

openai.ChatCompletion.create(model="gpt-3.5-turbo",
                             messages=
                             [
                                 {"role": "system", "content": "You are a savy guru with knowledge
about existence and the secrets of life."},
                                 {"role": "user", "content": "What is the meaning of life?"}
                             ],
                             max_tokens=100,
                             temperature=0.9,
                             n = 1)
```

Notebook Presentation ChatGPT API

Prompt Engineering Guide

What is prompt engineering?

Prompt engineering: discipline for engineering prompts

Means by which LLMs can be programmed through prompting.

Process of creating a prompting function that results in the most effective performance on the downstream task.

The basic goal of prompt engineering is designing appropriate inputs for prompting methods.

Prompt Engineering Guide

A simplified guide of prompting techniques

- Zero-shot Prompting
- Few-shot Prompting
- Chain-of-Thought
- Self-Consistency
- Generate Knowledge
- Tree of thoughts (ToT)

Prompt Engineering Guide

A simplified guide of prompting techniques

Zero-shot prompting is when you solve the task without showing any examples of what a solution might look like

Prompt Engineering Guide

A simplified guide of prompting techniques

- **Zero-shot prompting** is when you solve the task without showing any examples of what a solution might look like

- One can use this as the first try at a model to see what kind of tasks LLM can already solve out of the box

Prompt Engineering Guide

A simplified guide of prompting techniques

- **Zero-shot prompting** is when you solve the task without showing any examples of what a solution might look like

- One can use this as the first try at a model to see what kind of tasks LLM can already solve out of the box

Classify the sentiment in this sentence as negative or positive:
***Text**: I will go to a vacation.*
Sentiment:

Few-shot Prompting

Provide information in the form of examples to the LLM

Few-shot Prompting: technique where you show a few examples of what a solution might look like.

Few-shot Prompting

Provide information in the form of examples to the LLM

Few-shot Prompting: technique where you show a few examples of what a solution might look like.

