Generative AI and Large Language Models: Inner Workings, Recent Advances, and Practical use in Academia

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

INNS-Morocco Regional Chapter

May 28, 2025

Some statistics about Large Language Models

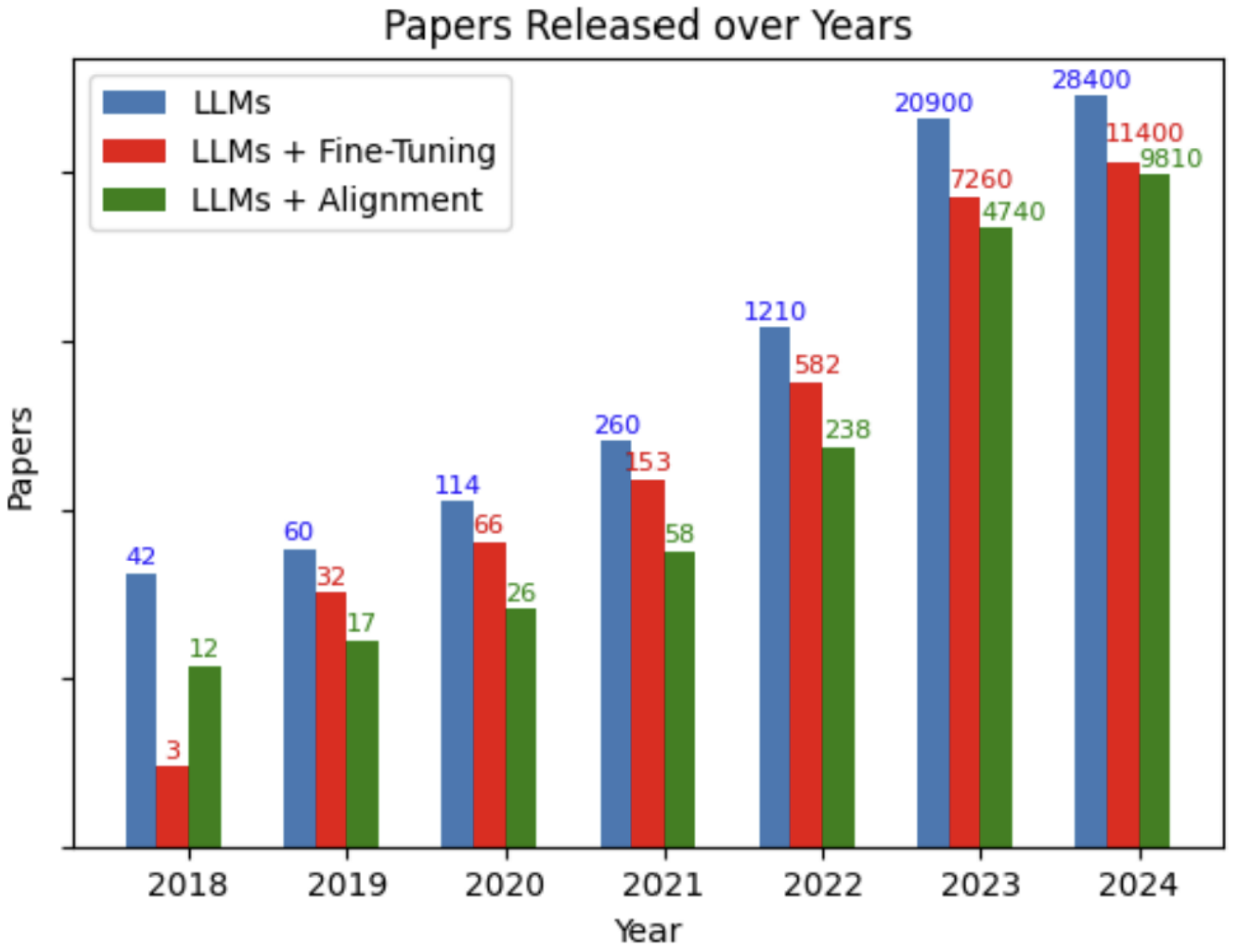
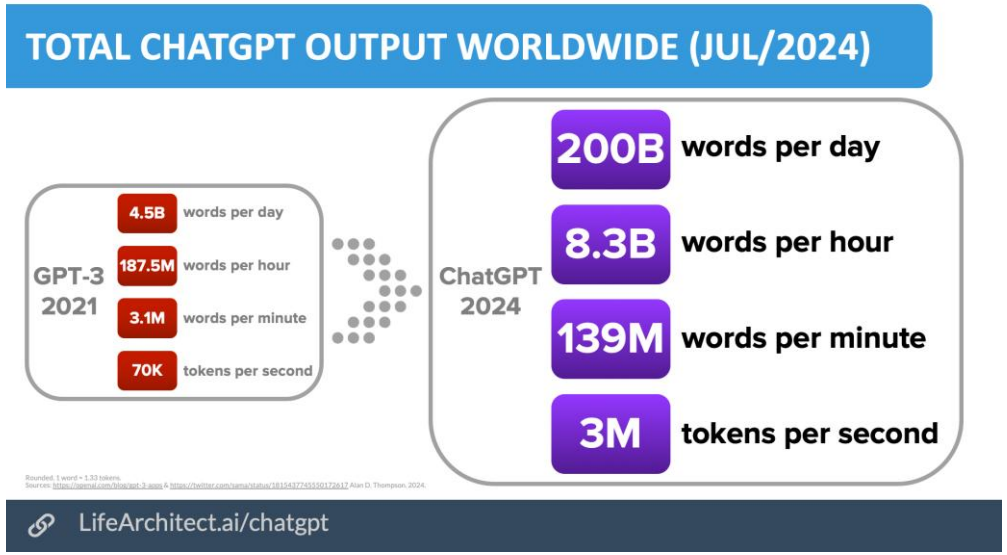


