Understanding Large Language Models: A Comprehensive Overview, LLM Project Life Cycle & OpenAI API

El Habib NFAOUI, Ph.D.
Full Professor
Department of Computer Science,
Faculty of Sciences Dhar El Mahraz,
Sidi Mohamed Ben Abdellah University, Fez
elhabib.nfaoui@usmba.ac.ma

IEEE Morocco Section
Computational Intelligence Society Chapter

International Neural Network Society
INNS-Morocco Regional Chapter

December 22, 2023

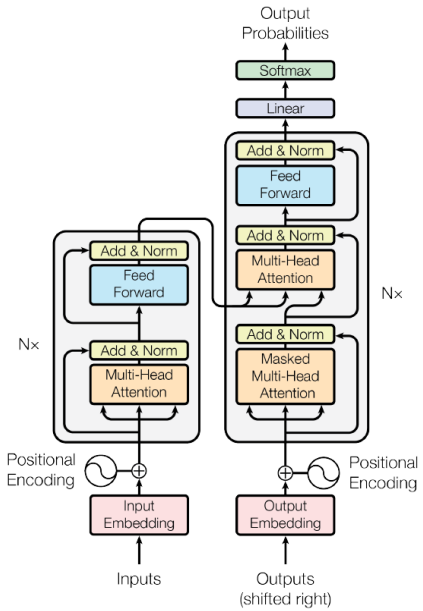
Outline

- **LLMs**:
  - Language Modeling Objective, Architectures, Decoding strategy,
  - Types of LLMs
  - In-context learning, Fine-tuning
  - Augmented LLMs
  - Retrieval Augmented Generation (RAG) and LLM-powered applications
  - Generative AI life cycle project

- **OpenAI API**
  - Chat Completions API endpoint
  - Fine-tuning API endpoint
  - Assistants API endpoint (beta)
  - Embeddings API endpoint

1. Definition of LLMs

Prompt (input) → **LLM** → Completion (Output)



The original Transformer model described in *"Attention is All You Need"* (Vaswani et al., 2017)

" Typically, large language models (LLMs) refer to **Transformer language models** that contain **hundreds of billions (or more) of parameters***, which are trained on **massive text data** [1], such as GPT-3 [2], PaLM [3], Galactica [4], and LLaMA [5]. LLMs exhibit strong capacities to understand natural language and solve complex tasks (**via text generation**)." (Wayne Xin Zhao et al., 2023)

Model parameters size:
GPT-3 (2020):175B
PaLM (2022): 540B

Pre-train Data Scale:
GPT-3 (2020): 300B tokens
PaLM (2022): 780B tokens

Prompt (NLP spell-checking task)

Proofread the following text and rewrite the corrected version:
"He did not did its homework."

→ **GPT-3.5** →

Completion

"He did not do his homework."

(NLP translation task)

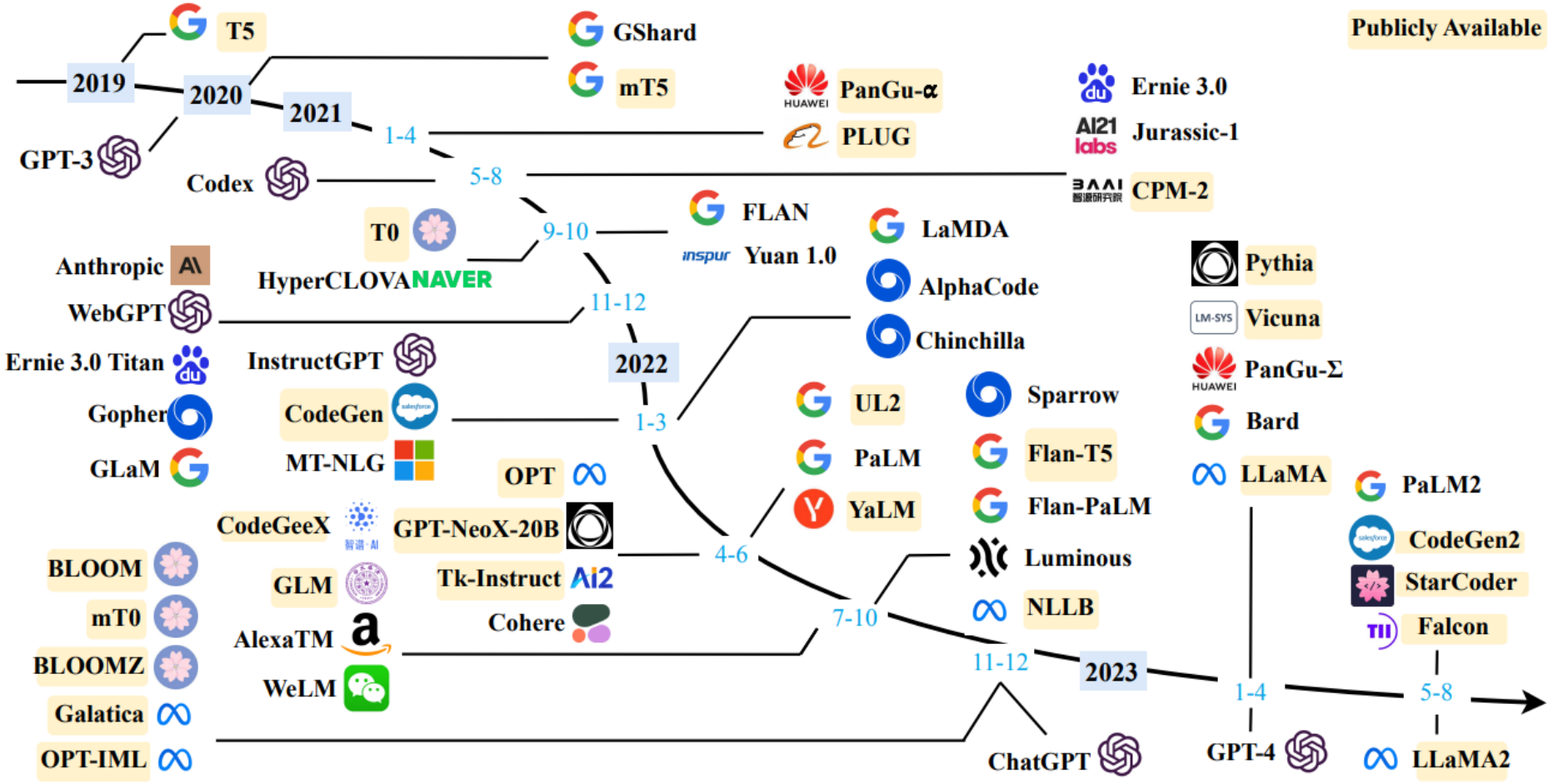
Translate the following text into Arabic and French:
"The poor and middle class work to money. The rich have money to work for them."

→ **GPT-3.5** →

Arabic translation: "الفقراء والطبقة الوسطى يعملون من أجل المال. الأغنياء لديهم المال ليعملوا عنهم."

French translation: "Les pauvres et la classe moyenne travaillent pour l'argent. Les riches ont de l'argent pour travailler à leur place."

Timeline of existing large language models (larger than 10B)



Gemini is a family of multimodal large language models, Comprising Gemini Ultra, Gemini Pro, and Gemini Nano,

December 6, 2023

Fig. 2: A timeline of existing large language models (having a size larger than 10B) in recent years. The timeline was established mainly according to the release date (*e.g.*, the submission date to arXiv) of the technical paper for a model. If there was not a corresponding paper, we set the date of a model as the earliest time of its public release or announcement. We mark the LLMs with publicly available model checkpoints in yellow color. Due to the space limit of the figure, we only include the LLMs with publicly reported evaluation results. (Wayne Xin Zhao et al., 2023)

Statistics of large language models (larger than 10B) (Wayne Xin Zhao et al., 2023)

TABLE 1: Statistics of large language models (having a size larger than 10B in this survey) in recent years, including the capacity evaluation, pre-training data scale (either in the number of tokens or storage size) and hardware resource costs. In this table, we only include LLMs with a public paper about the technical details. Here, "Release Time" indicates the date when the corresponding paper was officially released. "Publicly Available" means that the model checkpoints can be publicly accessible while "Closed Source" means the opposite. "Adaptation" indicates whether the model has been with subsequent fine-tuning: IT denotes instruction tuning and RLHF denotes reinforcement learning with human feedback. "Evaluation" indicates whether the model has been evaluated with corresponding abilities in their original paper: ICL denotes in-context learning and CoT denotes chain-of-thought. "*" denotes the largest publicly available version.

	Model	Release Time	Size (B)	Base Model	Adaptation IT	Adaptation RLHF	Pre-train Data Scale	Latest Data Timestamp	Hardware (GPUs / TPUs)	Training Time	Evaluation ICL	Evaluation CoT
Publicly Available	T5 [73]	Oct-2019	11	-	-	-	1T tokens	Apr-2019	1024 TPU v3	-	✓	-
	mT5 [74]	Oct-2020	13	-	-	-	1T tokens	-	-	-	✓	-
	PanGu-α [75]	Apr-2021	13*	-	-	-	1.1TB	-	2048 Ascend 910	-	✓	-
	CPM-2 [76]	Jun-2021	198	-	-	-	2.6TB	-	-	-	-	-
	T0 [28]	Oct-2021	11	T5	✓	-	-	-	512 TPU v3	27 h	✓	-
	CodeGen [77]	Mar-2022	16	-	-	-	577B tokens	-	-	-	✓	-
	GPT-NeoX-20B [78]	Apr-2022	20	-	-	-	825GB	-	96 40G A100	-	✓	-
	Tk-Instruct [79]	Apr-2022	11	T5	✓	-	-	-	256 TPU v3	4 h	✓	-
	UL2 [80]	May-2022	20	-	-	-	1T tokens	Apr-2019	512 TPU v4	-	✓	✓
	OPT [81]	May-2022	175	-	-	-	180B tokens	-	992 80G A100	-	✓	-
	NLLB [82]	Jul-2022	54.5	-	-	-	-	-	-	-	✓	-
	CodeGeeX [83]	Sep-2022	13	-	-	-	850B tokens	-	1536 Ascend 910	60 d	✓	-
	GLM [84]	Oct-2022	130	-	-	-	400B tokens	-	768 40G A100	60 d	✓	-
	Flan-T5 [64]	Oct-2022	11	T5	✓	-	-	-	-	-	✓	✓
	BLOOM [69]	Nov-2022	176	-	-	-	366B tokens	-	384 80G A100	105 d	✓	-
	mT0 [85]	Nov-2022	13	mT5	✓	-	-	-	-	-	✓	-
	Galactica [35]	Nov-2022	120	-	-	-	106B tokens	-	-	-	✓	✓
	BLOOMZ [85]	Nov-2022	176	BLOOM	✓	-	-	-	-	-	✓	-
	OPT-IML [86]	Dec-2022	175	OPT	✓	-	-	-	128 40G A100	-	✓	✓
	LLaMA [57]	Feb-2023	65	-	-	-	1.4T tokens	-	2048 80G A100	21 d	✓	-
	Pythia [87]	Apr-2023	12	-	-	-	300B tokens	-	256 40G A100	-	✓	-
	CodeGen2 [88]	May-2023	16	-	-	-	400B tokens	-	-	-	✓	-
	StarCoder [89]	May-2023	15.5	-	-	-	1T tokens	-	512 40G A100	-	✓	✓
	LLaMA2 [90]	Jul-2023	70	-	✓	✓	2T tokens	-	2000 80G A100	-	✓	-

Scaling Laws for LLMs:

- Chinchilla scaling law (Google DeepMind team, Hoffmann et al. 2022)
- OpenAI team (Kaplan et al., 2020)

Statistics of large language models (larger than 10B) (Wayne Xin Zhao et al., 2023)

	Model	Release Time	Size (B)	Base Model	Adaptation IT	Adaptation RLHF	Pre-train Data Scale	Latest Data Timestamp	Hardware (GPUs / TPUs)	Training Time	Evaluation ICL	Evaluation CoT
Closed Source	GPT-3 [55]	May-2020	175	-	-	-	300B tokens	-	-	-	✓	-
	GShard [91]	Jun-2020	600	-	-	-	1T tokens	-	2048 TPU v3	4 d	-	-
	Codex [92]	Jul-2021	12	GPT-3	-	-	100B tokens	May-2020	-	-	✓	-
	ERNIE 3.0 [93]	Jul-2021	10	-	-	-	375B tokens	-	384 V100	-	✓	-
	Jurassic-1 [94]	Aug-2021	178	-	-	-	300B tokens	-	800 GPU	-	✓	-
	HyperCLOVA [95]	Sep-2021	82	-	-	-	300B tokens	-	1024 A100	13.4 d	✓	-
	FLAN [62]	Sep-2021	137	LaMDA-PT	✓	-	-	-	128 TPU v3	60 h	✓	-
	Yuan 1.0 [96]	Oct-2021	245	-	-	-	180B tokens	-	2128 GPU	-	✓	-
	Anthropic [97]	Dec-2021	52	-	-	-	400B tokens	-	-	-	✓	-
	WebGPT [72]	Dec-2021	175	GPT-3	-	✓	-	-	-	-	✓	-
	Gopher [59]	Dec-2021	280	-	-	-	300B tokens	-	4096 TPU v3	920 h	✓	-
	ERNIE 3.0 Titan [98]	Dec-2021	260	-	-	-	-	-	-	-	✓	-
	GLaM [99]	Dec-2021	1200	-	-	-	280B tokens	-	1024 TPU v4	574 h	✓	-
	LaMDA [63]	Jan-2022	137	-	-	-	768B tokens	-	1024 TPU v3	57.7 d	-	-
	MT-NLG [100]	Jan-2022	530	-	-	-	270B tokens	-	4480 80G A100	-	✓	-
	AlphaCode [101]	Feb-2022	41	-	-	-	967B tokens	Jul-2021	-	-	-	-
	InstructGPT [61]	Mar-2022	175	GPT-3	✓	✓	-	-	-	-	✓	-
	Chinchilla [34]	Mar-2022	70	-	-	-	1.4T tokens	-	-	-	✓	-
	PaLM [56]	Apr-2022	540	-	-	-	780B tokens	-	6144 TPU v4	-	✓	✓
	AlexaTM [102]	Aug-2022	20	-	-	-	1.3T tokens	-	128 A100	120 d	✓	✓
	Sparrow [103]	Sep-2022	70	-	-	✓	-	-	64 TPU v3	-	✓	-
	WeLM [104]	Sep-2022	10	-	-	-	300B tokens	-	128 A100 40G	24 d	✓	-
	U-PaLM [105]	Oct-2022	540	PaLM	-	-	-	-	512 TPU v4	5 d	✓	✓
	Flan-PaLM [64]	Oct-2022	540	PaLM	✓	-	-	-	512 TPU v4	37 h	✓	✓
	Flan-U-PaLM [64]	Oct-2022	540	U-PaLM	✓	-	-	-	-	-	✓	✓
	GPT-4 [46]	Mar-2023	-	-	✓	✓	-	-	-	-	✓	✓
	PanGu-Σ [106]	Mar-2023	1085	PanGu-α	-	-	329B tokens	-	512 Ascend 910	100 d	✓	-
	PaLM2 [107]	May-2023	16	-	✓	-	100B tokens	-	-	-	✓	✓

2. Major development stages of Language Modeling Approach

- **Statistical language models:**
  - Based on statistical learning methods (Markov assumption)

- **Neural language models:**
  - Characterize the probability of word sequences by neural networks, e.g., recurrent neural networks (RNNs).
  - Word2vec (shallow NN)

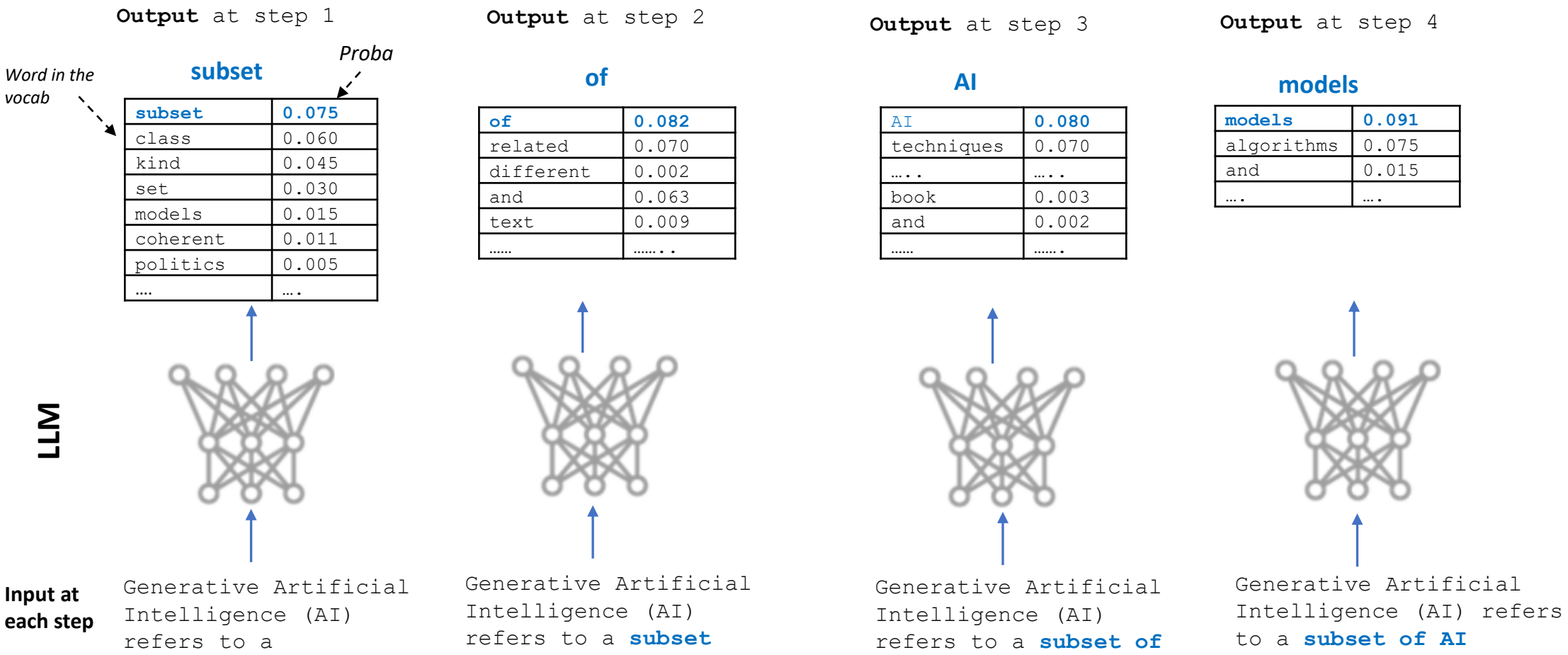
- **Pre-trained language models:**
  - ELMo (BiLSTM)
  - BERT (based on Transformer architecture)

- **Large language models* (term coined to large-sized pre-trained language models)**
  - Scaling pre-trained language models (scaling model/data sizes) often leads to an improved model capacity on downstream tasks (following the scaling law).