A "whatpu" is a small, furry animal native to Tanzania.
An example of a sentence that uses the word whatpu is: We were traveling in Africa and we saw these very cute whatpus.
To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:

Example taken from (Brown et al. 2020)

Chain-of-Thought

Induce step-by-step reasoning and planning

Chain-of-thought (CoT) enables complex reasoning capabilities through intermediate reasoning steps (Wei et al. 2022).

Chain-of-Thought

Induce step-by-step reasoning and planning

Chain-of-thought (CoT) enables complex reasoning capabilities through intermediate reasoning steps (Wei et al. 2022).

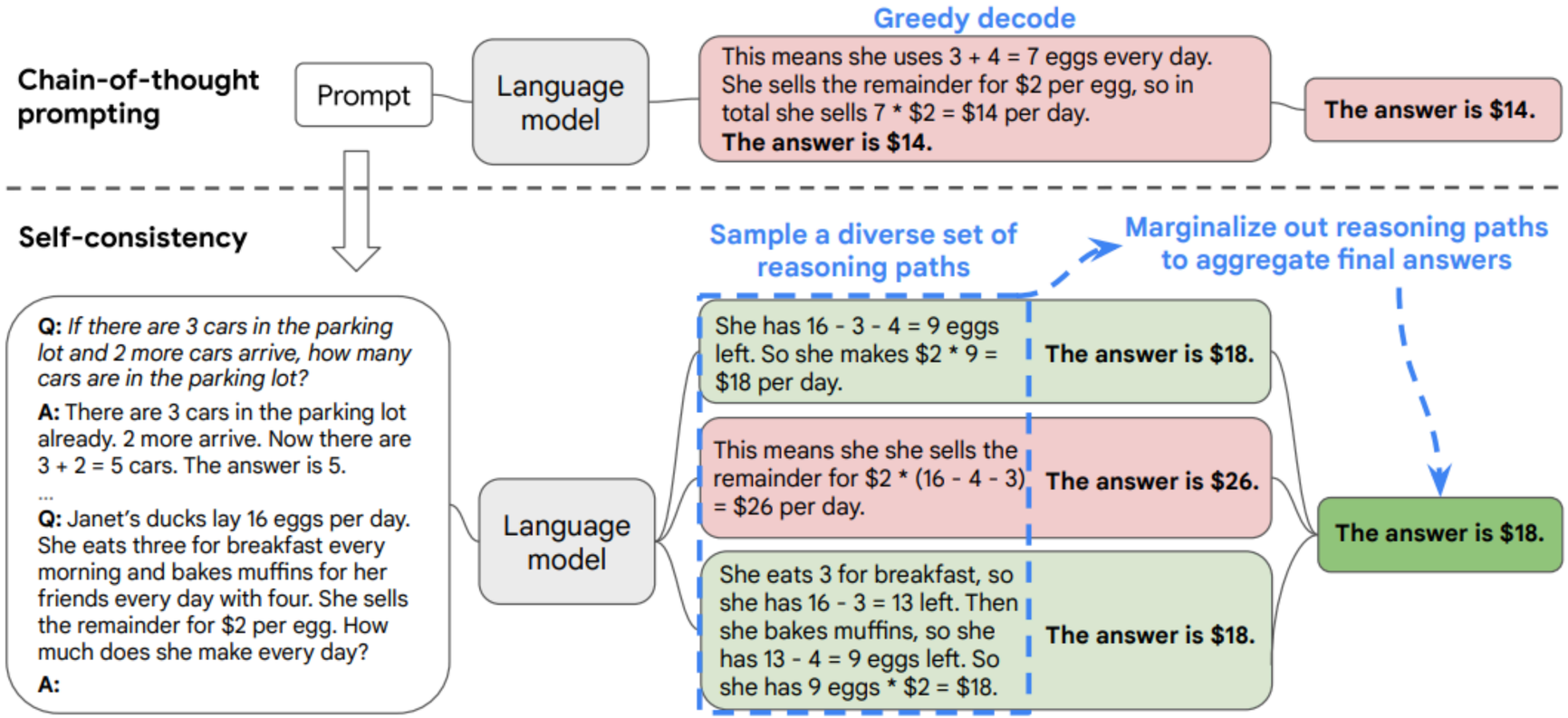
Q: I have one sister and one brother. I am 20 years of age. My sister is 5 years older and my brother 2 years younger than my sister. How old is my brother?

A: If I am 20 years of age and my sister is 5 years older, my sister is 20+5=25 years old. If my brother is 2 years younger than my sister, my brother is 25-2=23 years old. The answer is 23 years old.

Q: I have 2 friends, Jack and Sally. Jack is 2 years older than Sally. Sally is 5 years younger than me. I am 17 years old. How old is Jack?

Self-Consistency

Example

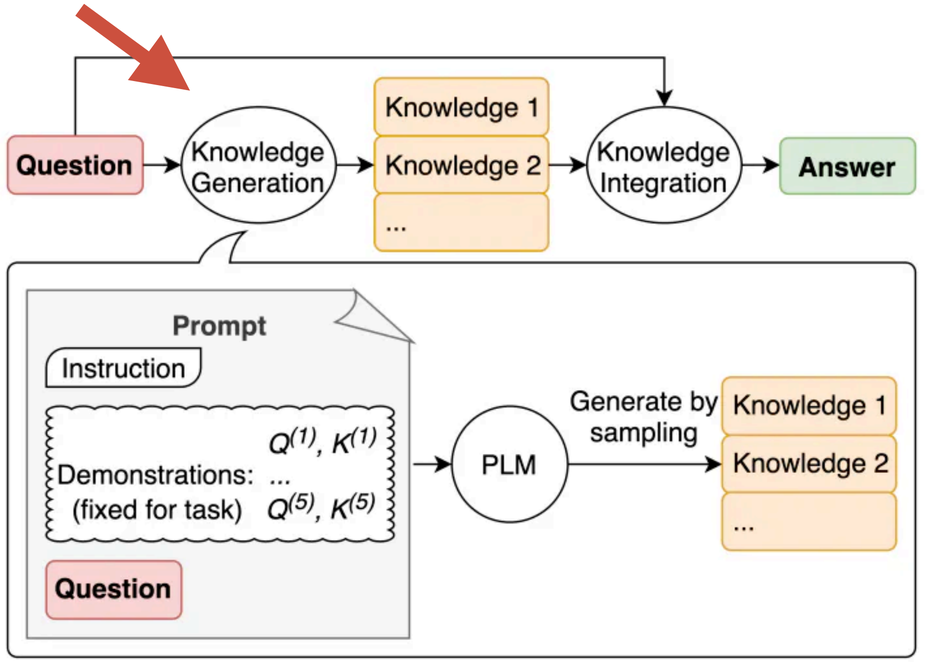


https://arxiv.org/pdf/2203.11171.pdf

Generate Knowledge

Example

- **Knowledge Generation**: Generating facts related to the question.

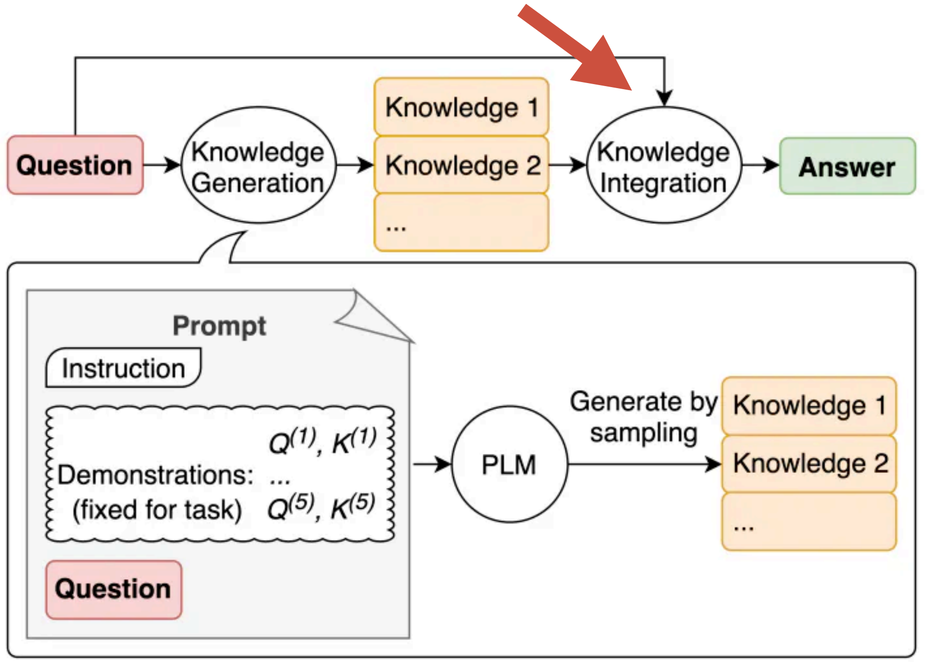


https://arxiv.org/pdf/2110.08387.pdf

Generate Knowledge

Example

- **Knowledge Generation**: Generating facts related to the question.
- **Knowledge Integration**: Using generated facts to answer the question comprehensively.



https://arxiv.org/pdf/2110.08387.pdf

There are many more prompt engineering techniques that grow in complexity, such as:

- ToT (Yao et al. 2023)
- Retrieval Augmented Generation (Lewis et el. (2021))
- Automatic Prompt Engineer (Zhou et al., (2022))
- React Prompting (Yao et al., 2022)
- Graph Prompting (Liu et al., 2023)
- Skeleton of Thought (Ning et al. 2023)
- Step Back Prompting (Zheng et al. 2023)

Break

A Framework for Building Good Prompts

4 principles: structured text, decomposition, self-criticism, ensembling

- Operate on Structured Text
-
-
-

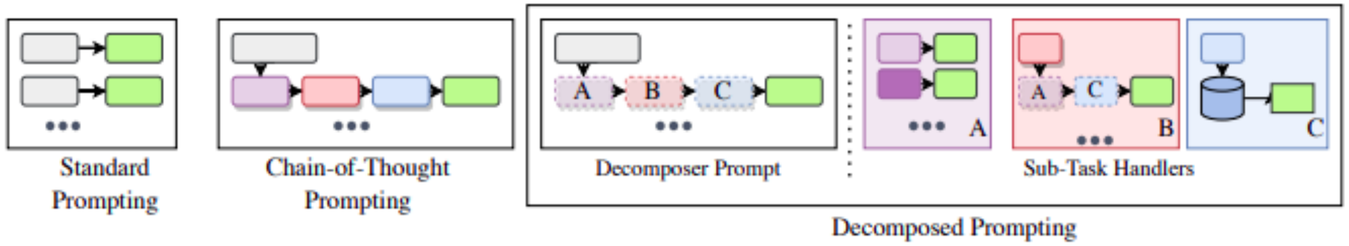
```
<python 3 shebang>