Figure 1: The trend of papers released over the years containing keywords "Large Language Model", "Large Language Model + Fine-Tuning", and "Large Language Model + Alignment". Source: Naveed, Humza, et al., 2023 (v10, Oct 2024)



E. Nfaoui

Outline

- Inner working of LLMs, Language Modeling Objective, LLM Architectures
- Training Stages, Decoding Strategy
- Types of LLMs
- Prompt engineering, Automatic prompt generation
- Augmented LLMs and LLM-powered applications
- Retrieval Augmented Generation (RAG), Multimodal RAG
- LLM-based Autonomous Agents
- Generative AI life cycle project

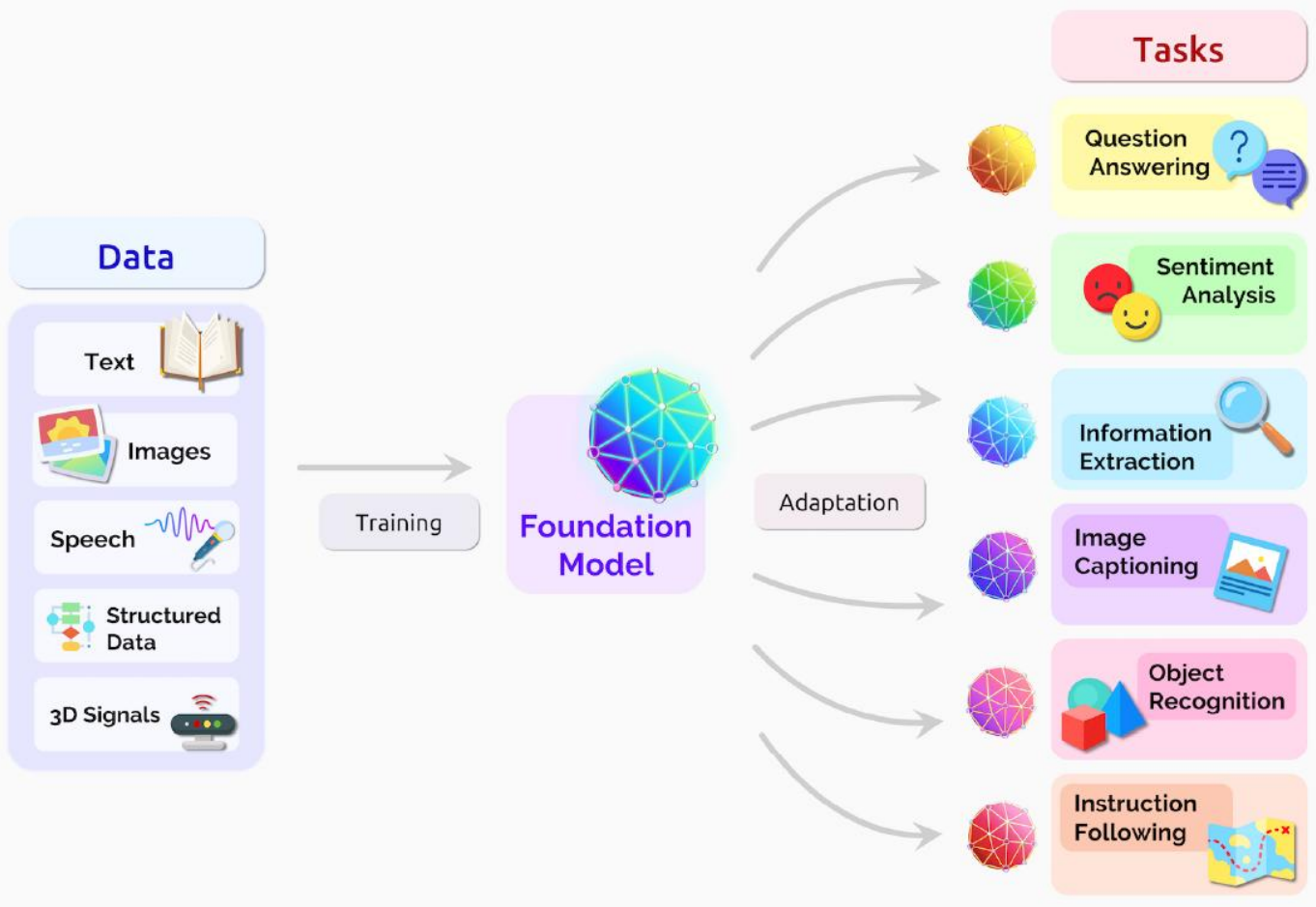
Introduction

What can foundation models do?

- A **foundation model** can centralize the information from all the data from various modalities. It is trained on broad data (generally using self-supervision at scale). This one model can then be adapted to a wide range of downstream tasks (Bommansani et al., 2021)
- Foundation models, are already being used with many data sources for a host of applications.