✓ LLMs show surprising abilities (called **emergent abilities**) in solving a series of complex tasks (typicaly, **in-context learning** (present in GPT-3 and not observed in small-scale language models (e.g., BERT, GPT-2), **instruction following**, and **step-by-step reasoning** (a.k.a. Chain-of-thought).

* Note that an LLM is not necessarily more capable than a small Pre-trained language models, and emergent abilities may not occur in some LLMs.

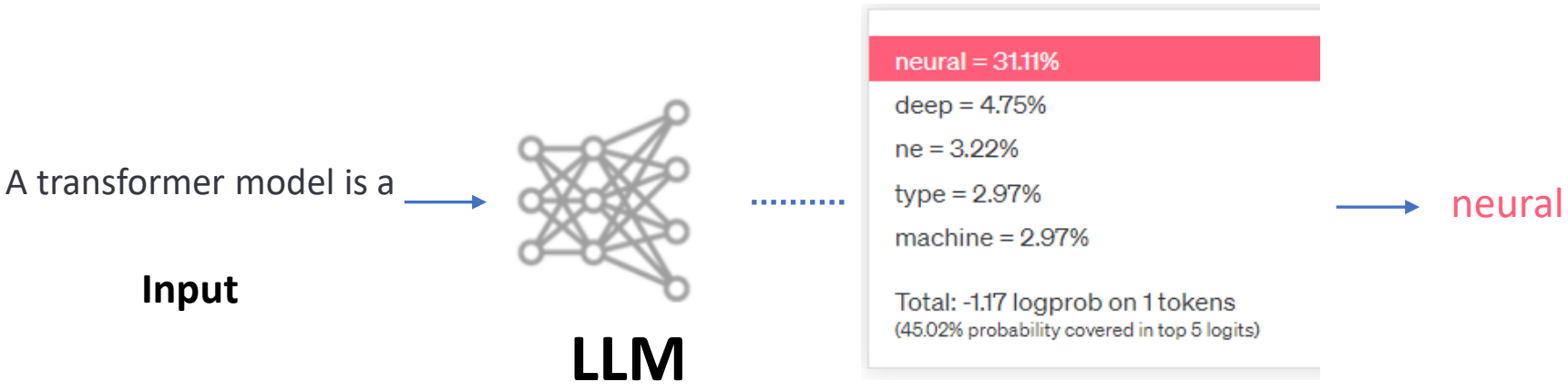
3. Text Generation:
General working flow of an LLM predicting the next word (autoregressive model)



General working flow of an LLM predicting the next word (e.g., select the token (word) with the highest probability) (auto-regressive models).

The LLM is focused on generating the next token given the sequence of tokens. The model does this in a loop appending the predicted token to the input sequence. Then, it can **generate text** by predicting **one word at a time**. LLMs are an example of Generative AI.

3. Text Generation



The model first generates ***logits*** for each **possible output token**. Those logits then are passed to a **softmax function** to generate **probabilities** for each possible output, giving a probability distribution over the **vocabulary**. Here is the softmax equation for calculating the actual probability of a token:

$$\mathrm{P}(\mathrm{token_k}\,|\mathrm{token\ context}) = \mathrm{softmax}(\mathrm{logit_k}) = \frac{\mathrm{e}^{\mathrm{logit_k}}}{\sum_{\mathrm{j}} \mathrm{e}^{\mathrm{logit_j}}}$$

Where:
- $P(token_k\,|token\ context)$ is the probability of $token_k$ given the context from previous tokens ($token_1$ to $token_{k-1}$)
- $logit_k$ is the output of the *neural network*

4. Decoding Strategy

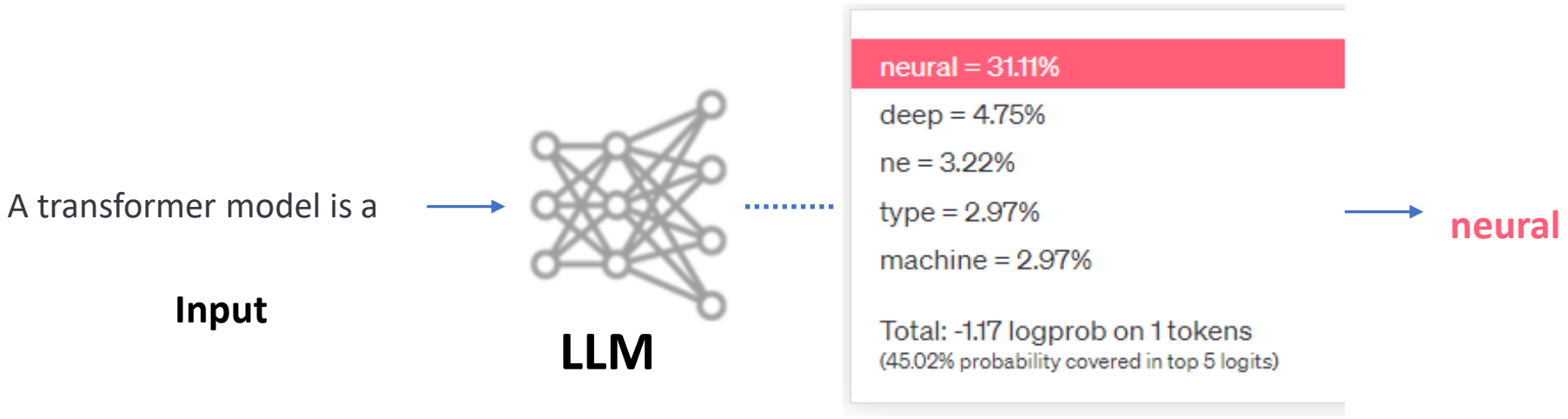
How to select the output tokens (a.k.a. **decoding strategy**)?

✓ **Greedy search:**

A basic decoding method that predicts the **most likely token at each step** based on the previously generated tokens, formally modeled as:

$$x_i = \arg\max_x P(x|\mathbf{x}_{<i})$$

where x_i is the token with with the highest probability at i-th step of generation conditioned on the context $\mathbf{x}_{<i}$.



4. Decoding Strategy

✓ **Random sampling (sampling based methods):**

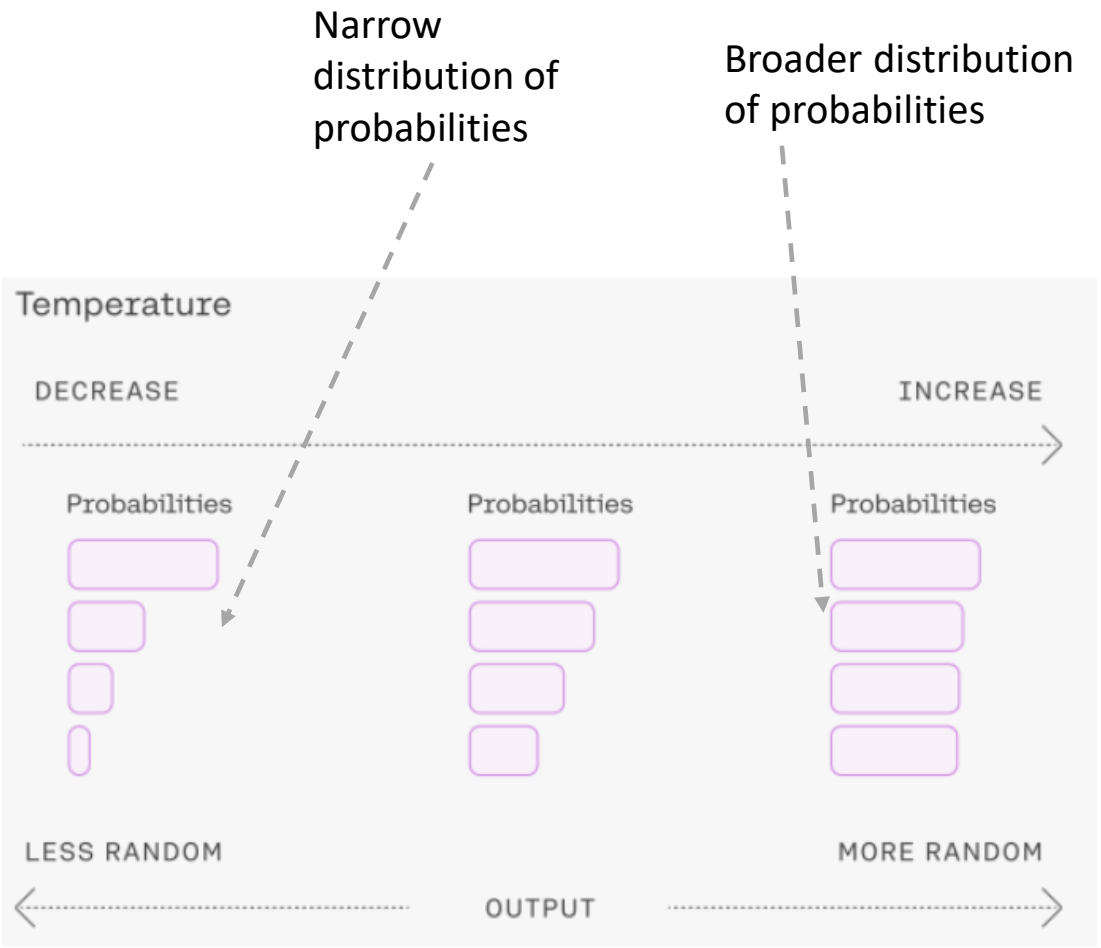
- **Temperature sampling:**

To modulate the randomness of sampling, a practical method is to adjust the temperature coefficient of the **softmax** function for computing the probability of the j-th token over the vocabulary:

$$P(x_j|\mathbf{x}_{<i}) = \frac{\exp(l_j/t)}{\sum_{j'} \exp(l_{j'}/t)}$$

where lj' is the logits of each word and t is the temperature coefficient.

Temperature is a parameter that you can access in LLMs which essentially guides the model on how random their behaviour is, which means that the temperature influences the model's creativity.



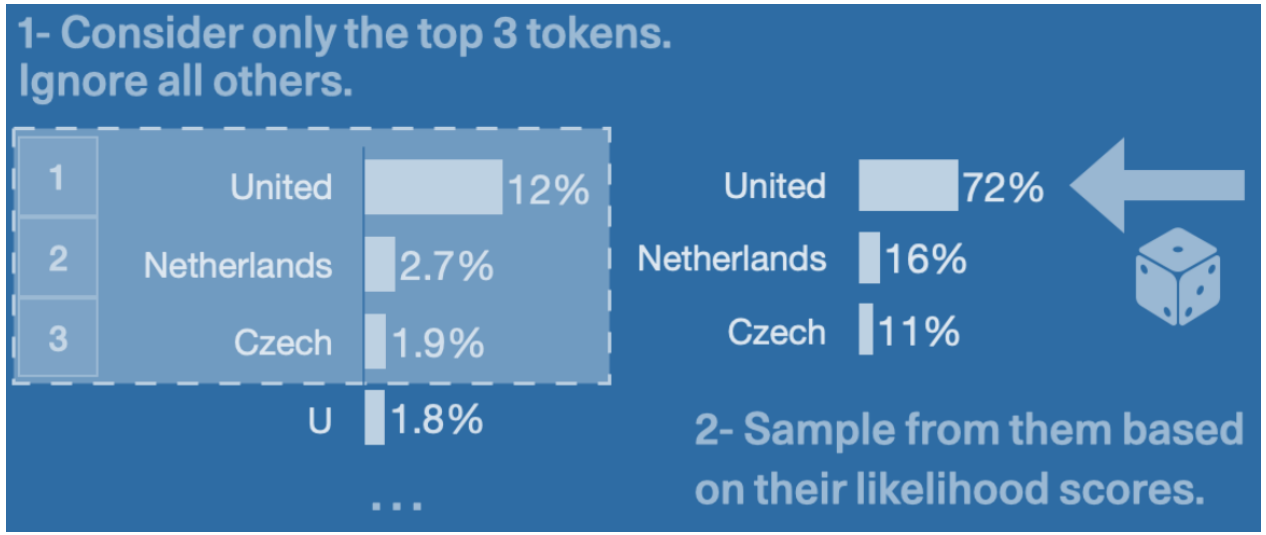
cohere.ai

4. Decoding Strategy

- **Top-k sampling:**

top-k sampling directly **truncates** the tokens **with lower probability** and **only samples from the tokens with the top k highest probabilities**.

K=3 =>



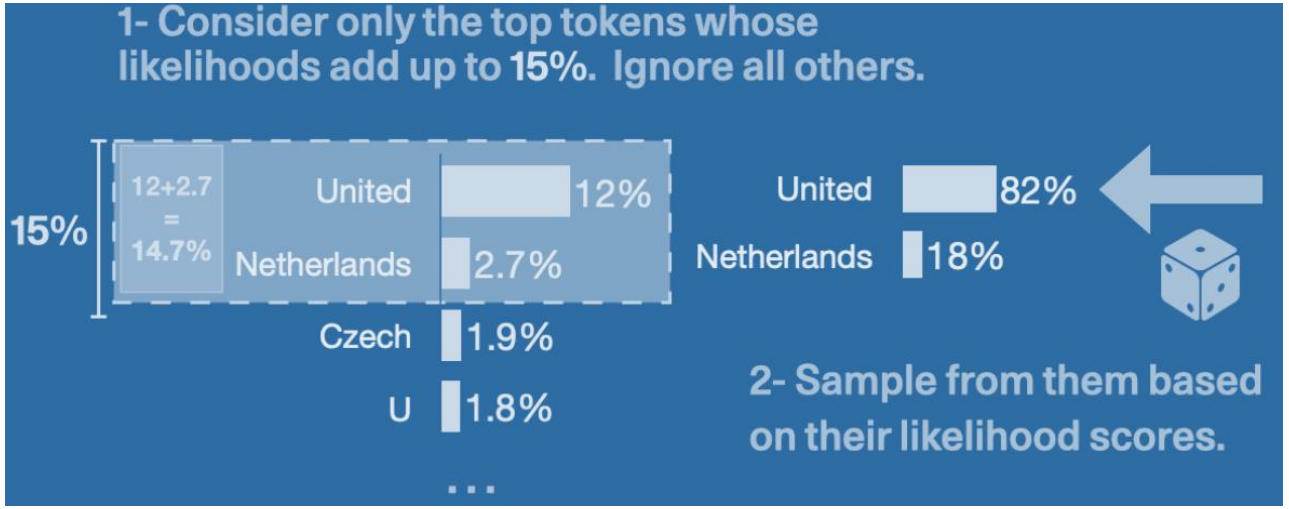
cohere.com

- top-k sampling does not consider the **overall possibility distribution**, a constant value of k may not be suitable for different contexts.

4. Decoding Strategy

- **Top-p sampling**

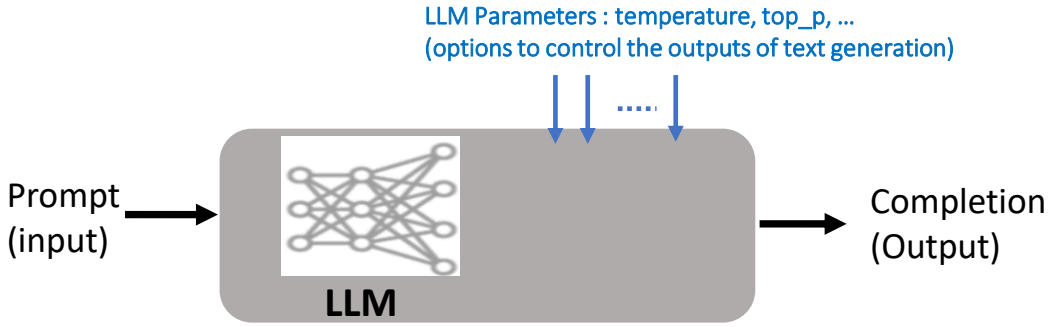
top-p sampling (a.k.a., **nucleus** sampling) is proposed by sampling from the **smallest set** having a cumulative probability above (or equal to) **p**.



cohere.com

Generally, it is recommended to alter **top-p** or **temperature** but not both.

4.1 Practical Settings (LLM Parameters: options to control the outputs of text generation)



- While the model decides what is the most probable output, there are key parameters to consider **tuning** (tweaking) to influence those probabilities for getting the best outputs of your LLM projects.

Category	**Example: OpenAI API parameter name** (try experimenting on the playground)
Let the model know when to stop: number of tokens, stop words	- max_tokens=1728 - stop=["."]
Predictability vs. creativity (decoding strategies): temperature, top_p	- temperature=0.19 - top_p=1
Control the repetition degree of generation: repetition penalty	- frequency_penalty=0.26 - presence_penalty=0.29 - logit_bias

POST /v1/chat/completions — python — Copy

```python
# This code is for v1 of the openai package: pypi.org/project/openai
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {
      "role": "user",
      "content": "A transformer model is a "
    }
  ],
  temperature=0.19,
  max_tokens=1728,
  top_p=1,
  frequency_penalty=0.26,
  presence_penalty=0.29,
  stop=["."]
)
```

OpenAI API: repetition penalty

✓ **Frequency and presence penalties:**
The frequency and presence penalties can be used to reduce the **likelihood of sampling repetitive sequences** of tokens. They work by directly modifying the logits (un-normalized log-probabilities) with an additive contribution (formula below from OpenAI documentation).

```
mu[j] -> mu[j] - c[j] * alpha_frequency - float(c[j] > 0) * alpha_presence
```

Where:

- `mu[j]` is the logits of the j-th token
- `c[j]` is how often that token was sampled prior to the current position
- `float(c[j] > 0)` is 1 if `c[j] > 0` and 0 otherwise
- `alpha_frequency` is the frequency penalty coefficient
- `alpha_presence` is the presence penalty coefficient

- The presence penalty is a **one-off additive contribution** that applies to all tokens that have been **sampled at least once** and the frequency penalty is a contribution that is **proportional** to how often a particular token has already been sampled.