<module docstring>

<imports>


<do not include email dunder>

<initialize dotenv>
<set key using OPENAI_API KEY env var>

def complete(prompt: str, **openai_kwargs) -> str:
    <one-line docstring; no params>
    <use default kwargs: model=text-davinci-003, top_p=0.7,
max_tokens=512, 'engine' parameter is deprecated>
    <get completion>
    <strip whitespace before returning>

<as script, demo using prompt "English: Hello\nFrench:">
```

@goodside

A Framework for Building Good Prompts

4 principles: structured text, decomposition, self-criticism, ensembling

- Operate on Structured Text
- Introduce decomposition
-
-



QC: Concatenate the first letter of every word in "Jack Ryan" using spaces
Q1: [split] What are the words in "Jack Ryan"?
#1: ["Jack". "Ryan"]
Q2: (foreach) [str_pos] What is the first letter of #1?
#2: ["J", "R"]
Q3: [merge] Concatenate #2 with spaces
#3: "J R"
Q4: [EOQ]
...
decomp

Q: What are the words in "Elon Musk Tesla"?
A: ["Elon", "Musk", "Tesla"]

Q: What are the letters in "C++"?
A: ["C", "+", "+"]
...
split

Q: Concatenate ["n", "i", "e"]
A: "nie"

Q: Concatenate ["n", "i", "c", "e"] using spaces
A: "n i c e"
...
merge

[Khot et al 2023]

A Framework for Building Good Prompts

4 principles: structured text, decomposition, self-criticism, ensembling

- Operate on Structured Text
- Introduce decomposition
-
-

Self-Ask

GPT-3

Question: Who lived longer, Theodor Haecker or Harry Vaughan Watkins?
Are follow up questions needed here: Yes.
Follow up: How old was Theodor Haecker when he died?
Intermediate answer: Theodor Haecker was 65 years old when he died.
Follow up: How old was Harry Vaughan Watkins when he died?
Intermediate answer: Harry Vaughan Watkins was 69 years old when he died.
So the final answer is: Harry Vaughan Watkins

Question: Who was president of the U.S. when superconductivity was discovered?
Are follow up questions needed here: Yes.
Follow up: When was superconductivity discovered?
Intermediate answer: Superconductivity was discovered in 1911.
Follow up: Who was president of the U.S. in 1911?
Intermediate answer: William Howard Taft.
So the final answer is: William Howard Taft.

Press et al 2023

A Framework for Building Good Prompts

4 principles: structured text, decomposition, self-criticism, ensembling

- Operate on Structured Text
- Introduce decomposition
-
-

(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✔

Kojima et al 2023

A Framework for Building Good Prompts

4 principles: structured text, decomposition, self-criticism, ensembling

- Operate on Structured Text
- Introduce decomposition
- Self-criticism
-

Q: Valerie earns $5000 per month, 1/2 of what her brother earns. If their mother earns twice their combined salary, what's the total amount of money they all have together?

Zero-Shot

A:
Valerie's brother earns $10,000 per month, while Valerie herself earns half of that amount. Together, they earn a total of $15,000 per month. Their mother's monthly earnings amount to $30,000, which is twice the combined salary of Valerie and her brother. Thus, the total amount of money they all have together is $30,000. **(Wrong)**

Few-Shot-CoT

<Few shot examples>
A:
Valerie earns $5000 per month, half of her brother's earnings, making his monthly income $10000. Combined, Valerie and her brother earn $15000 per month. Their mother earns double their combined salary, amounting to $30000 per month. In total, they have $30000 collectively. **(Wrong)**

Zero-Shot-CoT

A: Let's think step by step. First, Valerie's brother earns 2x and Valerie earns x (half of her brother's earnings). Next, their mother's income is twice the combined earnings of Valerie and her brother. In this case, their total income amounts to $30,000. **(Wrong)**

RCI

<Initial output generation>

A: Valerie's brother earns $10,000 per month, while Valerie herself earns half of that amount. Together, they earn a total of $15,000 per month. Their mother's monthly earnings amount to $30,000, which is twice the combined salary of Valerie and her brother. Thus, the total amount of money they all have together is $30,000.

<Critique>

Review your previous answer and find problems with your answer.

All individuals, including Valerie, her brother, and mother, should be taken into account when calculating the overall sum of money.