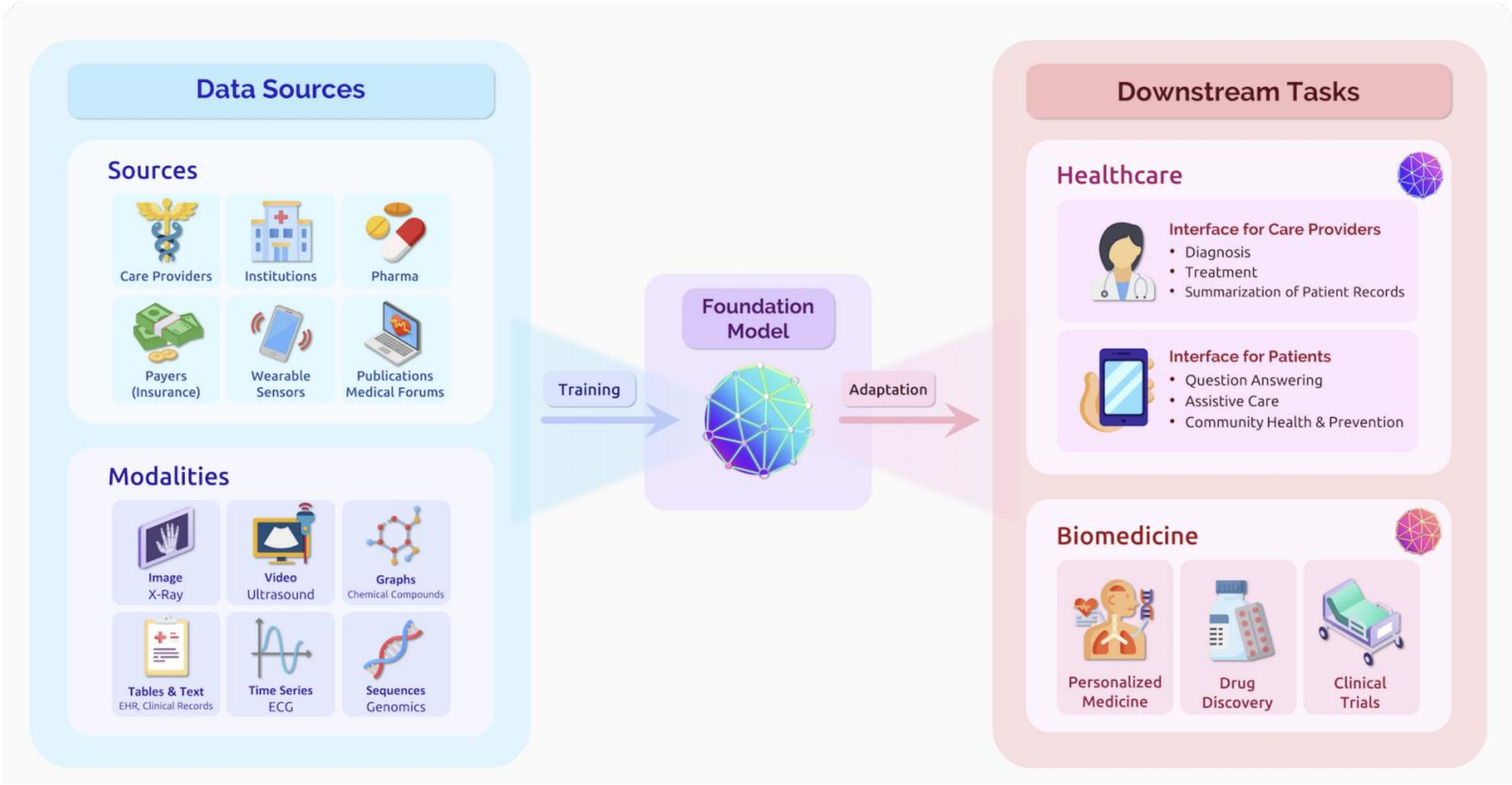
Conference Scope: Large Language Models

- A **subset of foundation models** (FMs) called **Large Language Models (LLMs)** are trained on trillions of words across many natural-language tasks. They are powered by **the transformer architecture**.



Foundation models (Bommansani et al., 2021)

Foundation models in healthcare and biomedicine



Foundation models in healthcare and biomedicine. We visualize an interactive framework where foundation models enable various tasks across healthcare and biomedicine when trained on multimodal data generated by various sources in the healthcare ecosystem. The first column lists several sources of data, including care providers, payers, institutions (universities, non-profits, and governments), pharma, wearables, and medical publications/forums. The second column shows several data modalities generated by the data sources. They include images (e.g., chest X-rays), videos (such as ultrasounds), graphs of chemical compounds, tables for electronic health records (EHRs), text such as clinical notes, time series such as ECGs, and genetic data. The third column visualizes a foundation model trained on such data and then applied to healthcare and biomedicine downstream tasks listed in the fourth column. This process can generate new data that will further improve the foundation model, hence the bidirectional relation between the foundation models and the tasks. (Bommansani et al., 2021)