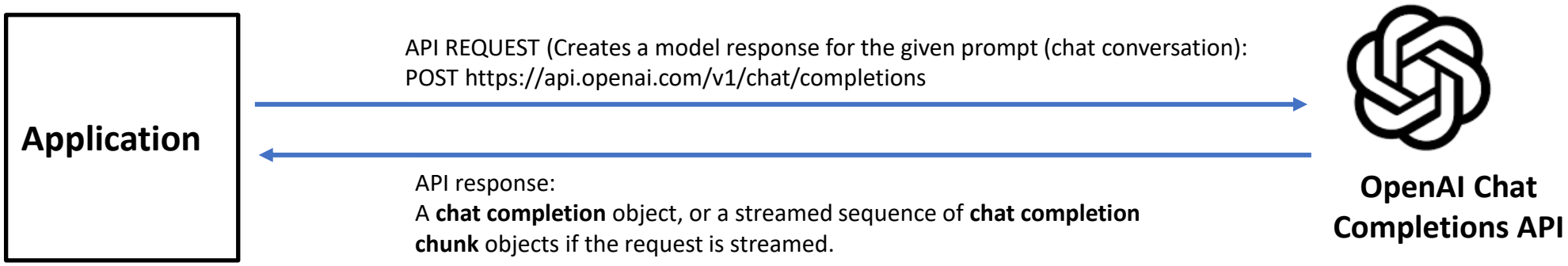
- Reasonable values for the penalty coefficients are around 0.1 to 1 if the aim is to just reduce repetitive samples somewhat. If the aim is to strongly suppress repetition, then one can increase the coefficients up to 2, but this can noticeably degrade the quality of samples. Negative values can be used to increase the likelihood of repetition.

5. OpenAI Text generation models

- OpenAI's **text generation models** (often called generative pre-trained transformers or large language models) have been trained to understand natural language, code, and images. The models provide text outputs in response to their inputs.

	MODEL FAMILIES
Newer models (2023–)	gpt-4, gpt-4 turbo, gpt-3.5-turbo
Updated legacy models (2023)	gpt-3.5-turbo-instruct, babbage-002, davinci-002
Legacy models (2020–2022)	text-davinci-003, text-davinci-002, davinci, curie, babbage, ada

5.1 OpenAI Chat Completions API endpoint



Typically, a conversation is formatted as follows:

- system message
- user messages
- assistant messages

```
# Example chat completions API request (OpenAI Python library)
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
      {"role": "system", "content": "……….."},
      {"role": "user", "content": "……."},
      {"role": "assistant", "content": "……..."},
      {"role": "user", "content": "……"}
  ],
  temperature=0, # Defaults to 1.
  max_tokens=256,
  top_p=1,
  frequency_penalty=0,
  presence_penalty=0
)
```

An example API call looks as follows: (try experimenting on the playground)

Chat Completions response format

Example of Chat Completions Request:

```
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {
      "role": "system",
      "content": "You are a helpful assistant. You
         will be provided with a general knowledge
         question and your task is to provide a
         concise answer."
    },
    {
      "role": "user",
      "content": "Who won the world series in 2020?"
    }
],
  temperature=1,
  max_tokens=256,
  top_p=1,
  frequency_penalty=0,
  presence_penalty=0
)
```

An example **Chat Completions API response** looks as follows:

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "The Los Angeles Dodgers won the
                    World Series in 2020. "
      }
    }
  ],
  "created": 1677664795,
  "id": "chatcmpl-7QyqpwdfhqwajicIEznoc6Q47XAyW",
  "model": "gpt-3.5-turbo-0613",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 13,
    "prompt_tokens": 28,
    "total_tokens": 41
  }
}
```

6. Pre-training LLMs (Unsupervised pre-training Task): Causal language modeling task (autoregressive models)

- Most LLMs are developed based on the **decoder-only architecture**, such as **GPT-3,** BLOOM, Gopher, and OPT.
- During the **pre-training stage**, LLMs are trained using **the language modeling objective on a large-scale corpus**.
- **Language Modeling task** (i.e., the conventional LM**)** is the **most commonly** used objective to pre-train decoder-only LLMs (e.g., GPT-3).

Given a sequence of tokens:

$$\mathbf{x} = \{x_1, \ldots, x_n\}$$

The LM task aims to autoregressively predict the target tokens x_i based on the **preceding** tokens $x_{<i}$ **in a sequence**. A general training objective is to maximize the following likelihood:

$$\mathcal{L}_{LM}(\mathbf{x}) = \sum_{i=1}^{n} \log P(x_i | \mathbf{x}_{<i})$$

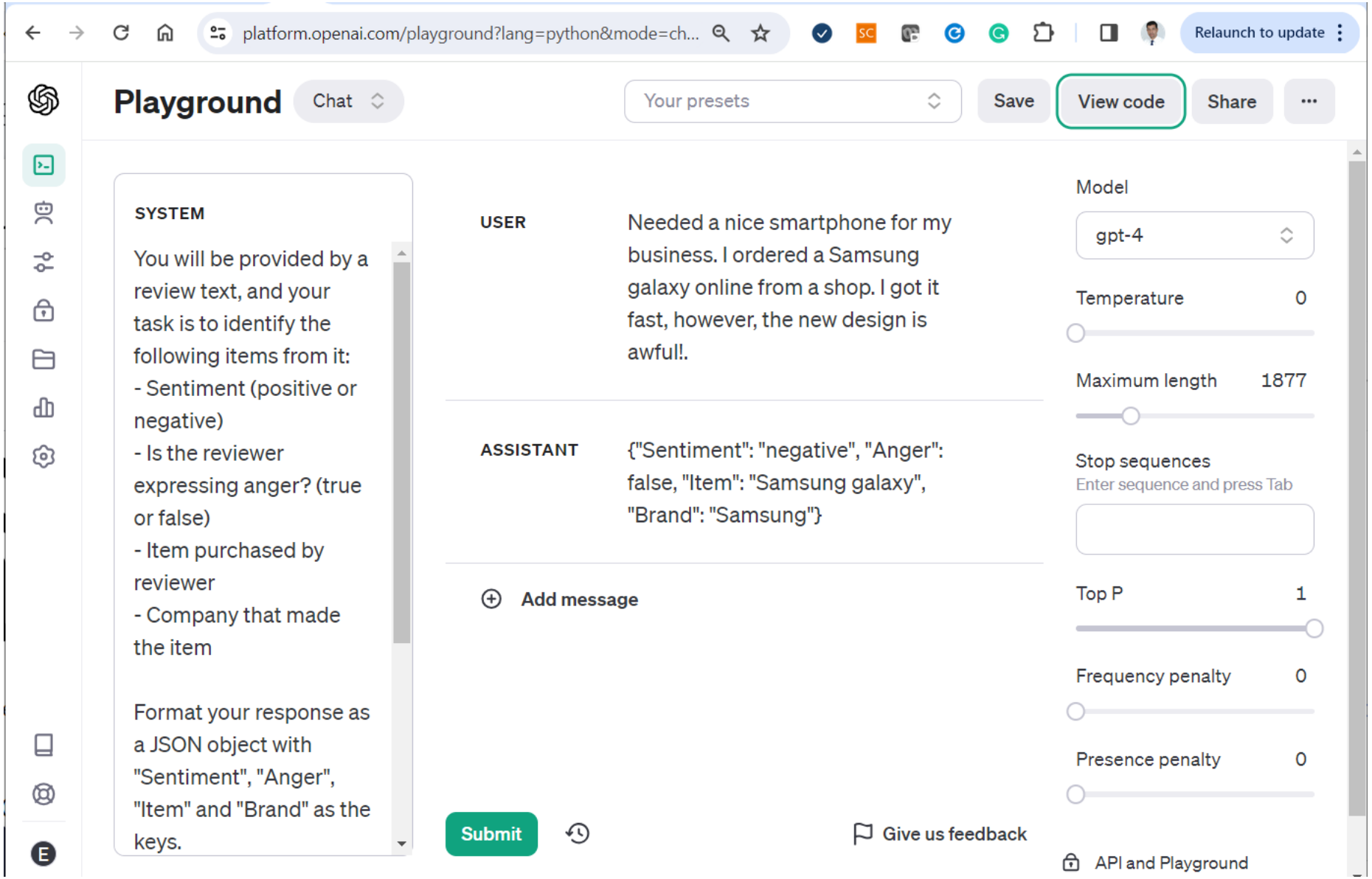
- ✓ A large language model, after pretraining (using a language modeling task), is **able to provide a global understanding of the language it is trained on**.
- ✓ Large language models can perform **hundreds of NLP tasks they were not trained for**.

LLMs can perform **hundreds of NLP tasks they were not trained for**.

Example: Brand Monitoring, keep a close eye on your brand's reputation

In this example, the LLM performs 3 tasks at once:

- ✓ Sentiment Analysis
- ✓ Emotion recognition
- ✓ Entity extraction



Ratios of various data sources in the pre-training data for existing LLMs

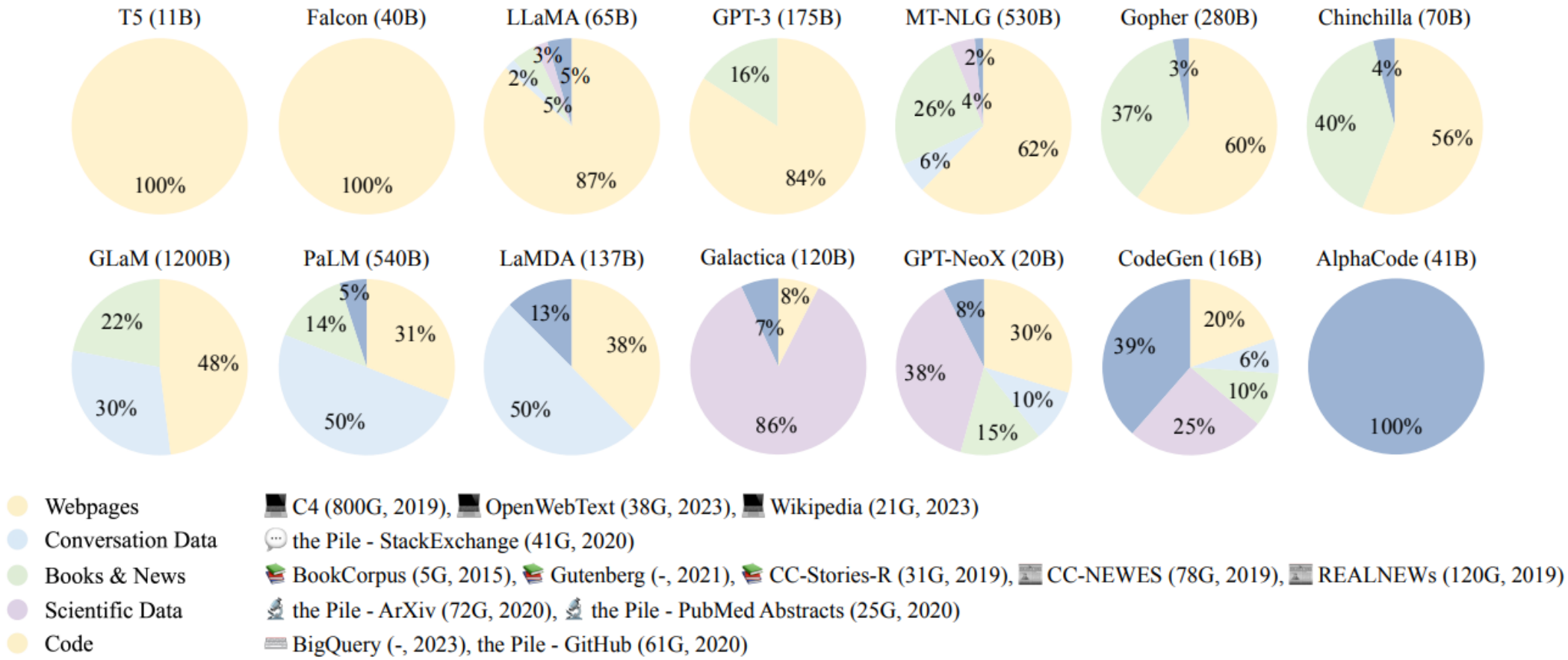
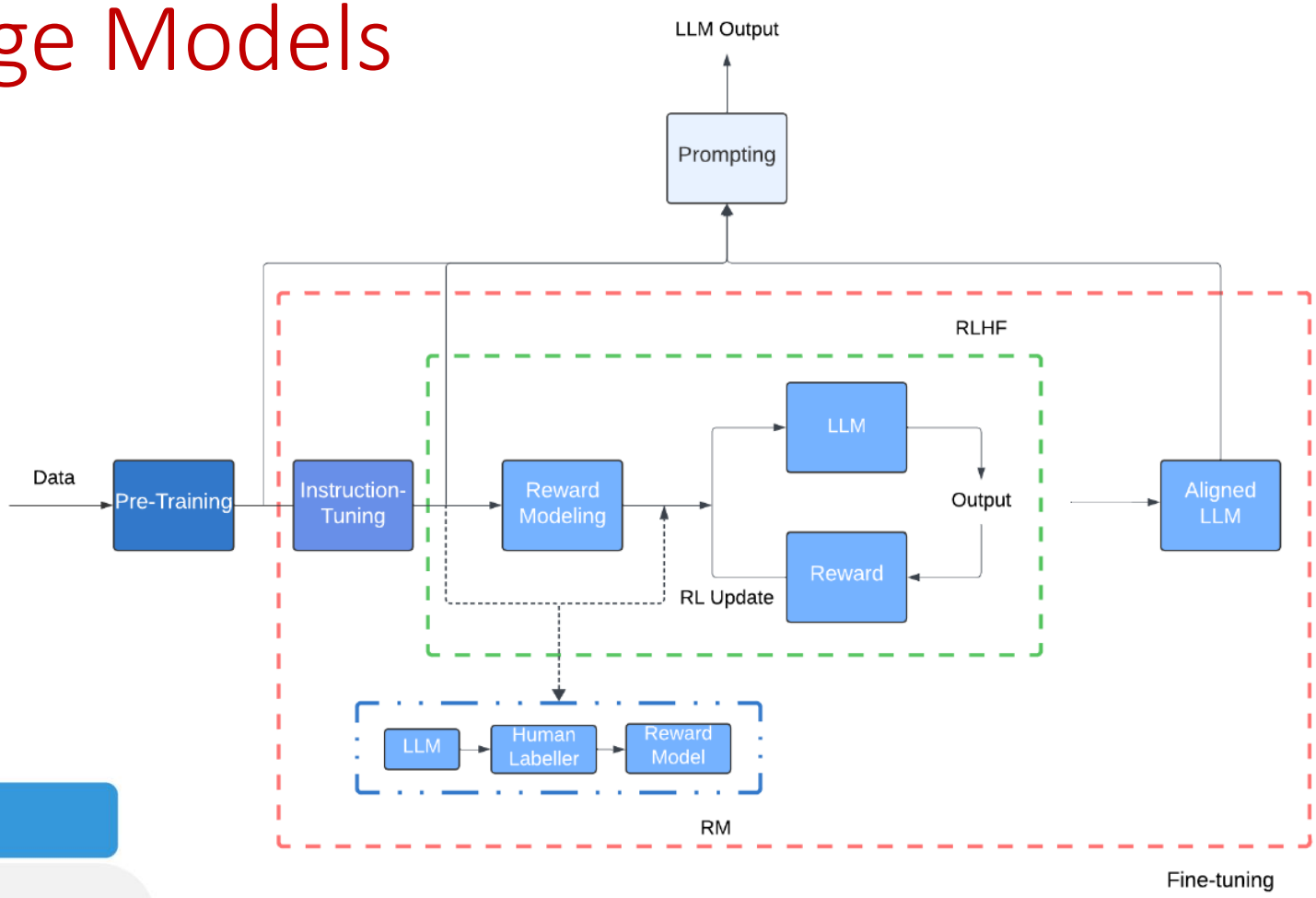


Fig. 5: Ratios of various data sources in the pre-training data for existing LLMs. (Wayne Xin Zhao et al., 2023)

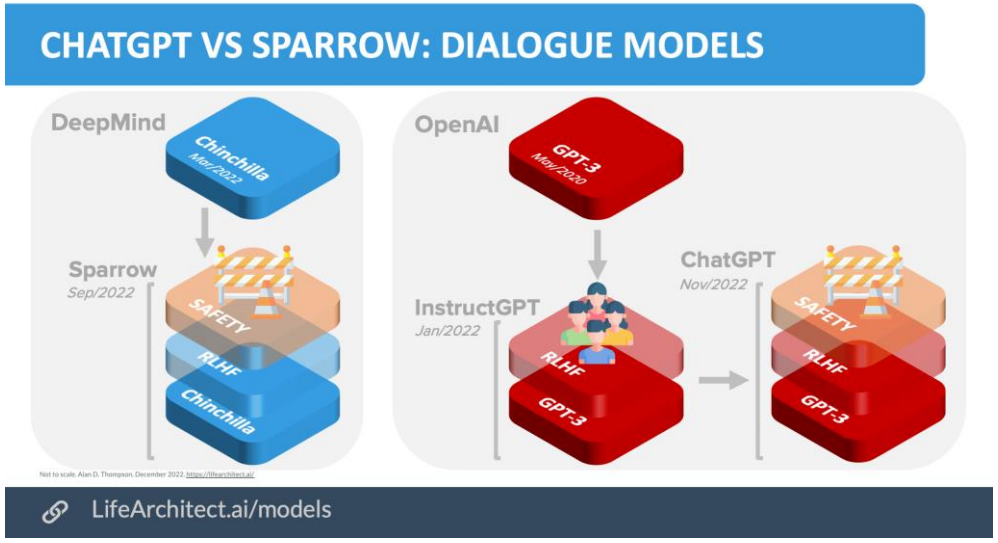
7. Types of Large Language Models

There are three types of LLMs:

- **Base LLMs** (pre-trained model checkpoints): Model checkpoints obtained right after pre-training.
  - ✓ A base LLM acquires the **general abilities for solving various tasks**.
  - ✓ It lacks the consideration of **human values or preferences**.
- **Fine-tuned LLMs** (**adapting pre-trained LLMs**: **instruction** or **alignment** fine-tuned model checkpoints, - also called chat models)
  - Example: representative alignment criteria (i.e., helpful, honest, and harmless)
- **Specialized LLMs** (adapted model checkpoints for some specific task or domain: Healthcare, Finance, Legal, etc.).

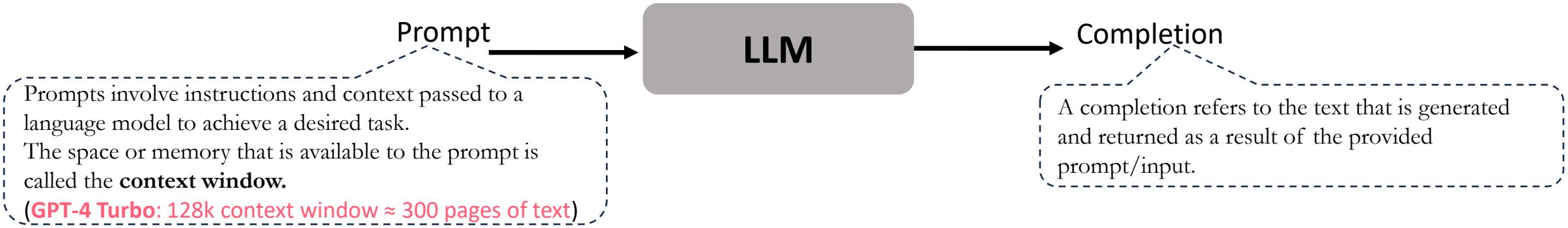


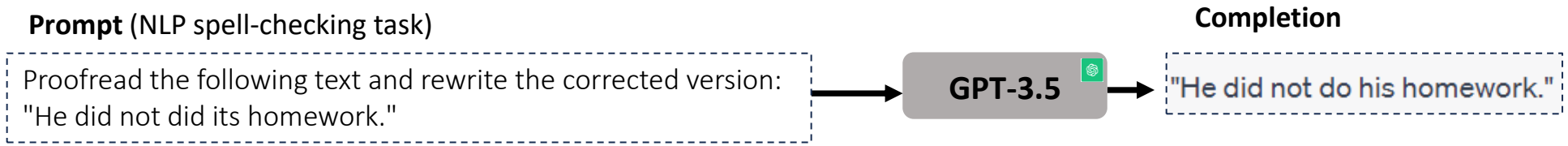
A basic flow diagram depicting various stages of LLMs from pre-training to prompting/utilization. Prompting LLMs to generate responses is possible at different training stages like pre-training, instruction-tuning, or alignment tuning. (H. Naveed et al., 2023)

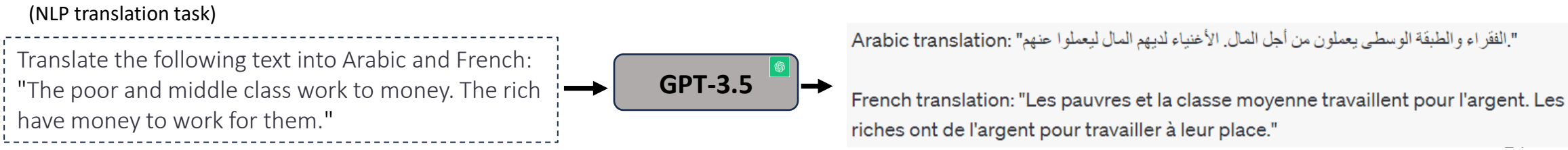


8. LLMs utilization:
How to work with large language models?

8.1. Prompt, completion, and prompt engineering







Prompt engineering:

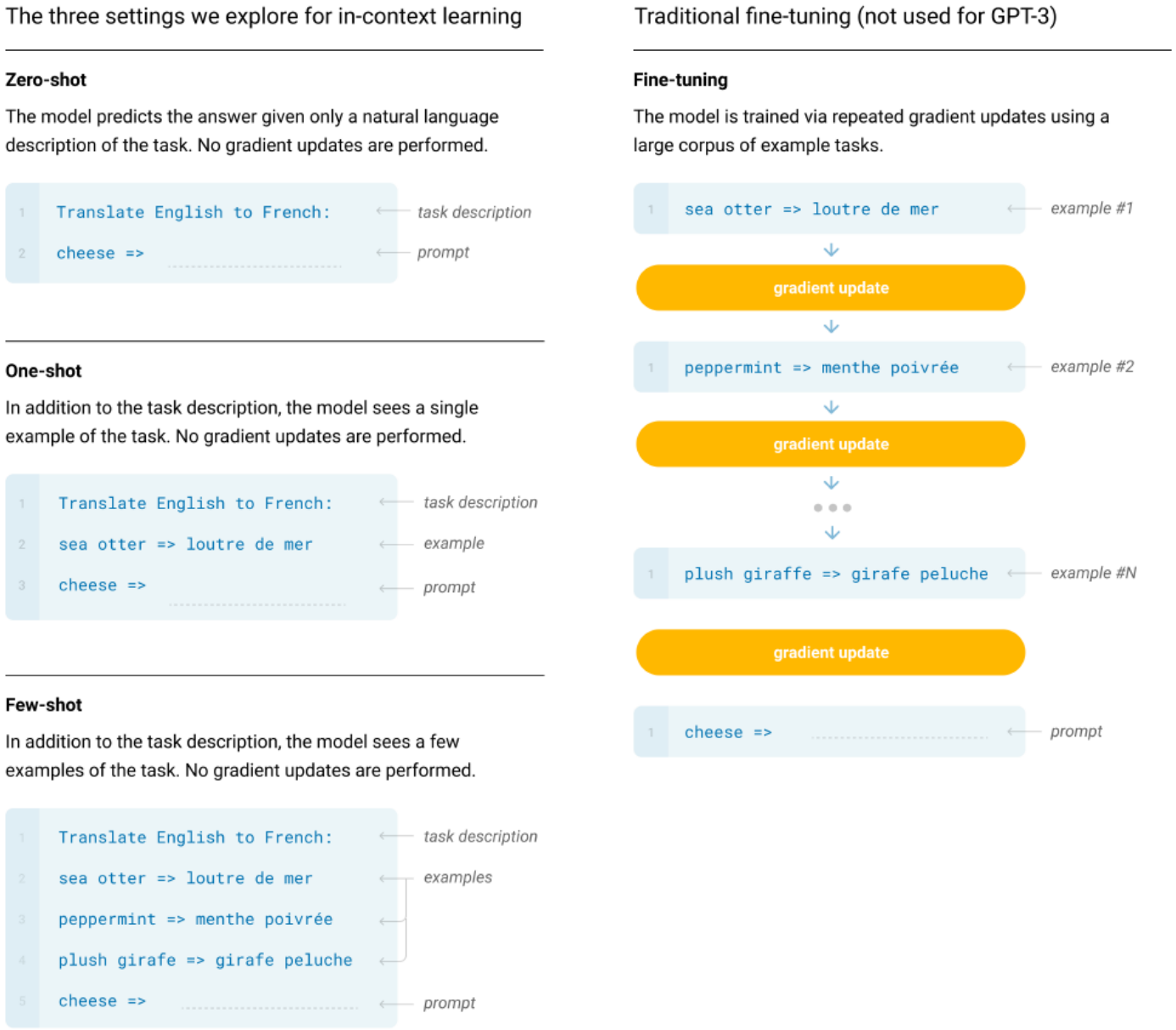
Prompt engineering is a useful skill for AI engineers and researchers to improve and efficiently use language models.
OpenAI Guide: https://platform.openai.com/docs/guides/prompt-engineering

8.2. Different kinds of prompts

- ✓ Large language models can be prompted to produce output in a few ways :
  - Zero-shot prompt
  - One-shot prompt
  - Few-shot prompt
  - CoT prompt
  - Planning
- ✓ Fine-tuning LLM for a specific task (supervised fine-tuning)

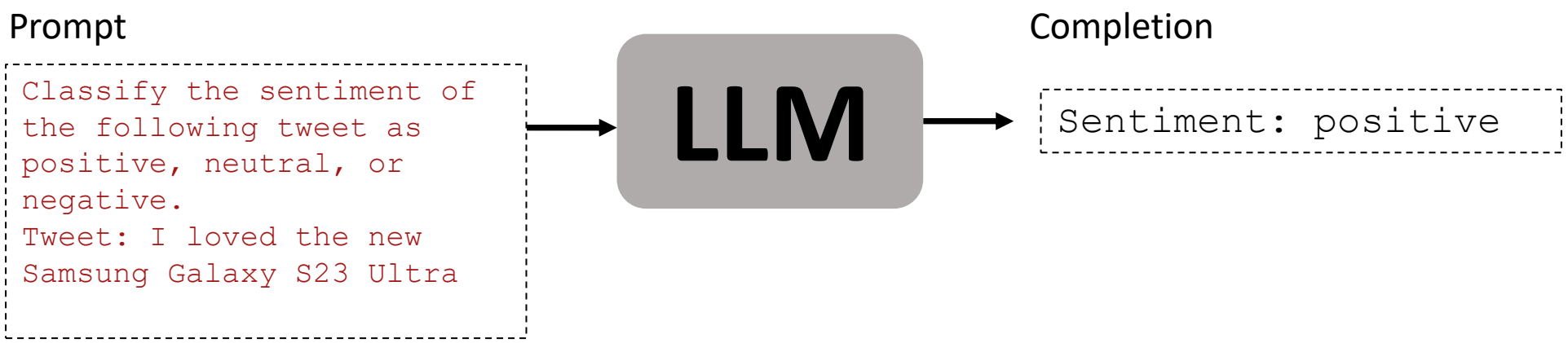
In-context learning (Zero-shot prompt, One-shot prompt, Few-shot prompt):
A typical prompting method which formulates the task description and/or demonstrations in the form of natural language text.

LLMs are capable of learning from the **examples concatenated with the input**, known as context augmentation, in context learning (ICL), or few-shot prompting. They show excellent generalization to unseen tasks with few-shot prompting, enabling LLMs to answer queries beyond the capacity acquired during training (T. Brown et al., 2020)



Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning (Brown et al., 2020: GPT-3 original paper)

Zero-shot prompt



Zero-shot prompt

- Example: Zero-shot prompt given to GPT-3.5-turbo

Prompt

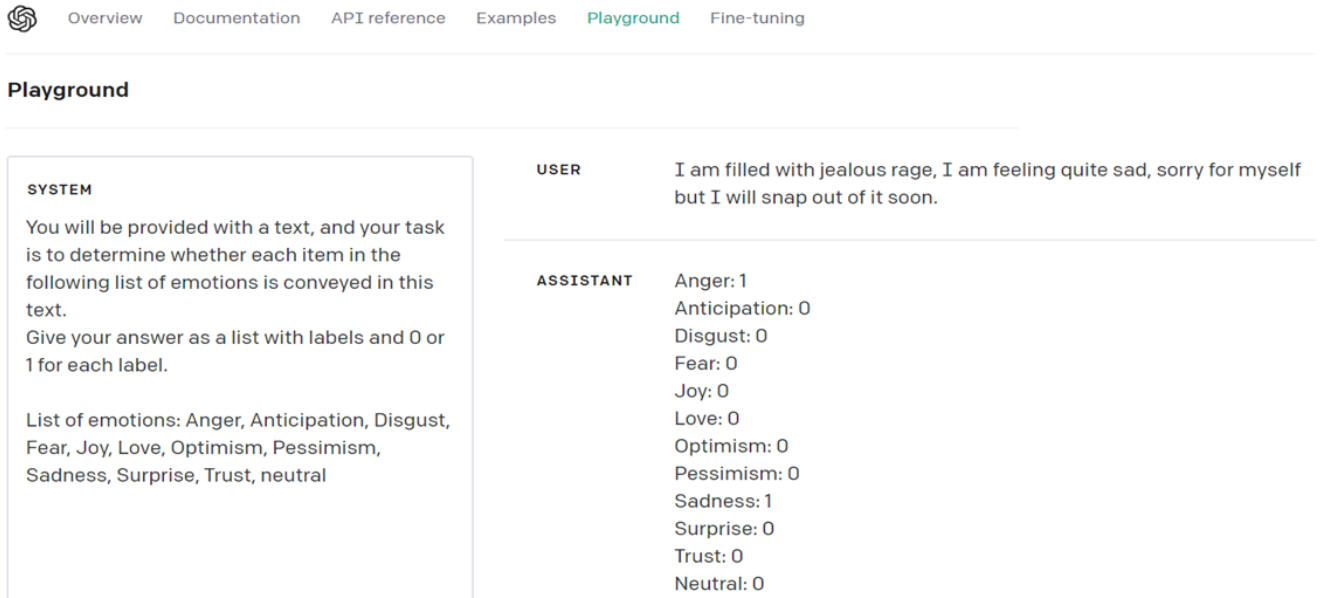
```
Determine whether each item in the following list
of emotions is conveyed in the text below, which
is delimited with triple backticks.
Give your answer as a list with labels and 0 or 1
for each label.
List of emotions: Anger, Anticipation, Disgust,
Fear, Joy, Love, Optimism, Pessimism, Sadness,
Surprise, Trust, neutral
Text : ``` I am filled with jealous rage, I am
feeling quite sad, sorry for myself but I will
snap out of it soon.```
```

GPT-3.5-turbo

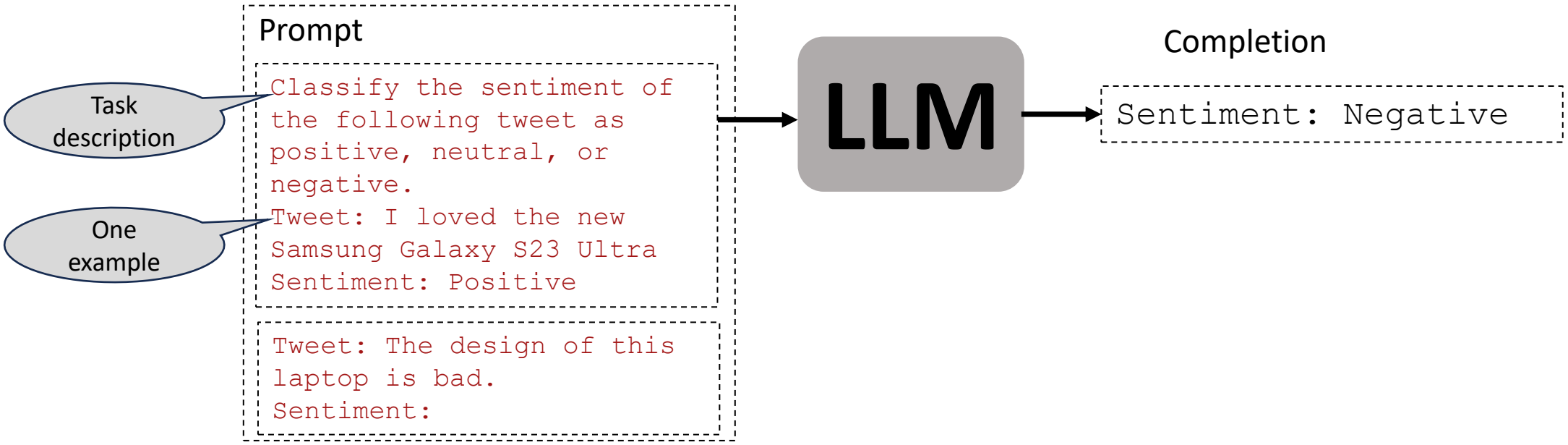
Completion

Anger: 1
Anticipation: 0
Disgust: 0
Fear: 0
Joy: 0
Love: 0
Optimism: 0
Pessimism: 0
Sadness: 1
Surprise: 0
Trust: 0
Neutral: 0

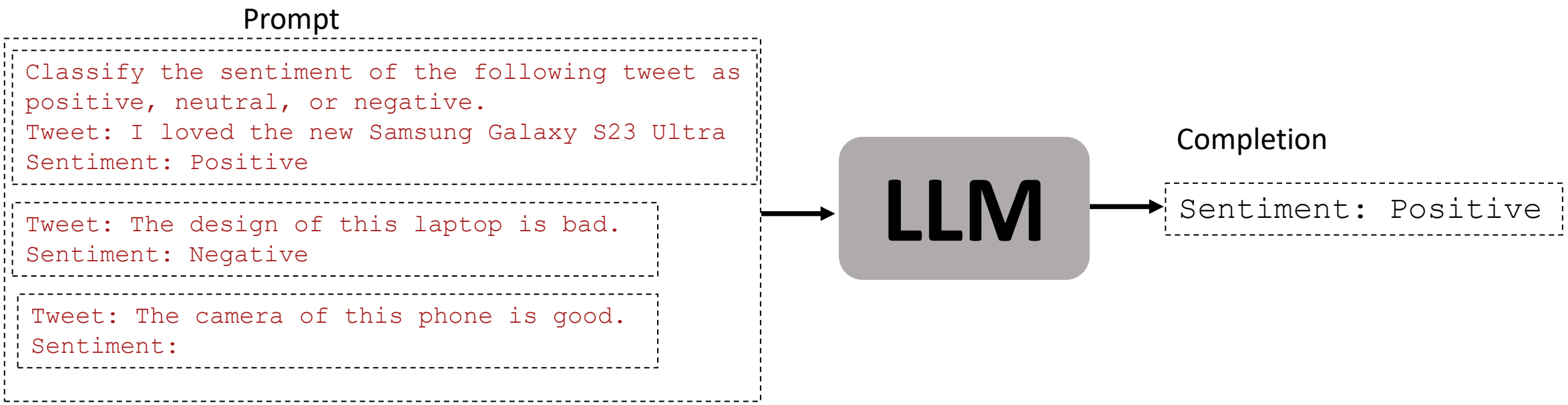
- OpenAI playground (GPT-3.5-turbo)



One-shot prompt

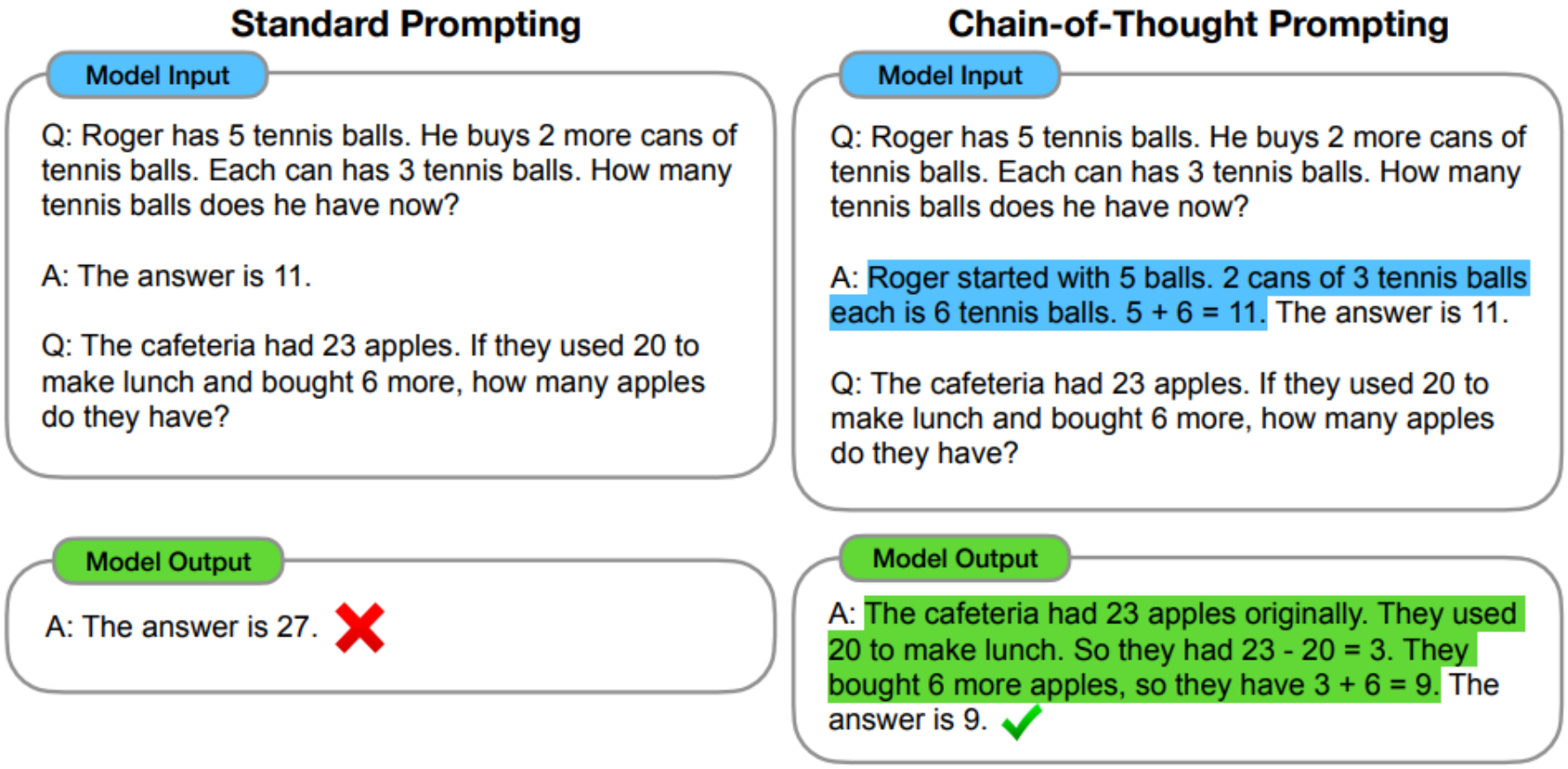


Few-shot prompt



- Typically, 10 to 100 shots for GPT-3 (Brown et al., 2020: GPT-3 original paper)

Chain of thought (CoT) prompting



Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted. (Wei, Jason et al., 2023)

- ✓ The core concept behind CoT is that by presenting the LLM **few-shot examples that include reasoning**, it will subsequently incorporate the reasoning process into its responses when addressing prompts.

Chain of thought (CoT) prompting: Zero-shot CoT

Examples with reasoning in the prompt = adding "**Let's consider step by step**" across the task.

(a) Few-shot

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: 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:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

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: 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:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4. ✓

(c) Zero-shot

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: The answer (arabic numerals) is

(Output) 8 ✗

(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. ✓

(Takeshi Kojima et al., 2023)

Fine-tuning LLM for a specific task (supervised fine-tuning)

- Fine-tuning LLMs can make them better for specific applications.
- Fine-tuning is suitable for continuous domain adaptation, enabling significant improvements in model quality but often incurring **higher costs**.
- Fine-tuning involves **updating the weights of a pre-trained model** by training on a **supervised dataset** specific to the desired task.
- Once a model has been fine-tuned, you won't need to provide as many examples in the prompt (few-shot).

8.3 OpenAI Fine-tuning API endpoint

Fine-tuning workflow using OpenAI Fine-tuning endpoint :

- Prepare training and validation data (formatted as a JSONL) document).
- Upload training and validation data files (use **Files endpoint**)
- Fine-tune the selected model.
- Analyse the results.

Once a model finishes the fine-tuning process, it is available to be used in production right away.