<Improve>

Based on the problems you found, improve your answer.

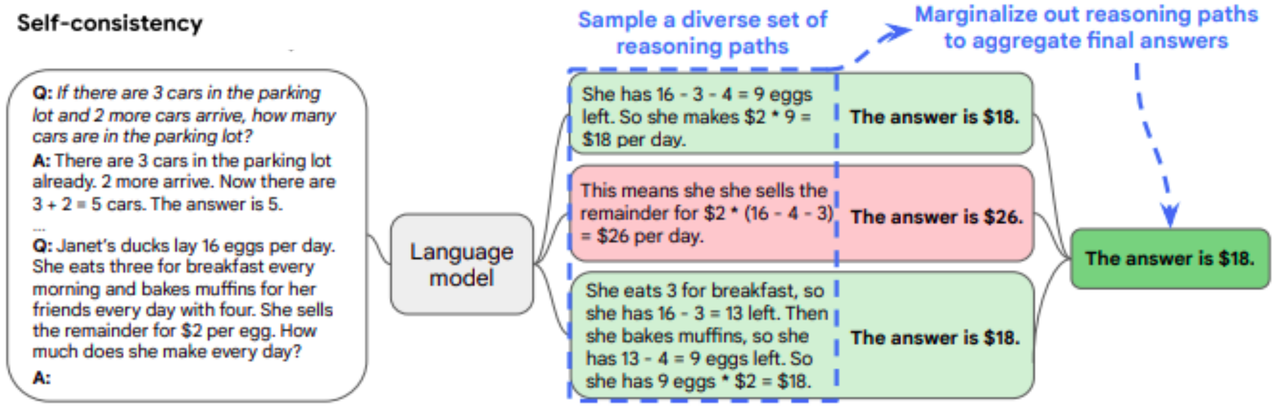
Valerie's brother earns $10,000 per month, while Valerie herself earns $5,000. Together, they earn a total of $15,000 per month. Their mother earns double their combined salary, which amounts to $30,000 per month. In total, they have $45,000 collectively. **(Correct)**

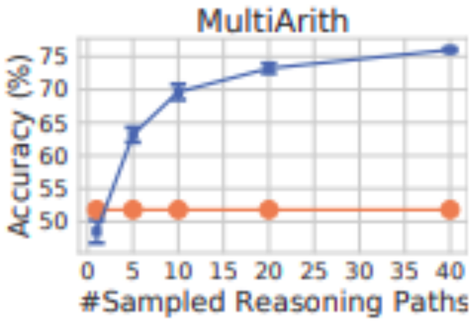
Kim, Baldi & McAleer 2023

A Framework for Building Good Prompts

4 principles: structured text, decomposition, self-criticism, ensembling

- Operate on Structured Text
- Introduce decomposition
- Self-criticism
- Ensembling



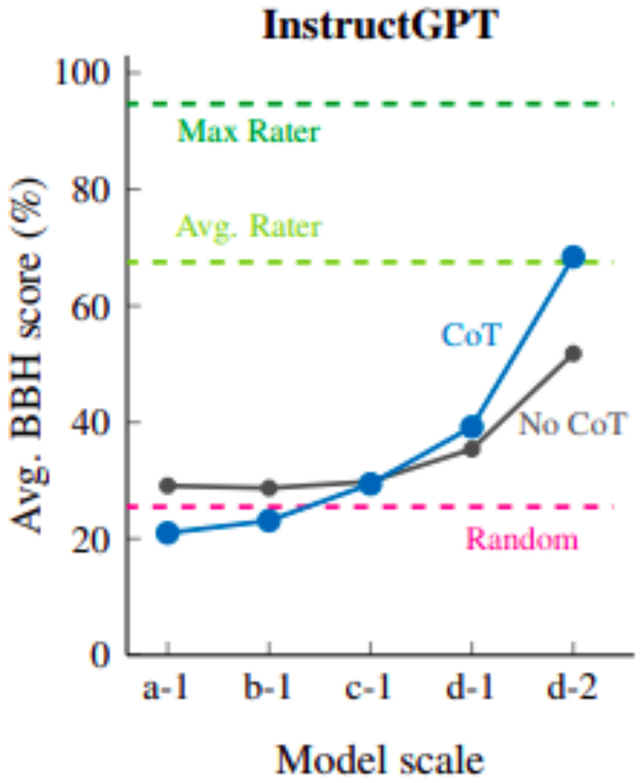


[Wang et al 2023]

A Framework for Building Good Prompts

4 principles: structured text, decomposition, self-criticism, ensembling

- Operate on Structured Text
- Introduce decomposition
- Self-criticism
- Ensembling



Combine for better performance!

Suzgun et al 2022

A Framework for Building Good Prompts

4 principles: structured text, decomposition, self-criticism, ensembling

- Operate on Structured Text
- Introduce decomposition
- Self-criticism
- Ensembling

	Latency	Compute
Few-Shot CoT	⬆️⬆️	💰💰
Zero-Shot CoT	⬆️	💰
Decomposition	⬆️⬆️	💰💰
Ensembling	✅	💰💰💰💰
Self-Criticism	⬆️⬆️⬆️⬆️	💰💰

Be aware of quality-cost tradeoff!

[FSDL LLM Bootcamp 2023]

Exercise / Lab

- Prompt engineering using the ChatGPT API
- Extracting dates from unstructured data
- Prompt engineering for text summarization and question answering

Notebook Lab Exercises

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning:

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning: process of “specialising an LLM on your specific dataset”

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning: process of “specialising an LLM on your specific dataset”

Why?

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning: process of “specialising an LLM on your specific dataset”

Why? Better results on specific tasks, lower latency… **See OpenAI docs**

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning: process of “specialising an LLM on your specific dataset”

Why? Better results on specific tasks, lower latency… **See OpenAI docs**

How?

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning: process of “specialising an LLM on your specific dataset”

Why? Better results on specific tasks, lower latency… **See OpenAI docs**

How? By preparing a custom `.jsonl` dataset on your specific use case

Fine Tuning ChatGPT Applications

A primer on how to fine tune ChatGPT on your data

Fine Tuning: process of “specialising an LLM on your specific dataset”

Why? Better results on specific tasks, lower latency… **See OpenAI docs**

How? By preparing a custom `.jsonl` dataset on your specific use case

The goal of fine tuning is to adapt the model to your specific use case by providing it with a few examples of successful interactions

3

Fine Tuning ChatGPT Applications

Common Use Cases

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format

[OpenAI docs]

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures

[OpenAI docs]

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures
- Improving reliability

OpenAI docs

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures
- Improving reliability
- Handling edge cases

OpenAI docs

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures
- Improving reliability
- Handling edge cases
- Extracting structured output

OpenAI docs

Fine Tuning ChatGPT Applications

Common Use Cases

- Setting style, tone, format
- Correcting failures
- Improving reliability
- Handling edge cases
- Extracting structured output
- Function Calling

OpenAI docs

Fine Tuning ChatGPT Applications

Basic Steps

OpenAI docs

Fine Tuning ChatGPT Applications

Basic Steps

- Prepare and upload training data

[OpenAI docs]

Fine Tuning ChatGPT Applications

Basic Steps

- Prepare and upload training data
- Train a new fine-tuned model

OpenAI docs

Fine Tuning ChatGPT Applications

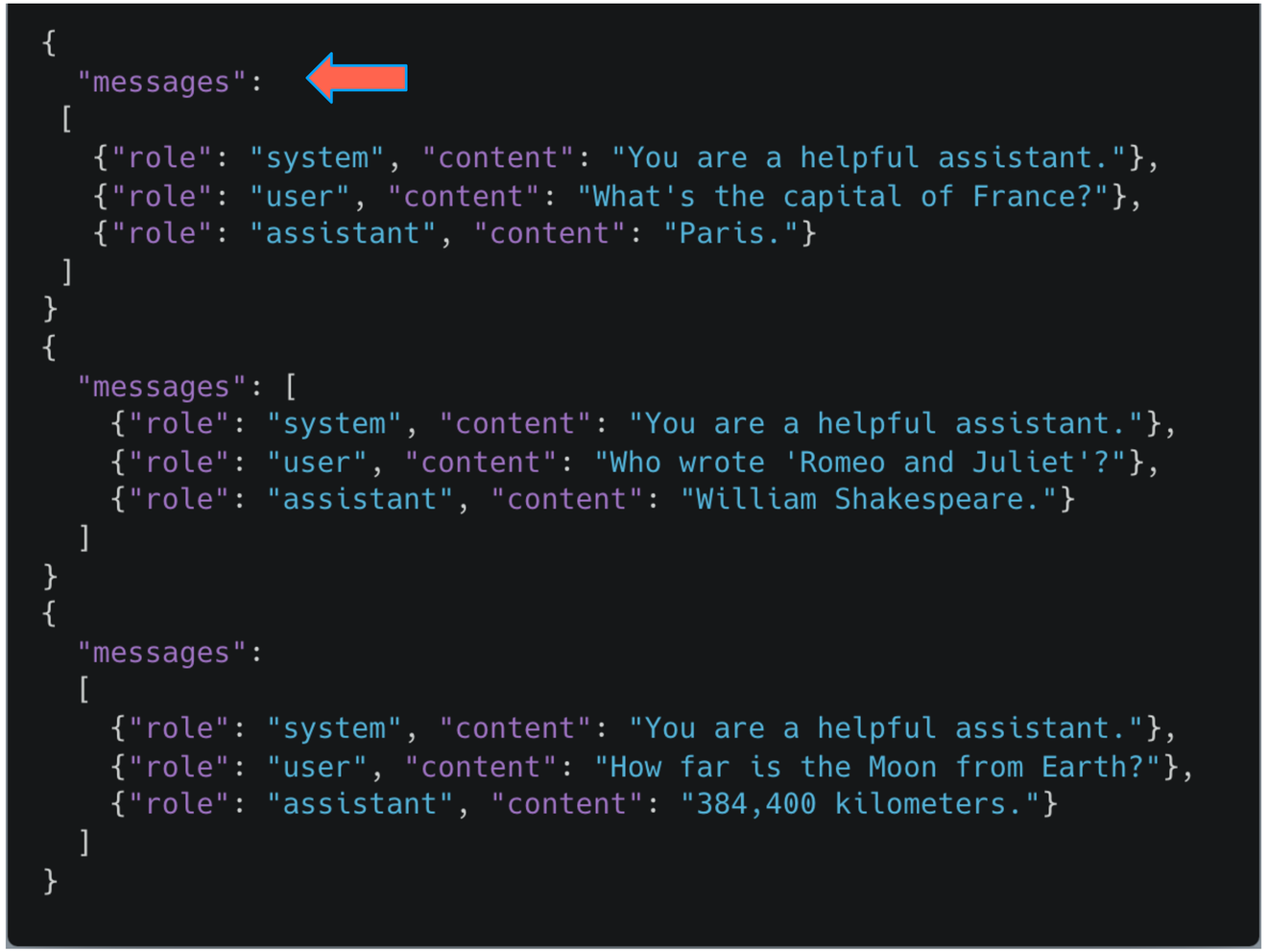
Basic Steps

- Prepare and upload training data
- Train a new fine-tuned model
- Use your fine-tuned model

OpenAI docs

Fine Tuning ChatGPT Applications

Prepare and upload training data

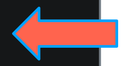