On the Opportunities and Risks of Foundation Models

Rishi Bommasani* Drew A. Hudson Ehsan Adeli Russ Altman Simran Arora
Sydney von Arx Michael S. Bernstein Jeannette Bohg Antoine Bosselut Emma Brunskill
Erik Brynjolfsson Shyamal Buch Dallas Card Rodrigo Castellon Niladri Chatterji
Annie Chen Kathleen Creel Jared Quincy Davis Dorottya Demszky Chris Donahue
Moussa Doumbouya Esin Durmus Stefano Ermon John Etchemendy Kawin Ethayarajh
Li Fei-Fei Chelsea Finn Trevor Gale Lauren Gillespie Karan Goel Noah Goodman
Shelby Grossman Neel Guha Tatsunori Hashimoto Peter Henderson John Hewitt
Daniel E. Ho Jenny Hong Kyle Hsu Jing Huang Thomas Icard Saahil Jain
Dan Jurafsky Pratyusha Kalluri Siddharth Karamcheti Geoff Keeling Fereshte Khani
Omar Khattab Pang Wei Koh Mark Krass Ranjay Krishna Rohith Kuditipudi
Ananya Kumar Faisal Ladhak Mina Lee Tony Lee Jure Leskovec Isabelle Levent
Xiang Lisa Li Xuechen Li Tengyu Ma Ali Malik Christopher D. Manning
Suvir Mirchandani Eric Mitchell Zanele Munyikwa Suraj Nair Avanika Narayan
Deepak Narayanan Ben Newman Allen Nie Juan Carlos Niebles Hamed Nilforoshan
Julian Nyarko Giray Ogut Laurel Orr Isabel Papadimitriou Joon Sung Park Chris Piech
Eva Portelance Christopher Potts Aditi Raghunathan Rob Reich Hongyu Ren
Frieda Rong Yusuf Roohani Camilo Ruiz Jack Ryan Christopher Ré Dorsa Sadigh
Shiori Sagawa Keshav Santhanam Andy Shih Krishnan Srinivasan Alex Tamkin
Rohan Taori Armin W. Thomas Florian Tramèr Rose E. Wang William Wang Bohan Wu
Jiajun Wu Yuhuai Wu Sang Michael Xie Michihiro Yasunaga Jiaxuan You Matei Zaharia
Michael Zhang Tianyi Zhang Xikun Zhang Yuhui Zhang Lucia Zheng Kaitlyn Zhou
Percy Liang*[1]

Center for Research on Foundation Models (CRFM)
Stanford Institute for Human-Centered Artificial Intelligence (HAI)
Stanford University

AI is undergoing a paradigm shift with the rise of models (e.g., BERT, DALL-E, GPT-3) trained on broad data (generally using self-supervision at scale) that can be adapted to a wide range of downstream tasks. We call these models foundation models *to underscore their critically central yet incomplete character. This report provides a thorough account of the opportunities and risks of foundation models, ranging from their capabilities (e.g., language, vision, robotic manipulation, reasoning, human interaction) and technical principles (e.g., model architectures, training procedures, data, systems, security, evaluation, theory) to their applications (e.g., law, healthcare, education) and societal impact (e.g., inequity, misuse, economic and environmental impact, legal and ethical considerations). Though foundation models are based on standard deep learning and transfer learning, their scale results in new emergent capabilities, and their effectiveness across so many tasks incentivizes homogenization. Homogenization provides powerful leverage but demands caution, as the defects of the foundation model are inherited by all the adapted models downstream. Despite the impending widespread deployment of foundation models, we currently lack a clear understanding of how they work, when they fail, and what they are even capable of due to their emergent properties. To tackle these questions, we believe much of the critical research on foundation models will require deep interdisciplinary collaboration commensurate with their fundamentally sociotechnical nature.*

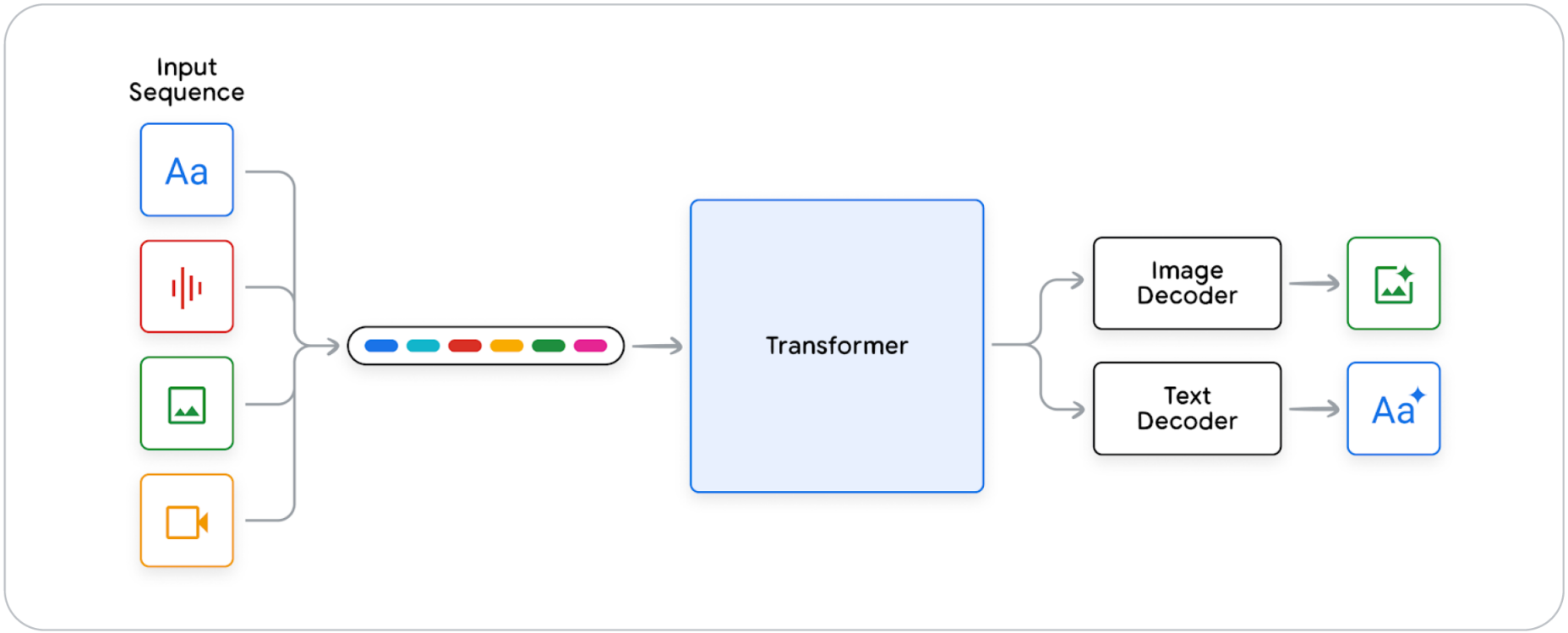
[1]Corresponding author: pliang@cs.stanford.edu

*Equal contribution.