```python
from openai import OpenAI
client = OpenAI()

client.fine_tuning.jobs.create(
  training_file="file-abc123",
  model="gpt-3.5-turbo",
  hyperparameters={
    "n_epochs":2
  }
)
```

Code to start a fine-tuning job using the OpenAI SDK

9. LLM use cases, tasks, real-world applications

Large language models have numerous applications in various fields, including but not limited to:

- **Language translation:** LLMs can be used to translate text from one language to another.
- **Question answering:** LLMs can be used to answer questions based on a given context.
- **Text summarization:** Large language models can be used to generate summaries of text documents.
- **Content creation (generation):** LLMs can be used to generate content for various purposes, such as marketing and advertising.
- **Code generation**
- **Sentiment analysis:** LLMs can be used to analyze the sentiment of text
- **Chatbots**
- **Summarization, Essay writing**
- **Entity extraction**
- Etc.,

✓ None of these capabilities are explicitly programmed in—they all **emerge** as a **result of training using language modeling task**.

✓ A large language model, after pretraining (using a language modeling task), is **able to provide a global understanding of the language it is trained on**.

✓ Large language models can perform **hundreds of NLP tasks they were not trained for**.

10. Augmented LLMs and LLM-powered applications

✓ Some of the limitations of large language models are:
- The internal knowledge held by a model cuts off at the moment of pretraining,
- Hallucination
- Struggling with complex math
- Previous work has shown that LLMs can solve simple reasoning problems but fail at complex reasoning (Creswell et al., 2022)- (e.g., performing complex arithmetic)

MODEL	TRAINING DATA
gpt-4-1106-preview	Up to Apr 2023
gpt-3.5-turbo-1106	Up to Sep 2021

✓ Integration of LLMs into businesses:

LLMs are pre-trained on huge amounts of publicly available data like *Wikipedia, mailing lists, textbooks, source code and more*. However, they are not trained on **your** data, which may be private or specific to the problem you're trying to solve. It's in **SQL databases**, trapped in **PDFs** and slide decks, or behind APIs, etc.

=> How can companies use the **reasoning capabilities of LLMs** to generate responses based on **their private,** or **domain-specific, and new data**?

Augmented LLMs overcome the aforementioned issues and beyond.

10.1 Augmented LLMs

There are different types of augmented LLMs:

1. **Retrieval Augmented LLMs**

   Retrieving relevant information from external up-to-date storage enables the LLMs to accurately answer with references and utilize more information.

   a. **Zero-Shot Retrieval Augmentation**

      This kind of augmentation keeps the original LLM architecture and weights unchanged and uses a retriever.

   b. **Training with Retrieval Augmentation**:

      Train or fine-tune retrievers and LLMs with a retrieval augmentation pipeline.

2. **Tool Augmented LLMs**

   Tool-augmented LLMs **capitalize on the reasoning abilities** of LLMs to iteratively plan by dividing tasks into sub-tasks, selecting necessary tools, and taking actions to complete the task.

   Sequence: Plan → Tool selection → Execute→ Inspect → Generate, to respond to the user query.

   a. **Zero-Shot Tool Augmentation:**

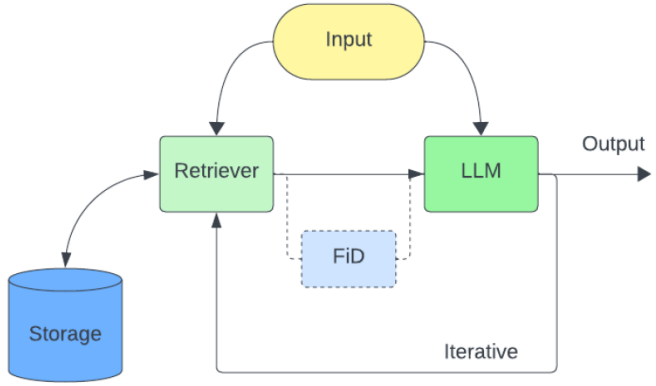
      LLMs in-context learning and reasoning abilities enable **them to interact with tools without training**.

   b. **Training with Tool Augmentation:**

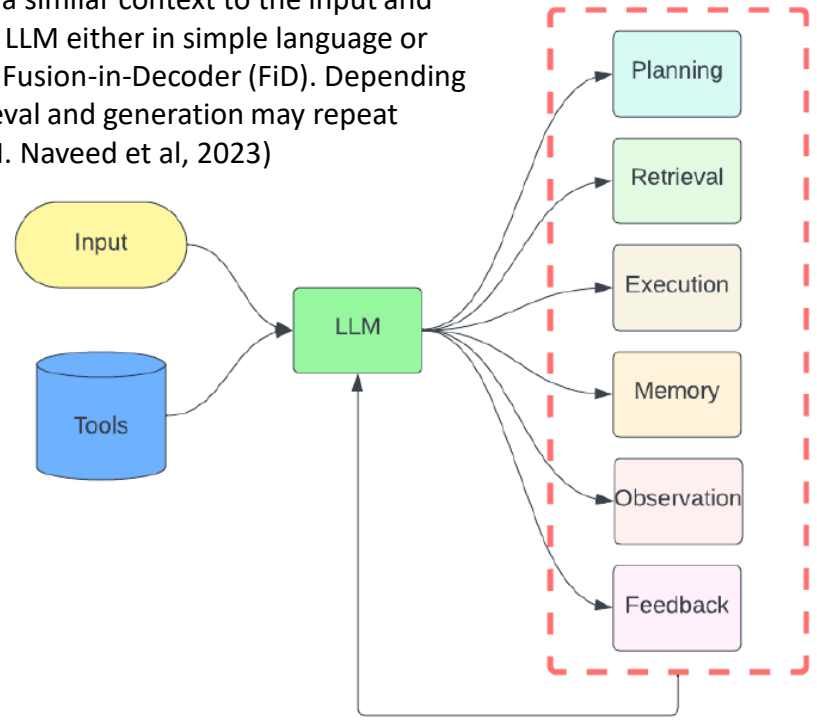
      LLMs are trained to interact with diverse tools, enhancing planning abilities to overcome the limitations of zero-shot tool augmentation.

   c. **Multimodal Tool Augmentation:**

      Here, the database of tools is rich in modalities, including text, images, etc.



(a) A flow diagram of **Retrieval Augmented LLMs**. The retriever extracts a similar context to the input and forwards it to the LLM either in simple language or encoded through Fusion-in-Decoder (FiD). Depending on the task, retrieval and generation may repeat multiple times. (H. Naveed et al, 2023)



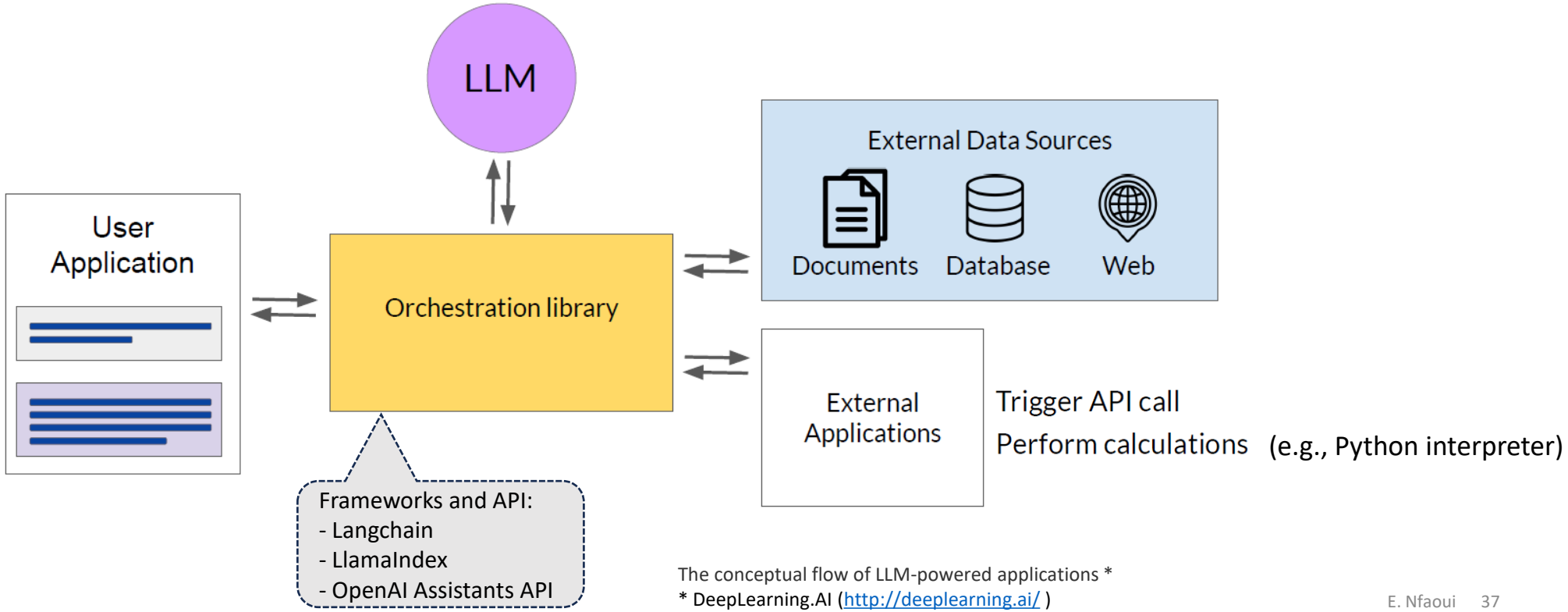
(b) A basic flow diagram of **tool-augmented LLMs**. Given an input and a set of available tools, the model generates a plan to complete the task. The tool-augmented LLMs utilize different modules iteratively, such as retriever, tool execution, read-write to memory, feedback, etc., depending on the task. (H. Naveed et al, 2023)

10.2 LLM-powered applications

There are techniques that help LLMs overcome the aforementioned issues by connecting to external data sources and applications.

- The **orchestration library** enables some powerful technologies that **augment** and enhance the performance of the LLM at **runtime** by providing access **to external data sources** or **connecting to existing APIs of other applications**. Implementation examples are **Langchain**, **LlamaIndex**, and **OpenAI Assistants API endpoint**.

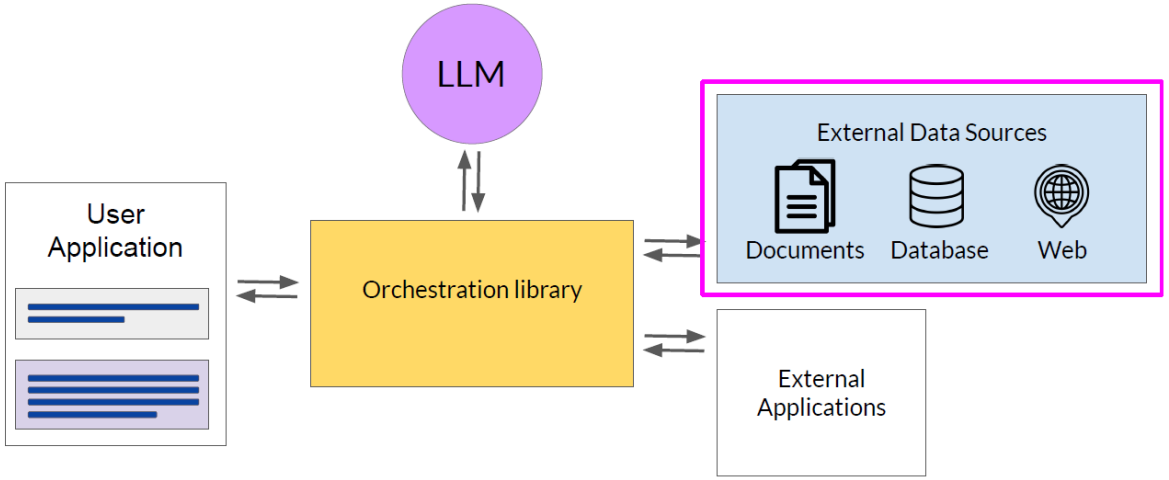
The orchestrator is a technical component that can manage the flow of information and the initiation of calls to external data sources or applications. It can also decide what actions to take based on the information contained in the output of the LLM.
- LLM is the application's reasoning engine. Ultimately, it creates the plan that the orchestrator will interpret and execute.



The conceptual flow of LLM-powered applications *
* DeepLearning.AI (http://deeplearning.ai/)

10.3 Retrieval Augmented Generation (RAG)

- **Retrieval Augmented Generation (RAG)** is a framework/pattern for building LLM-powered systems that make use of external data sources and applications to overcome some of the limitations of LLMs. Providing your model with external information can improve both the **relevance and accuracy** of its **completions**.
- RAG allows an LLM to **augment its knowledge at runtime** by retrieving relevant information from external data sources.
- RAG uses **prompt engineering** to supplement or guide the model **at inference time**.
- RAG allows the use of the same model as a reasoning engine over **new data provided in a prompt**. This technique enables **in-context learning** without the need for expensive fine-tuning, empowering businesses to use LLMs more efficiently.

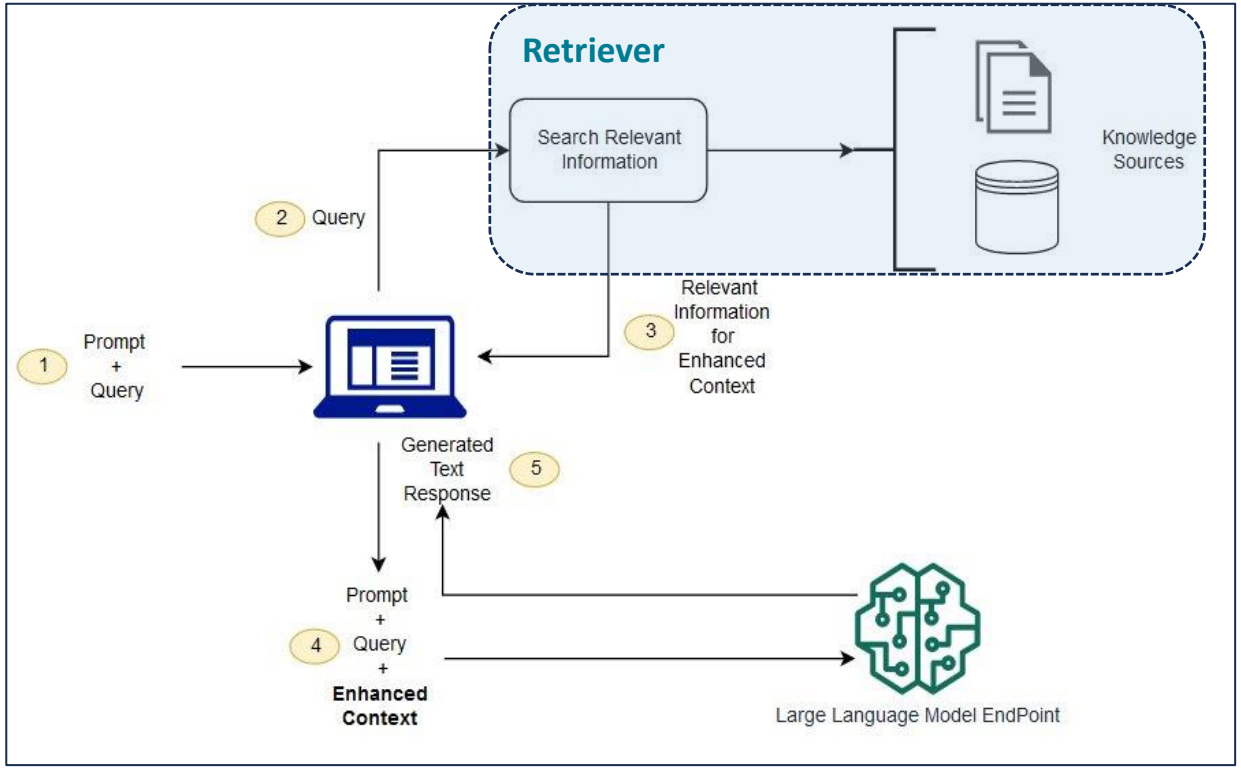


DeepLearning.AI (http://deeplearning.ai/)

10.3 Retrieval Augmented Generation (RAG)