```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```



Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Prepare and upload training data

```
{
  "messages":
 [
   {"role": "system", "content": "You are a helpful assistant."},
   {"role": "user", "content": "What's the capital of France?"},
   {"role": "assistant", "content": "Paris."}
 ]
}
{
  "messages": [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"},
    {"role": "assistant", "content": "William Shakespeare."}
  ]
}
{
  "messages":
  [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "How far is the Moon from Earth?"},
    {"role": "assistant", "content": "384,400 kilometers."}
  ]
}
```

Fine Tuning ChatGPT Applications

Fine Tuning Jobs

```python
from openai import OpenAI
# OpenAI API key should be set as
# environment variable - OPENAI_API_KEY
client = OpenAI()
client.files.create(
  file=open("dataset.jsonl", "rb"),
  purpose="fine-tune"
)
client.fine_tuning.jobs.create(
  training_file="file-id",
  model="gpt-3.5-turbo-1106"
)
```

Notebook Presentation Fine Tuning with ChatGPT API

Summary so far

Break

Langchain for LLM App Development

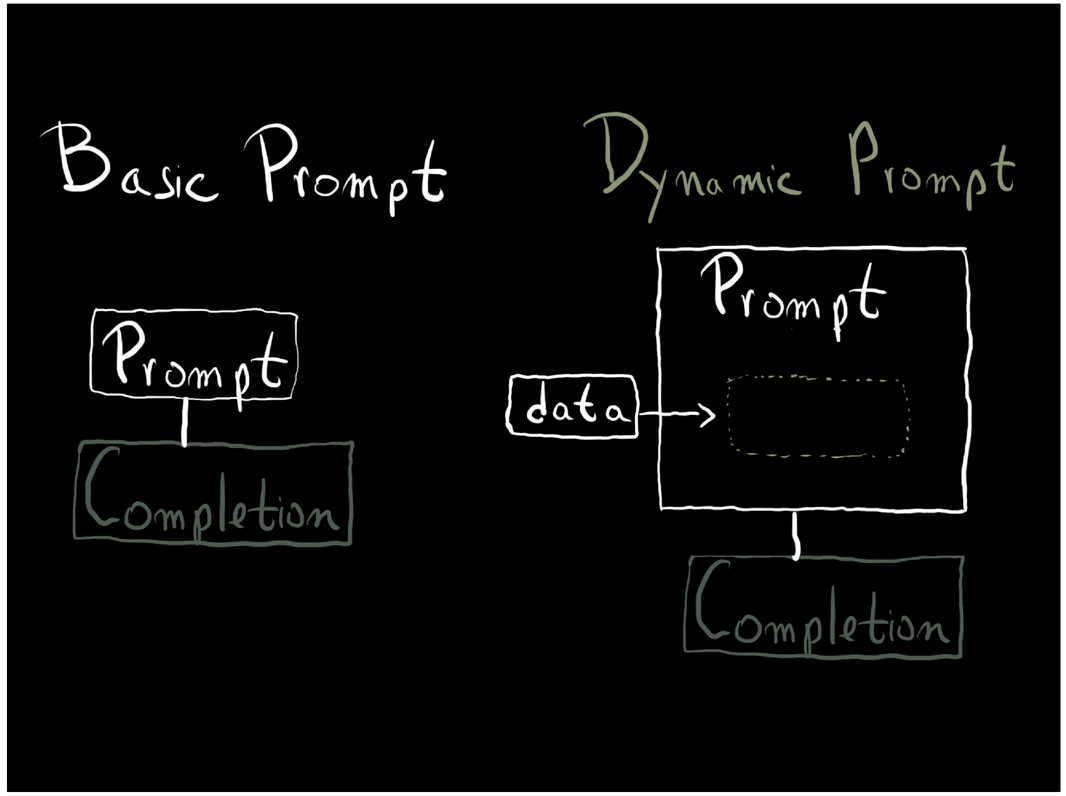
From static prompts to dynamic prompts

- Prompt management workflows that require dynamic prompts

Langchain for LLM App Development

From static prompts to dynamic prompts

- Prompt management workflows that require dynamic prompts
- This dynamics requirement leads to the need for creating certain types of abstractions



Langchain for LLM App Development

Langchain framework

- Langchain is a framework that facilitates the creation and management of dynamic prompts and chaining between prompts.
- **Main features:** components and off-the-shelf-chains.

Basics of Langchain

Langchain components

- **Models:** abstractions over the LLM APIs like the ChatGPT API

Basics of Langchain

Langchain components

- **Models:** abstractions over the LLM APIs like the ChatGPT API
- **Prompt Templates:** abstraction over standard prompts to LLMs

Basics of Langchain

Langchain components

- **Models:** abstractions over the LLM APIs like the ChatGPT API
- **Prompt Templates:** abstraction over standard prompts to LLMs
- **Output Parsers:** Translates raw output from LLM to a workable format

https://python.langchain.com/docs/get_started/quickstart

Basics of Langchain

Langchain components

- **Models:** abstractions over the LLM APIs like the ChatGPT API
- **Prompt Templates:** abstraction over standard prompts to LLMs
- **Output Parsers:** Translates raw output from LLM to a workable format

Notebook demo

https://python.langchain.com/docs/get_started/quickstart

Basics of Langchain

Models, Prompt Templates and Output Parsers

Models: LLMs and Chat models

Basics of Langchain

Models, Prompt Templates and Output Parsers

- **Models:** LLMs and Chat models
- **LLMs:** models that take a text string as input and return a text string

```python
from langchain.llms import OpenAI

llm = OpenAI()

llm.predict("Hi")
```

Basics of Langchain

Models, Prompt Templates and Output Parsers

- **Models:** LLMs and Chat models
- **LLMs:** models that take a text string as input and return a text string
- **Chat models:** takes a list of Chat Messages as input and returns a Chat Message

```python
from langchain.chat_models import ChatOpenAI

chat_model = ChatOpenAI()

chat_model.predict("Hi")

# or

chat_model.invoke("Hi")
```

Basics of Langchain

Models, Prompt Templates and Output Parsers

Messages?: inherits from BaseMessage interface

Basics of Langchain

Models, Prompt Templates and Output Parsers

- **Messages?:** inherits from BaseMessage interface
- **content:** content of the message (usually a string)

Basics of Langchain

Models, Prompt Templates and Output Parsers

- **Messages?:** inherits from BaseMessage interface
- **content:** content of the message (usually a string)
- **role:** entity from which the BaseMessage is coming

Basics of Langchain

Models, Prompt Templates and Output Parsers

- **Messages?:** inherits from BaseMessage interface
- **content:** content of the message (usually a string)
- **role:** entity from which the BaseMessage is coming

Objects to distinguish between different roles:

HumanMessage: A BaseMessage coming from a human/user.
AIMessage: A BaseMessage coming from an AI/ assistant.
SystemMessage: A BaseMessage coming from the system.
FunctionMessage / ToolMessage: A BaseMessage containing the output of a function or tool call.

Basics of Langchain

Models, **Prompt Templates** and Output Parsers

- **Prompt Templates:** pre-defined recipes for generating prompts for LLMs

```python
from langchain.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(
    "Tell me a {adjective} story about {content} in one paragraph."
)
prompt_template.format(adjective="funny", content="chickens")
```