This report consists of 214 pages

E. Nfaoui

Gemini



Gemini can receive multi-modal inputs including **text, audio, images, and video data**. These are all **tokenized** and **fed into its transformer model**. The transformer generates an output that can contain images and text.

- Gemini is a state-of-the-art **multimodal language family** of models that can take interleaved sequences of text, image, audio, and video as input. It's built on top of transformer decoders and has architectural improvements for scale as well as optimized inference on Google's Tensor Processing Units (TPUs).
- Gemini models also employ a **Mixture of Experts architecture** to optimize efficiency and capabilities of the models.
- The pre-training data consists of **web documents**, **books**, **code**, and **image**, **audio**, and **video data**.

Major development stages of Language Modeling Approach

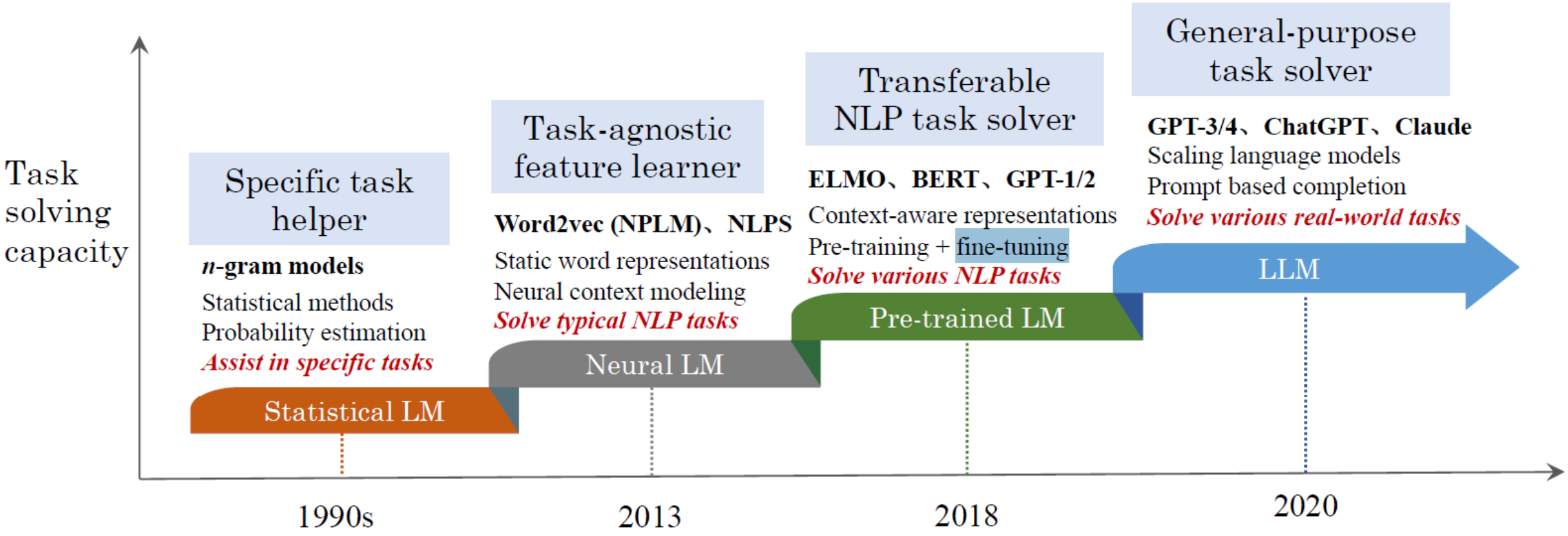
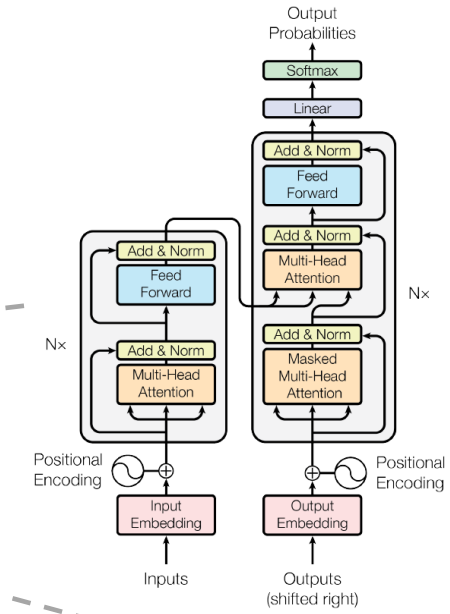


Fig. 2: An evolution process of the four generations of language models (LM) from the perspective of task solving capacity. Note that the time period for each stage may not be very accurate, and we set the time mainly according to the publish date of the most representative studies at each stage. For neural language models, we abbreviate the paper titles of two representative studies to name the two approaches: NPLM [1] (*"A neural probabilistic language model"*) and NLPS [2] (*"Natural language processing (almost) from scratch"*). Due to the space limitation, we don't list all representative studies in this figure.

Source: Wayne Xin Zhao et al., 2023 (v16 , Mar 2025)

2. 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**)." (Zhao, Wayne X., 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."

The Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

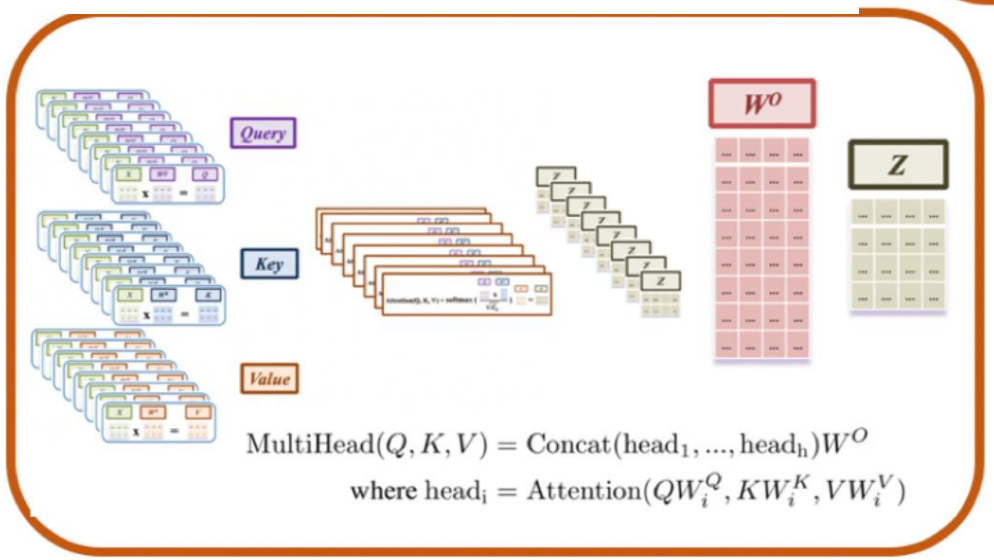
Jakob Uszkoreit* Google Research usz@google.com

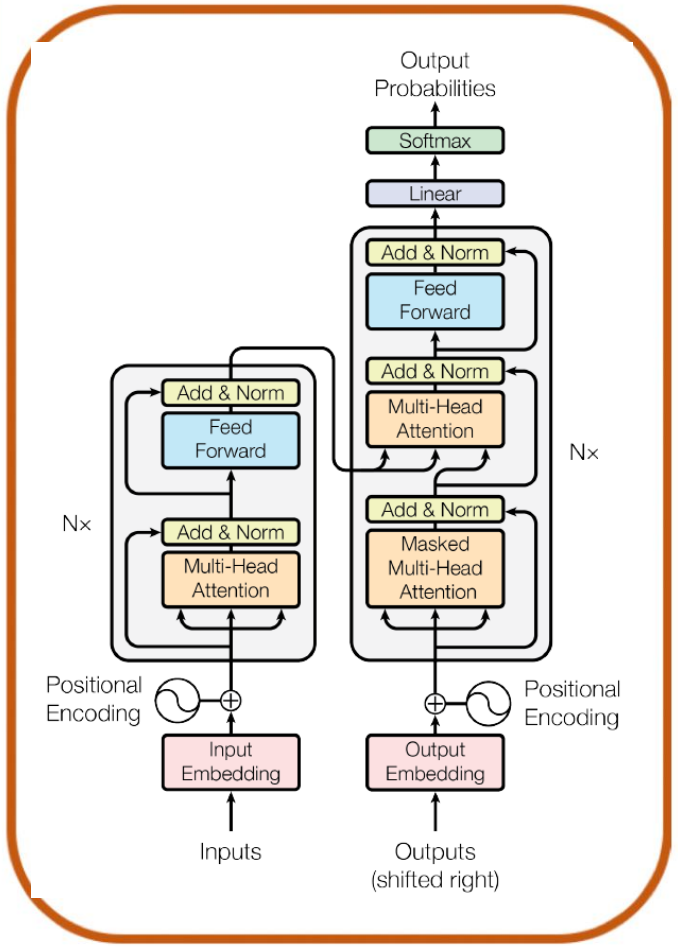
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com





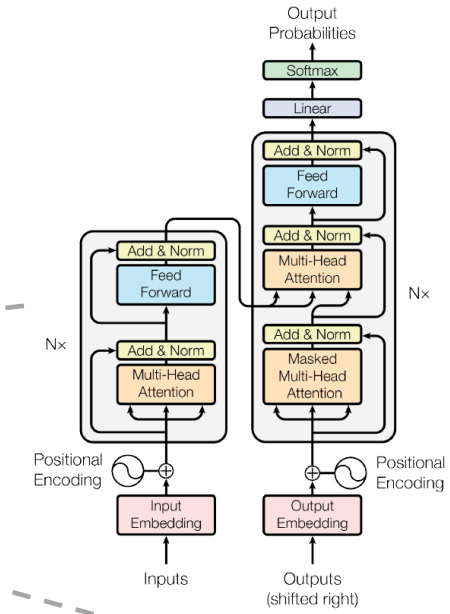
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

The basic operation of attention, with Q=query, K=Keys and V=Value, d_k = dimension of queries and keys, Z=Attention

E. Nfaoui

2. 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**)." (Zhao, Wayne X., 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** →