How to connect LLMs to external data sources?

- There are **different implementations of RAG**: one of the earliest implementations was proposed in (Lewis et al., 2020)



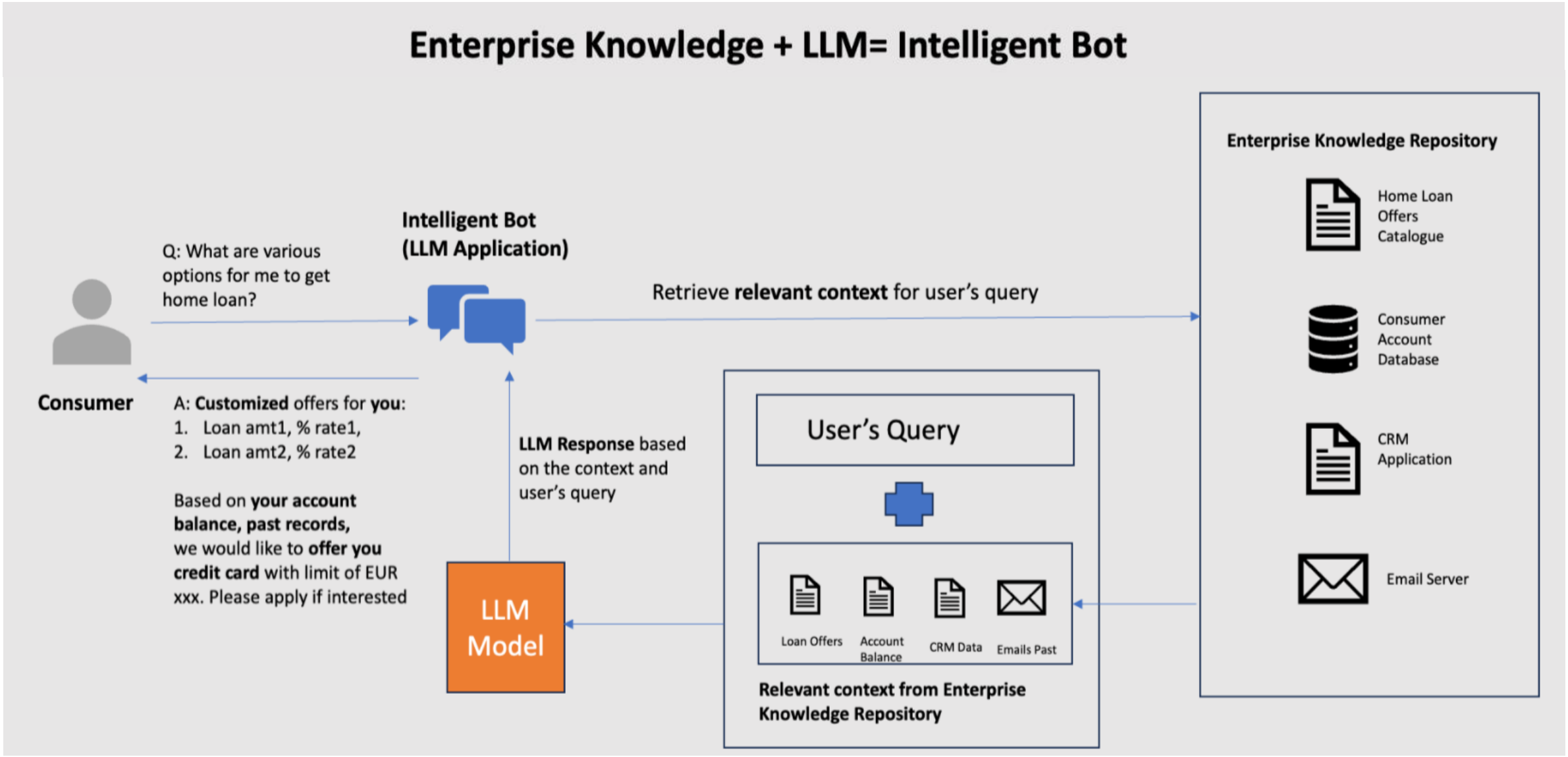
The conceptual flow of using RAG with LLMs (adapted from *) - (This flow corresponds to Zero-Shot Retrieval Augmentation which is a type of Retrieval Augmented LLMs)

- Retriever: encoders the query and searches for a relevant entry in the corpus of documents. The encoder takes the user's input prompt and encodes it into a form that can be used to query the data source. In the (Lewis et al., 2020) paper, the external data is a vector store.

- Step 4: the RAG model **augments** the user input (or prompts) by adding the relevant retrieved data in context.

RAG architectures can be used to integrate **multiple types of external information sources**. You can augment LLMs with access to local documents (including private wikis and expert systems), to the Internet to extract information posted on web pages. By **encoding the user input prompt as an SQL query**, RAG can also interact with databases.

* https://aws.amazon.com/what-is/retrieval-augmented-generation/

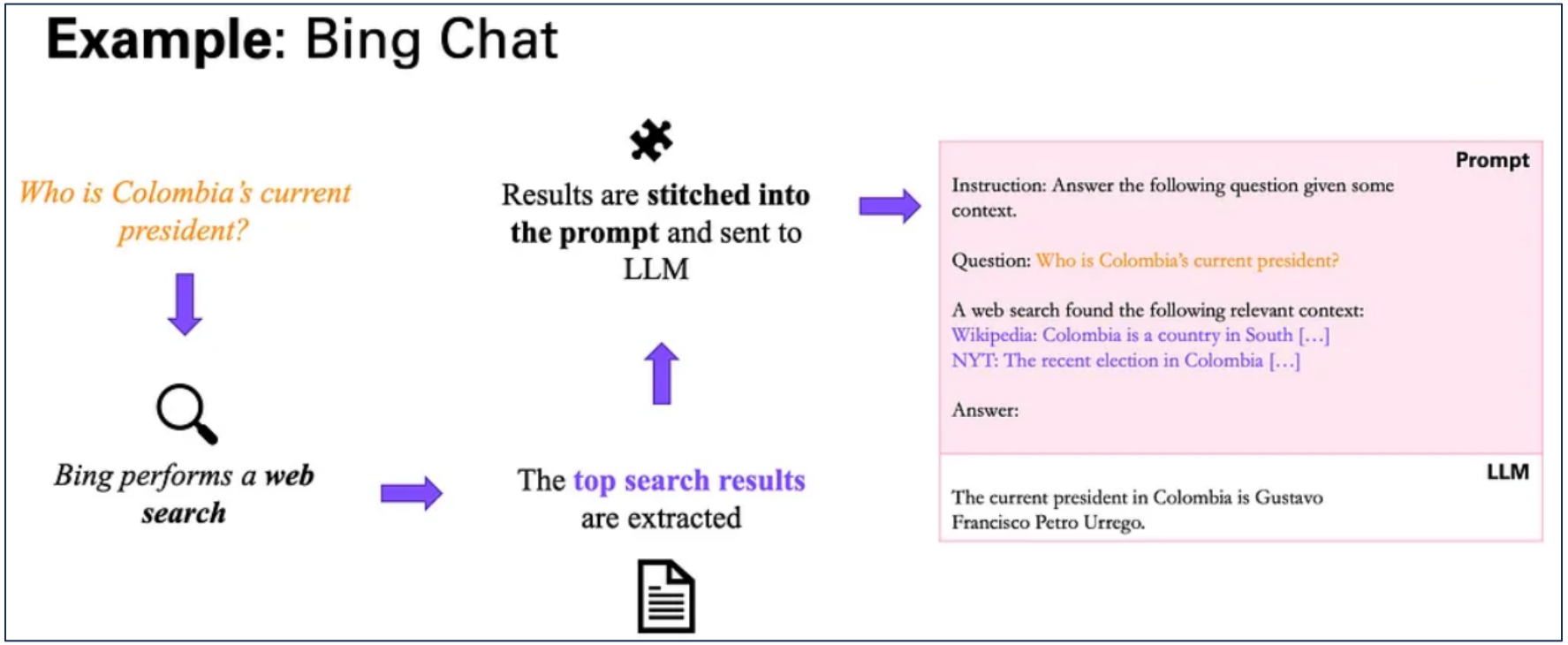


High Level Overview of integrating Enterprise Knowledge with LLM*

* Sachin Kulkarni, **Generative AI with Enterprise Data**
(https://medium.com/@Sachin.Kulkarni.NL/generative-ai-with-enterprise-data-3c81a8bffaf2)

- ChatGPT (as an LLM chat model) has potentially changed how humans **access information**, which has been implemented in the release of **Bing's new chat feature**.

  Bing's new chat feature uses **retrieval augmentation** by **combining ChatGPT with Microsoft's search engine**. Bing Chat generates a search query from your prompt, retrieves relevant documents, and uses them as context for its results. Bing Chat also provides links to sources of information for the sentences it generates.



Bing chat is an example of a search-based LLM workflow*

* https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f

10.3 Retrieval Augmented Generation

Architecture:

A typical RAG application has two main components:

- **Indexing**:

  A pipeline for ingesting data from a source and indexing it. *This usually happens offline.*

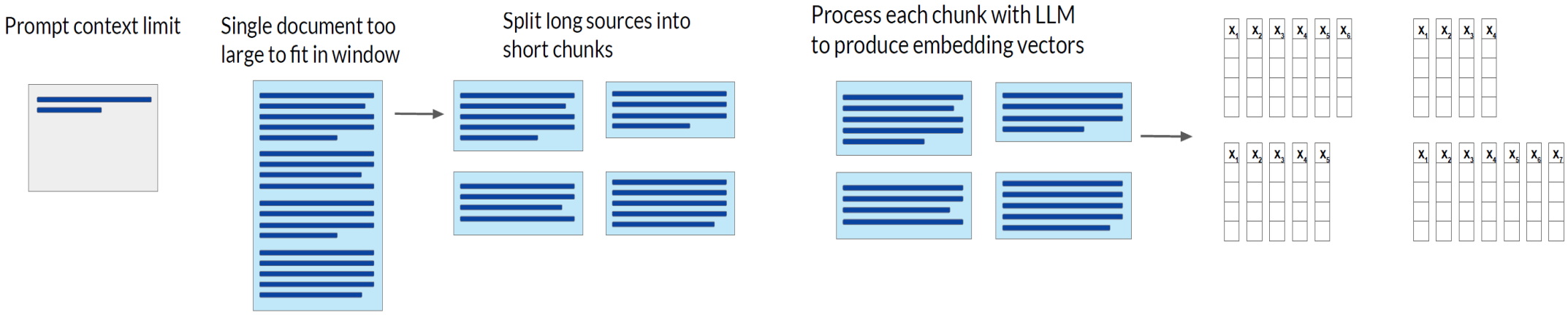
- **Retrieval and generation**:

  The actual RAG chain, which takes the user query at run time and retrieves the relevant data from the index, then passes that (a prompt that includes the user query and the retrieved data) to the model.

1. Data preparation for vector store for RAG

Two considerations for using external data in RAG:

1. Data must fit inside the context window of the LLM (each LLM has a max size (length) of the context window)
2. Data must be in a format that allows its relevance to be assessed at **inference time**: **Embedding vectors (**vector stores enable a **fast and efficient kind of relevant search** based on semantic similarity).



Vector databases are a particular implementation of a vector store where **each vector is also identified by a key**. This can allow, for instance, the text generated by RAG to also include a citation for the document from which it was received.

2. Indexing

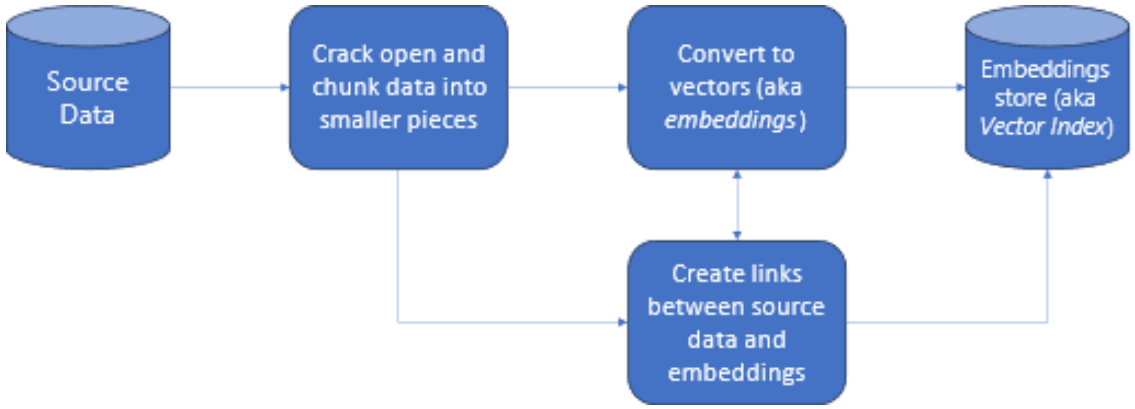
1. **Loading**:
Get the data into the pipeline: text files, PDFs, another website, a database, a Git repository, or from an API

2. **Data chunking (splitting):**
   - Text splitters break large Documents into smaller chunks (splits). Large chunks are **harder to search** over and won't be in a **model's finite context window**.

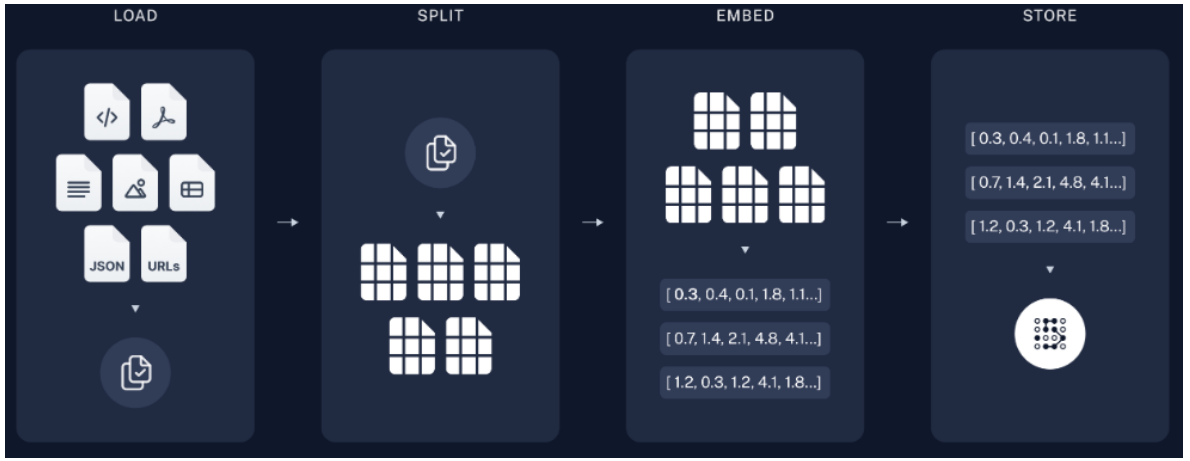
3. **Indexing:**
   - For LLMs, this nearly always means creating vector embeddings, numerical representations of the meaning of your data,
   - Links between source data and embeddings are stored as metadata on the chunks created which are then used to assist the LLMs to generate citations while generating responses.

4. **Storing:**
   - store your index, along with any other metadata, to avoid the need to re-index them.
     - Update external data**:**

     Update the documents and update the embedding representation of the documents through automated real-time processes or periodic batch processing.



Technical overview of using RAG on LLMs*



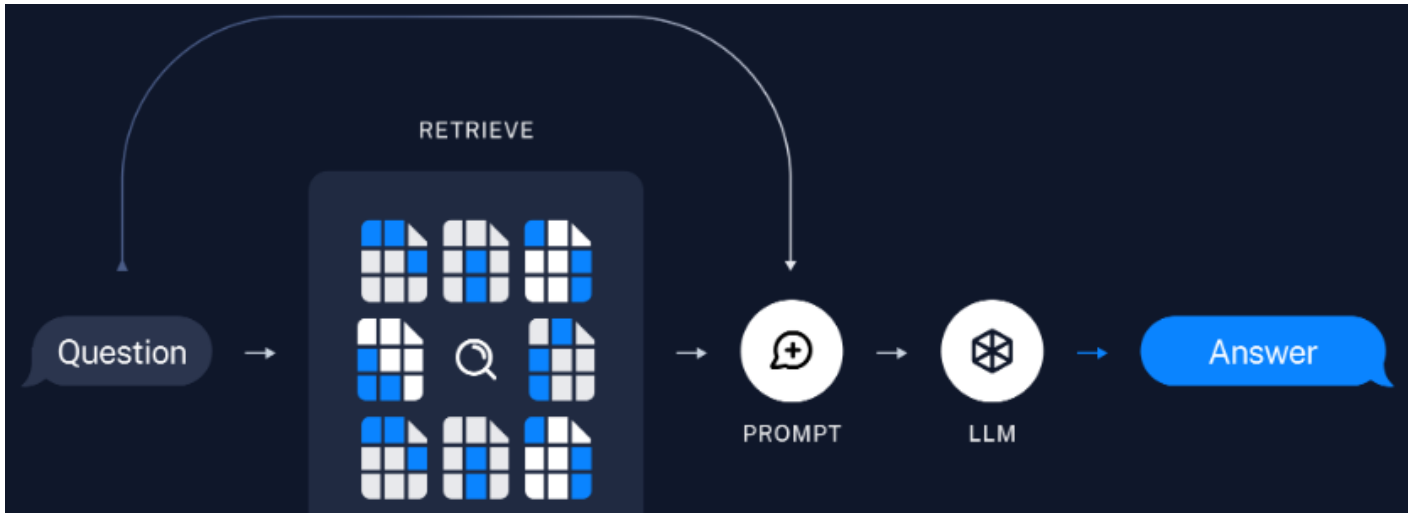
Pipeline for ingesting data from a source and indexing it**

* https://learn.microsoft.com/en-us/azure/machine-learning/concept-retrieval-augmented-generation?view=azureml-api-2
** https://python.langchain.com/docs/use_cases/question_answering/

3. Retrieval and generation:

Retrieval and generation

1. **Retrieve**: Given a user input, relevant splits are retrieved from storage using a retriever.
2. **Generate**: an LLM produces an answer using a prompt that includes the question and the retrieved data.

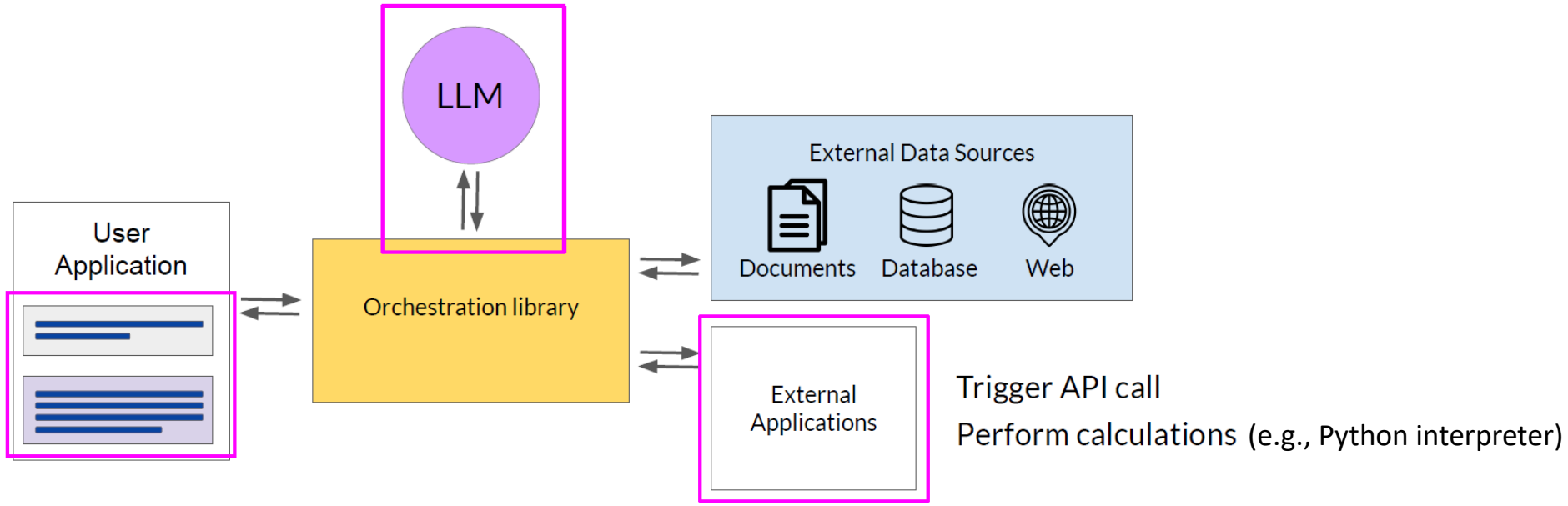


Retrieval and generation *

* https://python.langchain.com/docs/use_cases/question_answering/

10.4 Enabling interactions with external applications

- LLMs in-context learning and reasoning abilities enable **them to interact with tools without training (a.k.a, Zero-Shot Tool Augmentation which is a type of Tool Augmented LLMs)**. The figure below shows how LLMs can interact with external applications (e.g., a code interpreter).
- Different techniques can improve a **model's ability to reason and make plans** for important steps when using an **LLM to power** an application.



DeepLearning.AI (http://deeplearning.ai/)

10.5 Frameworks for developing applications powered by LLMs (LLM-powered applications)

https://www.langchain.com/

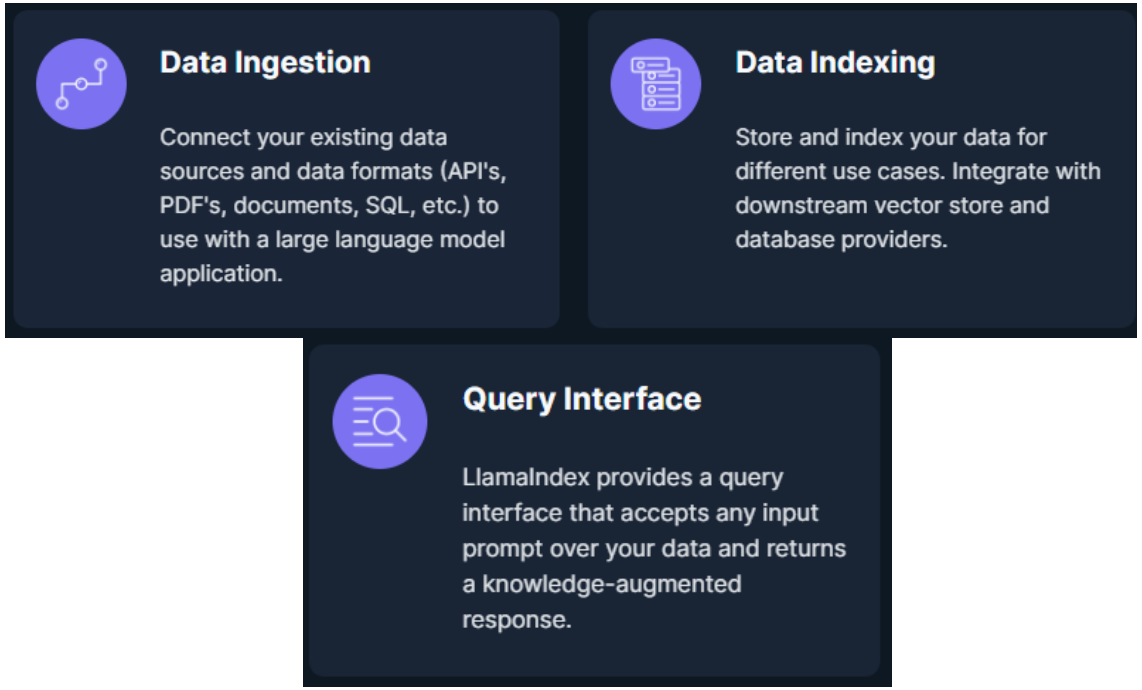
LangChain is a framework for developing applications powered by language models. It enables applications that:

- **Are context-aware**: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
- **Reason**: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

LlamaIndex

https://www.llamaindex.ai/

- LlamaIndex is a simple, flexible data framework for connecting custom data sources to large language models.
- LlamaIndex provides the key tools to augment your LLM applications with data



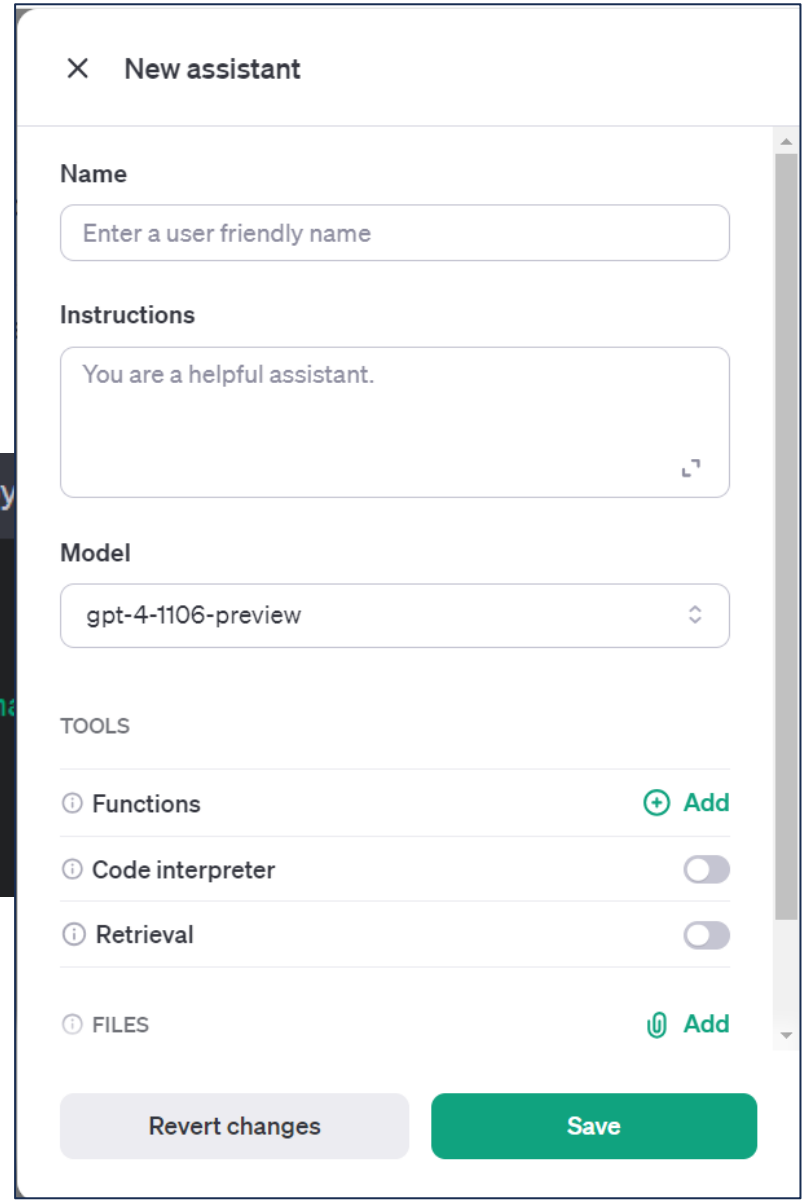
Assistants API endpoint (beta, released on Nov 6, 2023)

OpenAI

- The Assistants API allows developers to build AI assistants within their own applications.
- An assistant is a purpose-built AI that has specific instructions, leverages extra knowledge, and can call models and tools to perform tasks.