Basics of Langchain

Models, **Prompt Templates** and Output Parsers

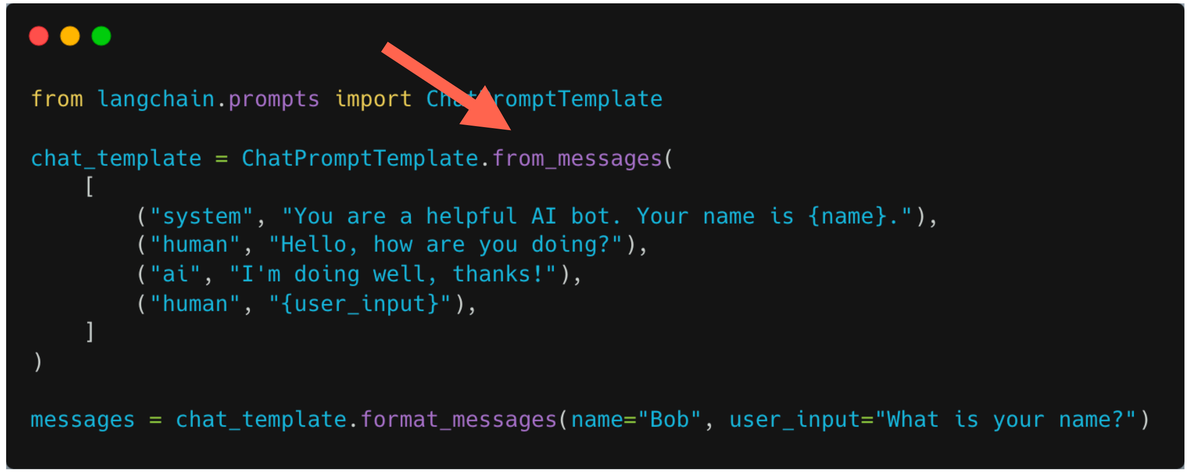
- **Prompt Templates:** pre-defined recipes for generating prompts for LLMs
- **ChatPromptTemplates:** prompt template for chat models

```python
from langchain.prompts import ChatPromptTemplate

chat_template = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful AI bot. Your name is {name}."),
        ("human", "Hello, how are you doing?"),
        ("ai", "I'm doing well, thanks!"),
        ("human", "{user_input}"),
    ]
)

messages = chat_template.format_messages(name="Bob", user_input="What is your name?")
```

Basics of Langchain

Models, **Prompt Templates** and Output Parsers

- **Prompt Templates:** pre-defined recipes for generating prompts for LLMs
- **ChatPromptTemplates:** prompt template for chat models
- **ChatPromptTemplates** are designed as a list of chat messages.



```python
from langchain.prompts import ChatPromptTemplate

chat_template = ChatPromptTemplate.from_messages(
    [
        ("system", "You are a helpful AI bot. Your name is {name}."),
        ("human", "Hello, how are you doing?"),
        ("ai", "I'm doing well, thanks!"),
        ("human", "{user_input}"),
    ]
)

messages = chat_template.format_messages(name="Bob", user_input="What is your name?")
```

Basics of Langchain

Models, Prompt Templates and **Output Parsers**

- **Output Parsers:** convert raw output of an LLM into a usable format

Basics of Langchain

Models, Prompt Templates and **Output Parsers**

- **Output Parsers:** convert raw output of an LLM into a usable format
- **Key Types**

- **Converting LLM text to structured formats (e.g., JSON).**
- **Transforming a ChatMessage into a simple string.**
- **Converting additional information from model calls into a string.**

Basics of Langchain

Models, Prompt Templates and **Output Parsers**

- **Output Parsers:** convert raw output of an LLM into a usable format
- **Key Types**
- **Writing a Custom Output Parser**

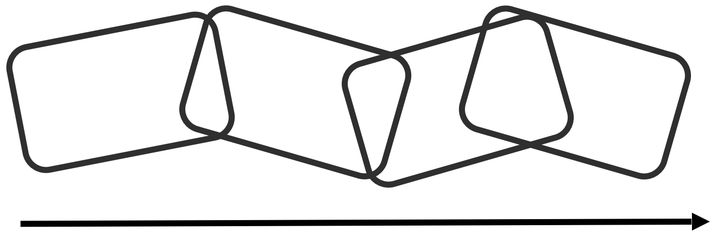
```python
from langchain.schema import BaseOutputParser

class CommaSeparatedListOutputParser(BaseOutputParser):
    """Parse the output of an LLM call to a comma-separated list."""


    def parse(self, text: str):
        """Parse the output of an LLM call."""
        return text.strip().split(", ")

CommaSeparatedListOutputParser().parse("hi, bye")
# >> ['hi', 'bye']
```

Notebook demo

Basics of Langchain

Models, Prompt Templates and Output Parsers

- **Chains:** prompt template + Model + output parsing



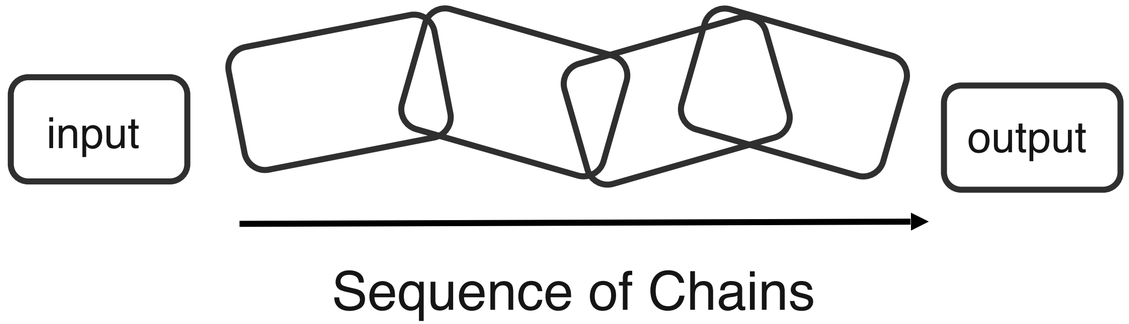
Chains of Prompts

Basics of Langchain

LLMChain & Sequential Chains

- **Chains:** prompt template + Model + output parsing

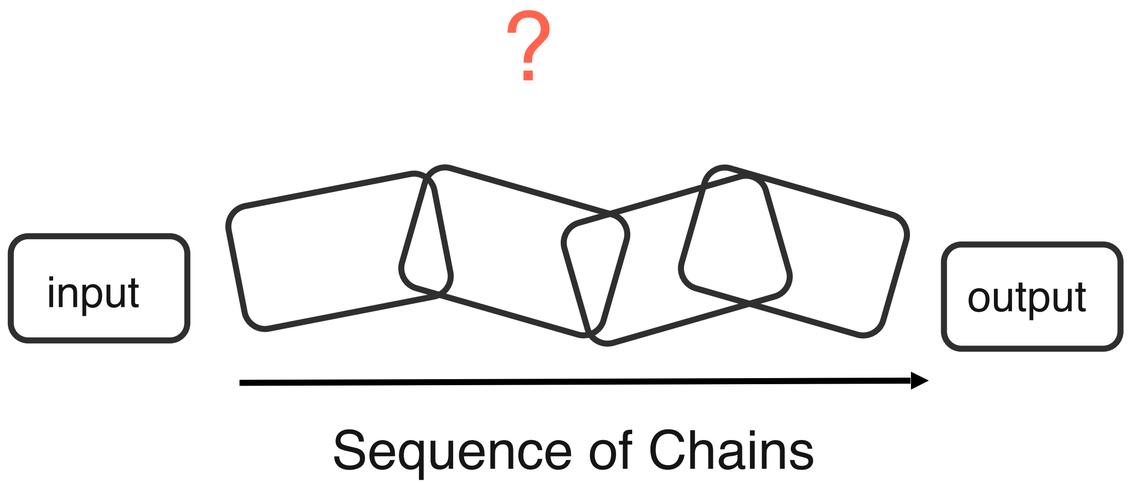
- Sequence of calls to an LLM chained together



Basics of Langchain

LLMChain & Sequential Chains

- **Chains:** prompt template + Model + output parsing
- Sequence of calls to an LLM chained together
- How to build these chains?