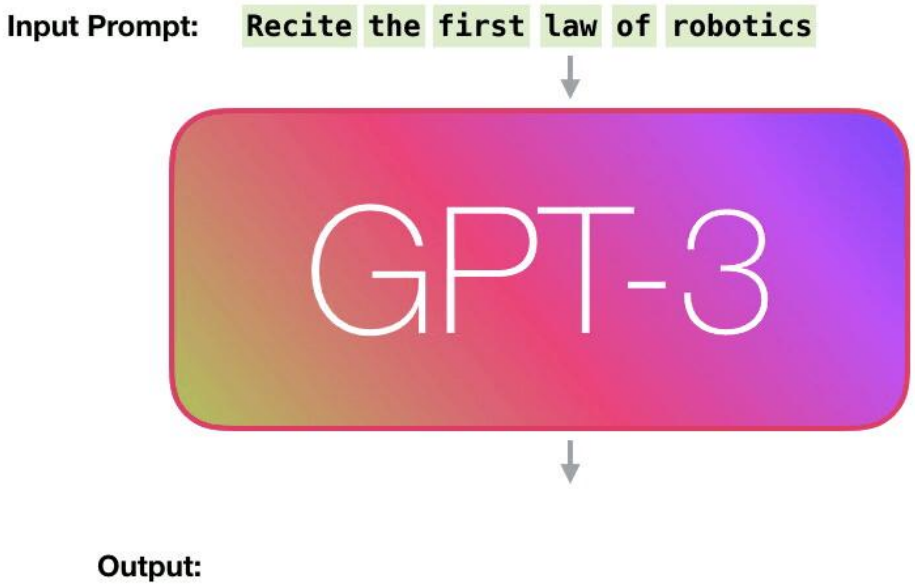
Arabic translation: "الفقراء والطبقة الوسطى يعملون من أجل المال. الأغنياء لديهم المال ليعملوا عنهم."

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

Text generation (autoregressive models)

An LLM generates output one token at a time.

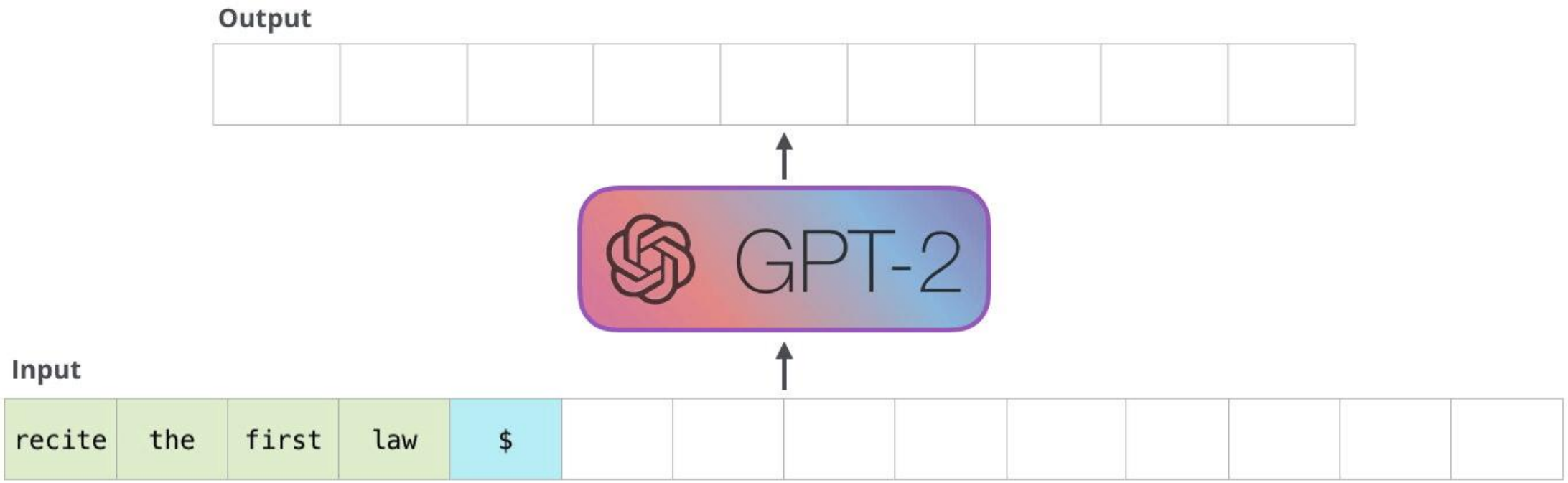
Example: GPT-3



Source: Alammar, J. How GPT3 Works - Visualizations and Animations [Blog post]

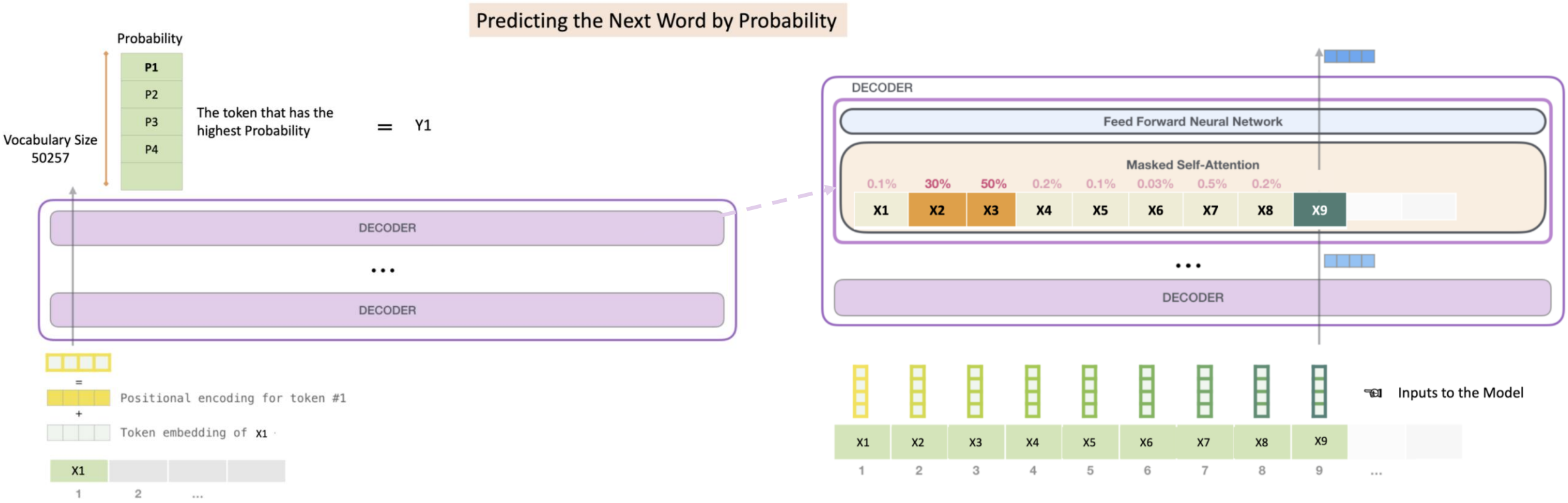
5. Text Generation:
General working flow of an Language Model predicting the next word (autoregressive model)