```python
assistant = client.beta.assistants.create(
    name="Math Tutor",
    instructions="You are a personal math tutor. Write and run code to answer ma
    tools=[{"type": "code_interpreter"}],
    model="gpt-4-1106-preview"
)
```

Code to create an assistant that is a personal math tutor, with the Code Interpreter tool enabled.



Chat Completions API vs Assistants API

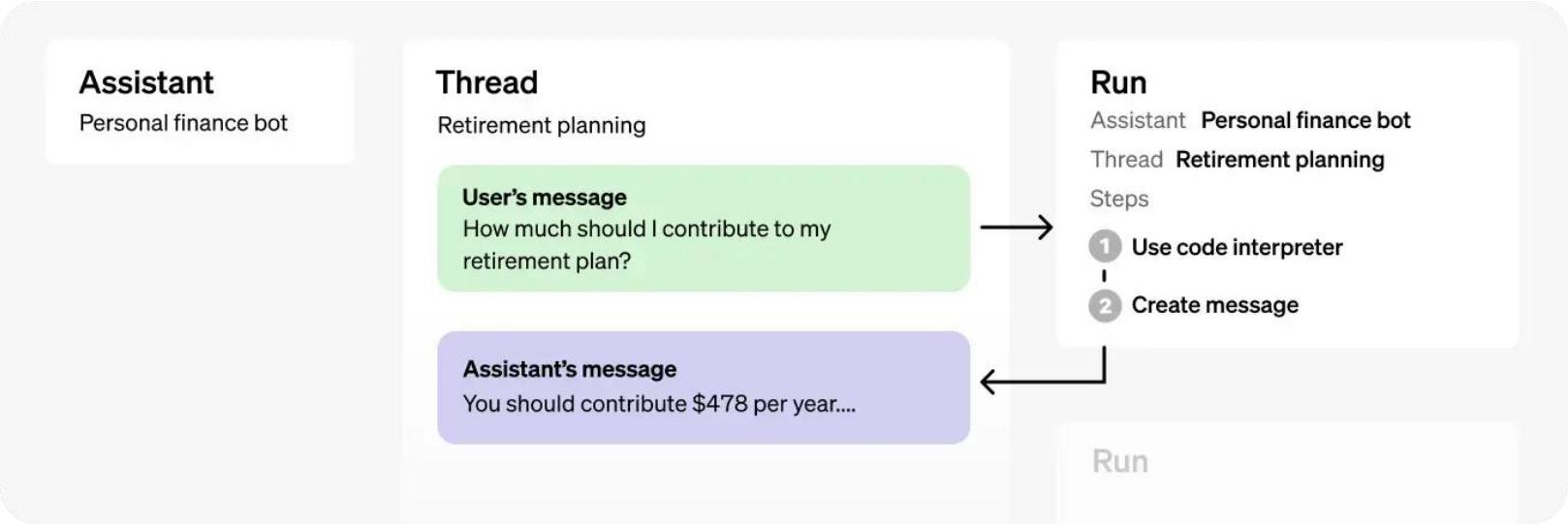
OpenAI

- The Assistants API is a **stateful** evolution of the Chat Completions API meant to simplify the creation of assistant-like experiences and enable developer access to powerful tools like **Code Interpreter** and **Retrieval**.

Chat Completions API	Assistants API
The primitives of the Chat Completions API are Messages, on which you perform a Completion with a Model (gpt-3.5-turbo, gpt-4, etc). It is lightweight and powerful, but inherently **stateless**, which means you have to manage conversation state, tool definitions, retrieval documents, and code execution manually.	The primitives of the Assistants API are: • **Assistants**, which encapsulate a base model, instructions, tools, and (context) documents, • **Threads**, which represent the state of a conversation, • **Runs**, which power the execution of an Assistant on a Thread, including textual responses and multi-step tool use.

Chat Completions API vs Assistants API

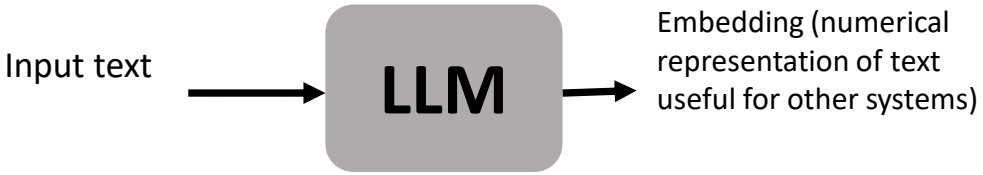
OpenAI

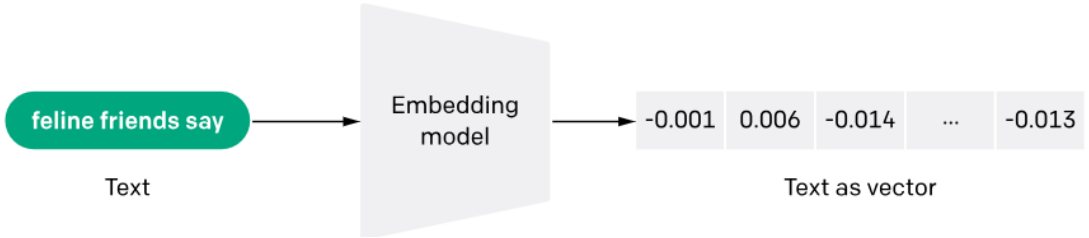


OBJECT	WHAT IT REPRESENTS
Assistant	Purpose-built AI that uses OpenAI's models and **calls tools**
Thread	A conversation session between an Assistant and a user. Threads store Messages and automatically handle truncation to fit content into a model's context.
Message	A message created by an Assistant or a user. Messages can include text, images, and other files. Messages stored as a list on the Thread.
Run	An invocation of an Assistant on a Thread. The Assistant uses its configuration and the Thread's Messages to perform tasks by **calling models and tools**. As part of a Run, the Assistant appends Messages to the Thread.
Run Step	A detailed list of **steps the Assistant took as part of a Run**. An Assistant can **call tools** or **create Messages** during it's run. Examining Run Steps allows you to introspect how the Assistant is getting to its final results.

11. Application of LLMs to embeddings

- LLMs can be used to provide embeddings for ML algorithms. **Embedding models return** a vector representation of a given input that can be easily consumed by machine learning models and algorithms.





Text embeddings measure the relatedness of text strings.
Embeddings are commonly used for:

- **Text Search**
- **Clustering** (where text strings are grouped by similarity)
- **Recommendations** (where items with related text strings are recommended)
- **Classification, Topic clustering, Anomaly detection**
- Preparing data to be fed into a machine learning model

11. OpenAI Embeddings API endpoint

OpenAI

- OpenAI has trained several embedding models with different dimensions and different capabilities.

OpenAI recommends using **text-embedding-ada-002** model for creating text embeddings for nearly all use cases.

Second-generation models

MODEL NAME	TOKENIZER	MAX INPUT TOKENS	OUTPUT DIMENSIONS
text-embedding-ada-002	cl100k_base	8191	1536

Example request — python — Copy

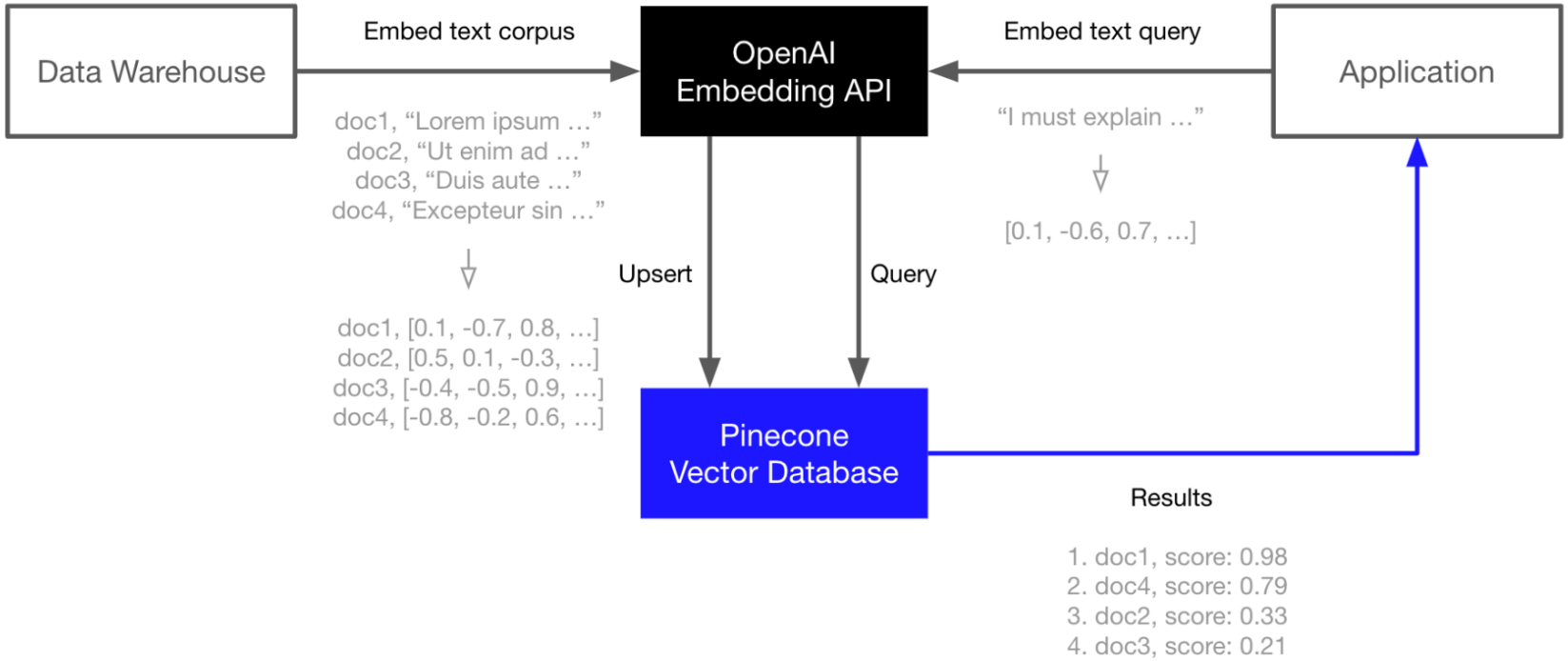
```python
from openai import OpenAI
client = OpenAI()

client.embeddings.create(
  model="text-embedding-ada-002",
  input="The food was delicious and the waiter...",
  encoding_format="float"
)
```

Response — Copy