Basics of Langchain

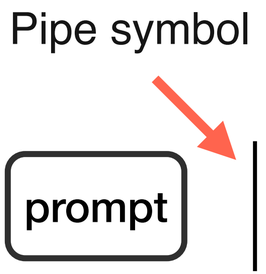
LCEL - LangChain Expression Language

LCEL - designed to build sequence of calls (to LLMs or any other component)

Basics of Langchain

LCEL - LangChain Expression Language

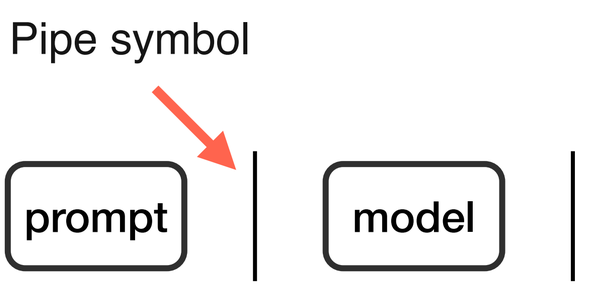
- **LCEL** - designed to build sequence of calls (to LLMs or any other component)
- Pipe syntax: prompt **|**



Basics of Langchain

LCEL - LangChain Expression Language

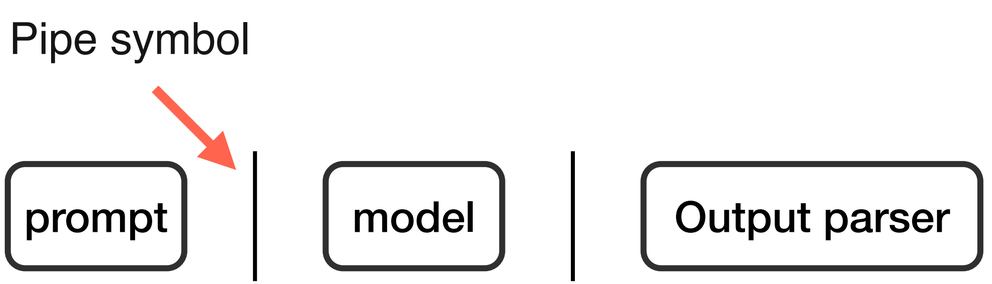
- **LCEL** - designed to build sequence of calls (to LLMs or any other component)
- Pipe syntax: prompt **|** model **|**



Basics of Langchain

LCEL - LangChain Expression Language

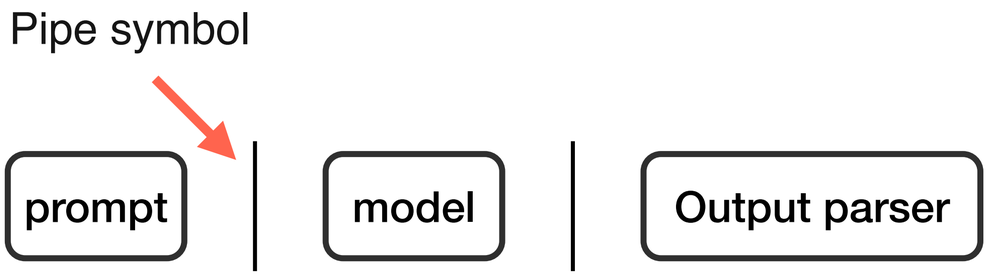
- **LCEL** - designed to build sequence of calls (to LLMs or any other component)
- Pipe syntax: prompt **|** model **|** output parser



Basics of Langchain

LCEL - LangChain Expression Language

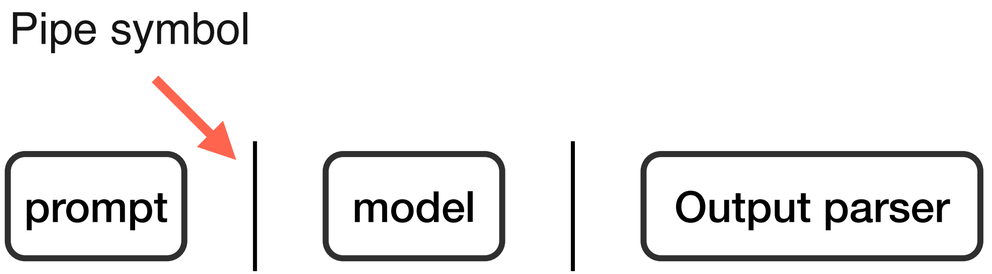
- **LCEL** - designed to build sequence of calls (to LLMs or any other component)
- Pipe syntax: prompt **|** model **|** output parser
- Allows you to **build complex chain pipelines** with a simple standard interface



Basics of Langchain

LCEL - LangChain Expression Language

- **LCEL** - designed to build sequence of calls (to LLMs or any other component)
- Pipe syntax: prompt **|** model **|** output parser
- Allows you to **build complex chain pipelines** with a simple standard interface



Notebook demo

Basics of LangChain

LCEL - LangChain Expression Language

stream: stream back chunks of the response

Basics of LangChain

LCEL - LangChain Expression Language

- ***stream***: stream back chunks of the response
- ***invoke***: call the chain on an input

Basics of LangChain

LCEL - LangChain Expression Language

- ***stream***: stream back chunks of the response
- ***invoke***: call the chain on an input
- ***batch:*** call the chain on a list of inputs

Basics of LangChain

LCEL - LangChain Expression Language

- ***stream***: stream back chunks of the response
- ***invoke***: call the chain on an input
- ***batch:*** call the chain on a list of inputs

- ***Runnable Protocol***: standard interface to facilitate defining custom chains

Basics of LangChain

LCEL - LangChain Expression Language

- ***stream***: stream back chunks of the response
- ***invoke***: call the chain on an input
- ***batch:*** call the chain on a list of inputs
- ***Runnable Protocol***: standard interface to facilitate defining custom chains
- ***Input schema***: description of the inputs accepted by a Runnable

Basics of LangChain

LCEL - LangChain Expression Language

- ***stream***: stream back chunks of the response
- ***invoke***: call the chain on an input
- ***batch:*** call the chain on a list of inputs
- ***Runnable Protocol***: standard interface to facilitate defining custom chains
- ***Input schema***: description of the inputs accepted by a Runnable
- ***Output schema***: description of the output produced by a Runnable

Basics of LangChain

LCEL - LangChain Expression Language

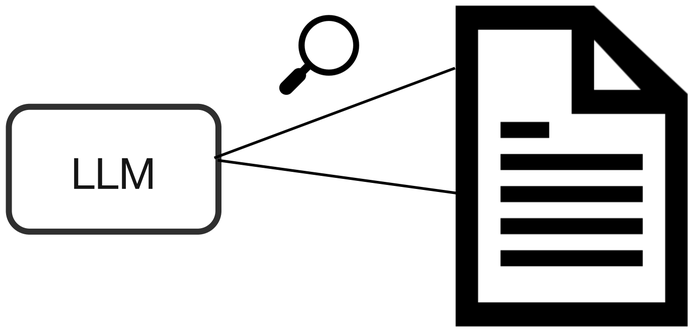
- ***stream***: stream back chunks of the response
- ***invoke***: call the chain on an input
- ***batch:*** call the chain on a list of inputs
- ***Runnable Protocol***: standard interface to facilitate defining custom chains
- ***Input schema***: description of the inputs accepted by a Runnable
- ***Output schema***: description of the output produced by a Runnable