```json
{
  "object": "list",
  "data": [
    {
      "object": "embedding",
      "embedding": [
        0.0023064255,
        -0.009327292,
        .... (1536 floats total for ada-002)
        -0.0028842222,
      ],
      "index": 0
    }
  ],
  "model": "text-embedding-ada-002",
  "usage": {
    "prompt_tokens": 8,
    "total_tokens": 8
  }
}
```

Application of LLMs to embeddings

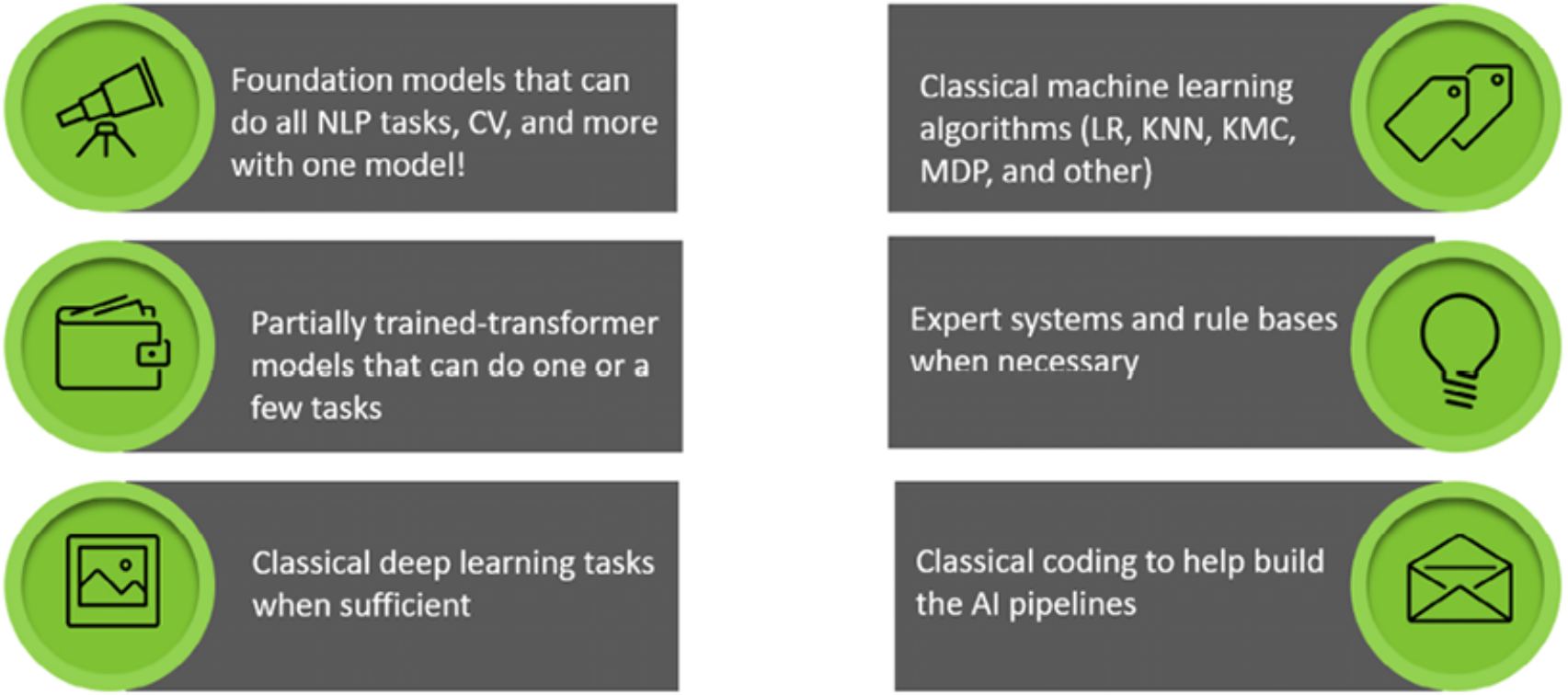
Example of using OpenAI Embedding API in semantic search, question-answering, threat detection ...



Integration of OpenAI's Large Language Models with Pinecone*

* Source: https://docs.pinecone.io/docs/openai

The new paradigm of AI

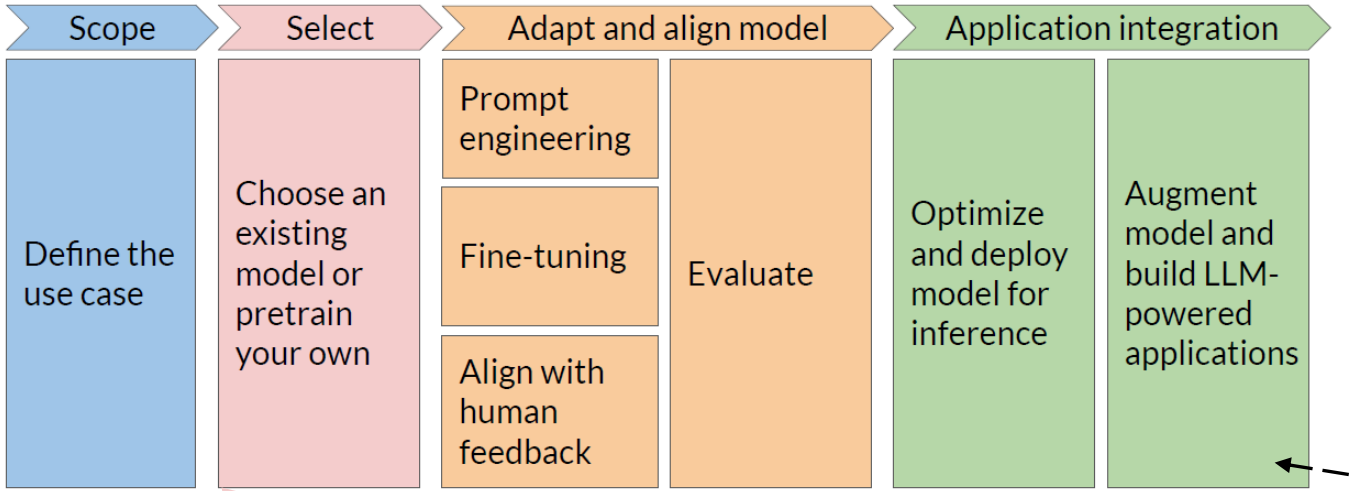


The scope of an Industry 4.0 artificial intelligence (I4.0 AI) specialist (Denis Rothman, 2022)

Foundation models, although designed with an innovative architecture, are built on top of the history of AI. As a result, an artificial intelligence specialist's range of skills is stretching!

13. Generative AI project lifecycle

The overall life cycle of a generative AI project involving LLMs (Figure from DeepLearning.AI, expanded by techniques used at each stage)



- **Stage 1**: Define the scope as accurately and narrowly as you can
- **Stage 2**: Decide whether to train your own model from scratch or work with an existing base model
- **Stage 3**: Adapt and align model (highly iterative)
- **Stage 4**: Application integration

Pre-training:

- *Data Collection*:
  - Data Source, Data Preprocessing
- *Model Architecture*:
  - Decoder-only, Encoder-decoder, etc.
  - Size, Tokenizer, Vocab size, Context window,
  - Decoding Strategy
  - Pre-training Tasks (Next Token Prediction, Denoising Autoencoding, Masked Language Modeling, etc.),
- *Model Training*:
  - Optimization Settings (optimizer, learning rate, …), Scalable Training Techniques

Adapting pre-trained LLMs :

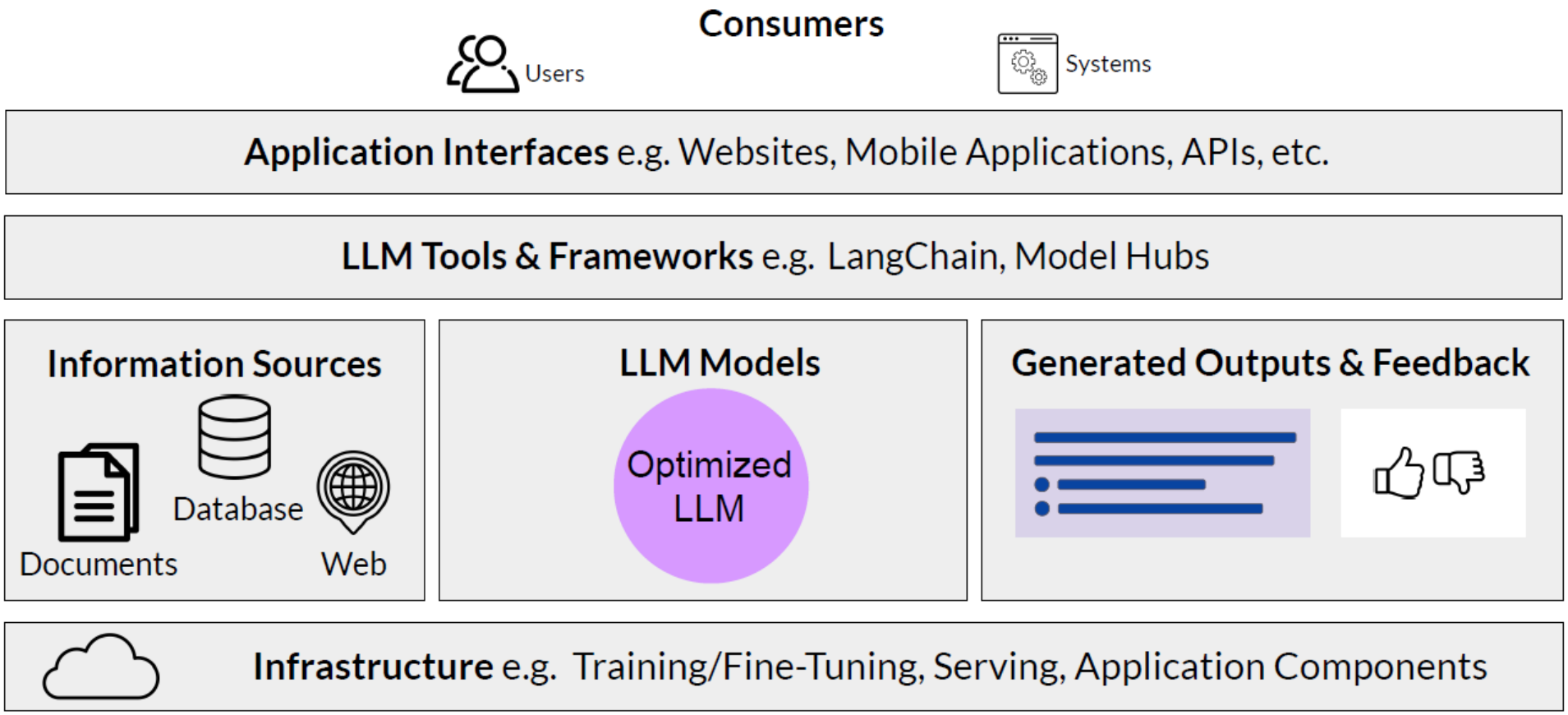
- *Instruction Tuning*:
Approach to fine-tuning pre-trained LLMs on a collection of formatted instances in the form of natural language to follow instructions
- *Alignment Tuning*:
Alignment requires considering very different criteria (e.g., helpfulness, honesty, and harmlessness).
  - Collecting human feedback data for aligning LLMs
  - Reinforcement Learning from Human Feedback (RLHF) for alignment tuning.
- *Parameter-Efficient Fine-Tuning Methods (PEFT)*
PEFT aims to reduce the number of trainable parameters while retaining a good performance as possible.
  - Reparameterization methods: Low-Rank Adaptation (LoRA),
  - Additive methods: Adapter Tuning (add trainable layers), Prefix Tuning, Prompt Tuning (add parameters to model)
  - Selective methods

Memory-Efficient Model Adaptation:
Due to the huge number of model parameters, LLMs take a significant memory footprint for inference, making it very costly to be deployed in real-world applications.
=> Reduce the memory footprint of LLMs
- Quantization Methods for LLMs, …
- *Quantized LLMs*: Compared with original models, quantized language models take a smaller memory footprint, and likely have a faster inference speed.

Augmented LLMs:
- *Retrieval Augmented LLMs*
- *Tool Augmented LLMs*

* The end users can also use prompt engineering techniques to develop and optimize prompts to efficiently use language models for a variety of applications

Building generative applications



DeepLearning.AI (http://deeplearning.ai/

Challenges and some future directions

- **Transformer complexity:**

  With a sequence of length L:

L=100	L^2 = 10K	(0.001s at 10M ops/s)
L=1000	L^2 = 1M	(0.1s at 10M ops/s)
L=10000	L^2 = 100M	(10s at 10M ops/s)
L=100000	L^2 = 10B	(1000s at 10M ops/s)

=> Efficiency has become an important issue when training and making inferences with long inputs.

=> Reduce the time complexity (originally to be quadratic costs) incurred by the standard self-attention mechanism

Retentive Network: A Successor to Transformer for Large Language Models

Yutao Sun*†‡ Li Dong*† Shaohan Huang† Shuming Ma†
Yuqing Xia† Jilong Xue† Jianyong Wang‡ Furu Wei†◇
† Microsoft Research ‡ Tsinghua University
https://aka.ms/GeneralAI

Abstract

In this work, we propose **Retentive Network** (RETNET) as a foundation architecture for large language models, simultaneously achieving training parallelism, low-cost inference, and good performance. We theoretically derive the connection between recurrence and attention. Then we propose the retention mechanism for sequence modeling, which supports three computation paradigms, i.e., parallel, recurrent, and chunkwise recurrent. Specifically, the parallel representation allows for training parallelism. The recurrent representation enables low-cost $O(1)$ inference, which improves decoding throughput, latency, and GPU memory without sacrificing performance. The chunkwise recurrent representation facilitates efficient long-sequence modeling with linear complexity, where each chunk is encoded parallelly while recurrently summarizing the chunks. Experimental results on language modeling show that RETNET achieves favorable scaling results, parallel training, low-cost deployment, and efficient inference. The intriguing properties make RETNET a strong successor to Transformer for large language models. Code will be available at https://aka.ms/retnet.

Challenges and some future directions

- **Emergent abilities**: when and how they are obtained by LLMs are not yet clear.

- **Catastrophic forgetting** (challenge for neural networks => has a negative impact on LLMs)
  - Task specialization: fine-tuning a LLM according to some specific tasks => can affect the general ability of LLMs
  - Alignment tuning with human values => alignment tax

- **Existing prompting approaches**:
  - Involve considerable **human efforts** in the design of prompts => automatic generation of effective prompts !
  - Lack flexible task formatting methods for complex tasks requiring **logic rules** (e.g., numerical computation)

- **RLHF** heavily relies on **high-quality human feedback** data from professional labelers
  => Difficult implementation in practice

References

- [1] M. Shanahan, Talking about large language models, CoRR, vol. abs/2212.03551, 2022.
- [2] T. B. Brown et al., Language models are few-shot learners," in Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020,
- [3] (A. Chowdhery et al., "Palm: Scaling language modeling with pathways," CoRR, vol. abs/2204.02311, 2022.
- [4] R. Taylor et al., "Galactica: A large language model for science," CoRR, vol. abs/2211.09085, 2022.
- [5] H. Touvron et al., "Scaling instruction-finetuned language models," CoRR, vol. abs/2210.11416, 2022.
- Alto, V. 2023. *Modern Generative AI with ChatGPT and OpenAI Models: Leverage the capabilities of OpenAI's LLM for productivity and innovation with GPT3 and GPT4*. Packt Publishing.
- Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M.S., Bohg, J., Bosselut, A., Brunskill, E. and Brynjolfsson, E., 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., et al., 2020. Language Models are Few-Shot Learners. ArXiv. /abs/2005.14165
- Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Dong, L., Xu, S. and Xu, B., 2018, April. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)* (pp. 5884-5888). IEEE.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S. and Uszkoreit, J., 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.
- Glaese, A., McAleese, N., Trębacz, M., Aslanides, J., Firoiu, V., Ewalds, T., Rauh, M., Weidinger, L., Chadwick, M., Thacker, P. and Campbell-Gillingham, L., 2022. Improving alignment of dialogue agents via targeted human judgements. *arXiv preprint arXiv:2209.14375*.
- Khan, S., Naseer, M., Hayat, M., Zamir, S.W., Khan, F.S. and Shah, M., 2022. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, *54*(10s), pp.1-41.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.T., Rocktäschel, T. and Riedel, S., 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, *33*, pp.9459-9474.
- Li, W., Luo, H., Lin, Z., Zhang, C., Lu, Z. and Ye, D., 2023. A survey on transformers in reinforcement learning. *arXiv preprint arXiv:2301.03044*.
- Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C.L., Ma, J. and Fergus, R., 2021. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, *118*(15), p.e2016239118.
- Rothman, D. and Gulli, A., 2022. *Transformers for Natural Language Processing: Build, train, and fine-tune deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, and GPT-3*. Packt Publishing Ltd.
- Schwaller, P., Laino, T., Gaudin, T., Bolgar, P., Hunter, C.A., Bekas, C. and Lee, A.A., 2019. Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction. *ACS central science*, *5*(9), pp.1572-1583.
- M. Shanahan, 2022. "Talking about large language models," CoRR, vol. abs/2212.03551, 2022.
- A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, et al., 2022. "Palm: Scaling language modeling with pathways," CoRR, vol. abs/2204.02311, 2022.
- H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, et al., 2023. "Llama: Open and efficient foundation language models," CoRR, 2023.
- Tunstall, L., Von Werra, L. and Wolf, T., 2022. *Natural language processing with transformers*. " O'Reilly Media, Inc."
- R. Taylor, M. Kardas, G. Cucurull, T. Scialom, et al., 2022. "Galactica: A large language model for science," CoRR, vol. abs/2211.09085, 2022.
- Takeshi Kojima et al., 2023. Large Language Models are Zero-Shot Reasoners. https://arxiv.org/pdf/2205.11916.pdf
- Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z. and Du, Y., 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems*, *30*.
- Wei, Jason et al., 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. https://arxiv.org/abs/2201.11903

  - G. Izacard, P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave, "Few-shot learning with retrieval augmented language models," arXiv preprint arXiv:2208.03299, 2022.

  - S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark et al., "Improving language models by retrieving from trillions of tokens," in International conference on machine learning. PMLR, 2022, pp. 2206–2240.

  - O. Ram, Y. Levine, I. Dalmedigos, D. Muhlgay, A. Shashua, K. Leyton-Brown, and Y. Shoham, "In-context retrieval-augmented language models," arXiv preprint arXiv:2302.00083, 2023.

  - X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, and D. Zhou, "Rationale-augmented ensembles in language models," arXiv preprint arXiv:2207.00747, 2022.

  - H. Naveed et al., 2023

Webgraphy:

- DeepLearning.AI (http://deeplearning.ai/)
- Huggingface (https://huggingface.co/docs/transformers/index)