Notebook demo

Langchain for LLM App Development

Langchain with Documents

- LLMs have a limited context length



Langchain for LLM App Development

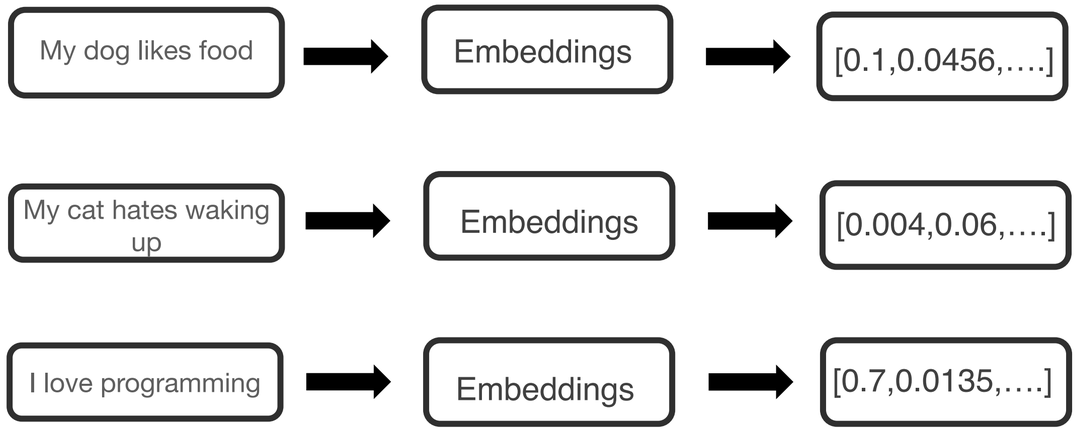
Langchain with Documents

- LLMs have a limited context length
- **Embeddings**: capture content and meaning

Langchain for LLM App Development

Langchain with Documents

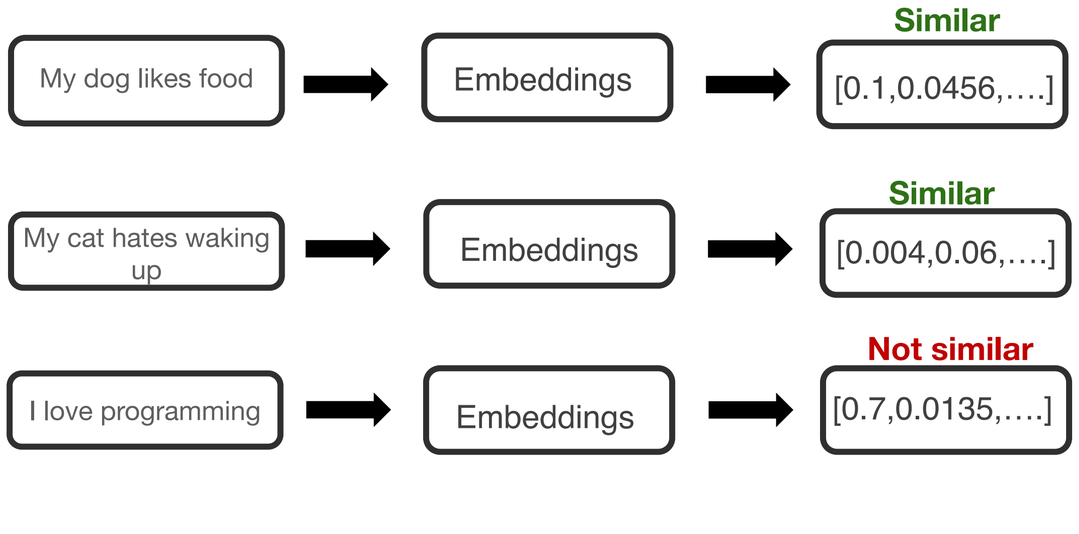
- LLMs have a limited context length
- **Embeddings**: capture content and meaning



Langchain for LLM App Development

Langchain with Documents

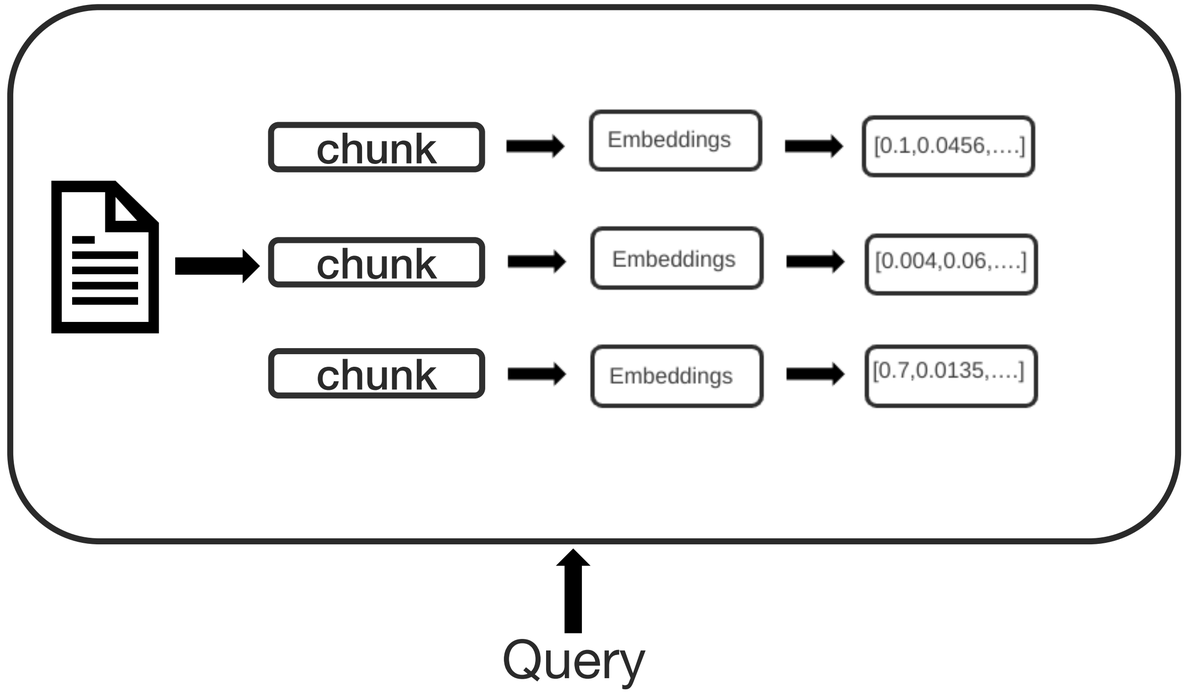
- LLMs have a limited context length
- **Embeddings**: capture content and meaning



Langchain for LLM App Development

Langchain with Documents

- LLMs have a limited context length
- Embeddings: capture content and meaning
- **Vector DBs**



Langchain for LLM App Development

Langchain with Documents

- LLMs have a limited context length
- Embeddings: capture content and meaning
- Vector DBs

Chunking methods for large docs

- Map_reduce
- Refine
- Map_rerank

Langchain for LLM App Development

Langchain with Documents

- LLMs have a limited context length
- Embeddings: capture content and meaning
- Vector DBs

Chunking methods for large docs

- Map_reduce
- Refine
- Map_rerank

Notebook demo