The way these models actually work is that after each token is produced, that token is added to the sequence of inputs, and that new sequence becomes the input to the model in its next step. This is an idea called “auto-regression”.



Source: Alammar, J (2019). The Illustrated GPT-2 [Blog post]. https://jalammar.github.io/illustrated-gpt2/

Predicting the next Token
Example : GPT-3

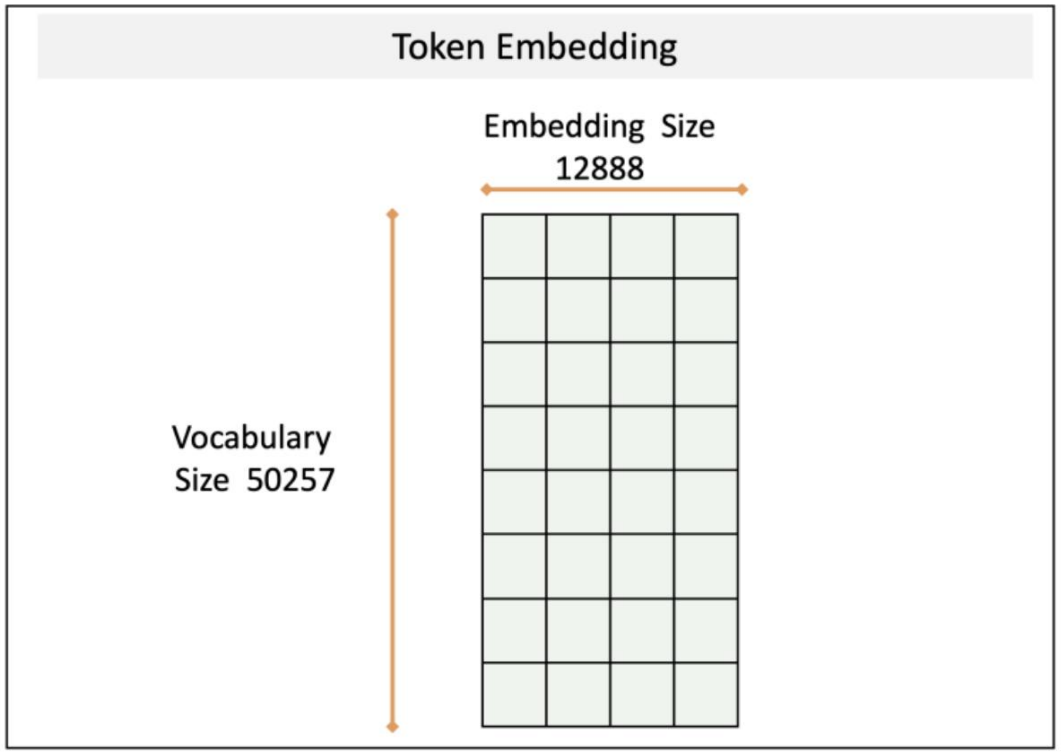


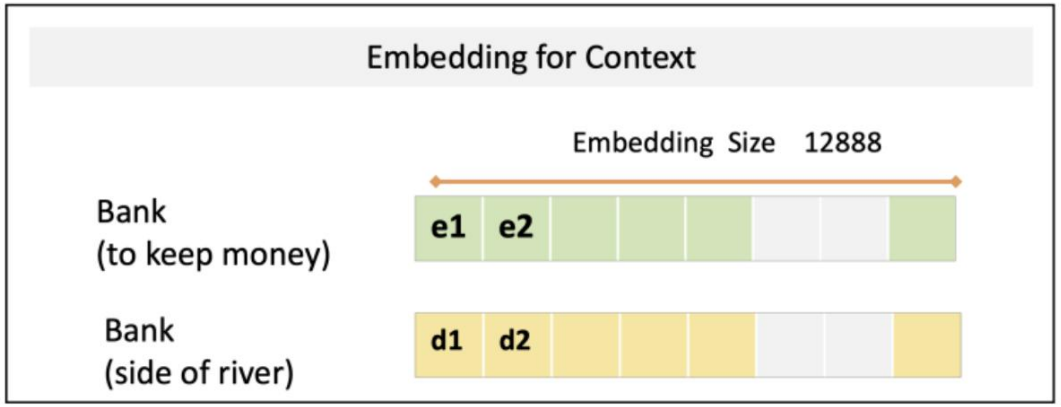
- As a rough rule of thumb, 1 token is approximately 4 characters or 0.75 words for English text (1000 tokens are roughly equal to 750 words.)
- GPT language models tokenizers: https://platform.openai.com/tokenizer

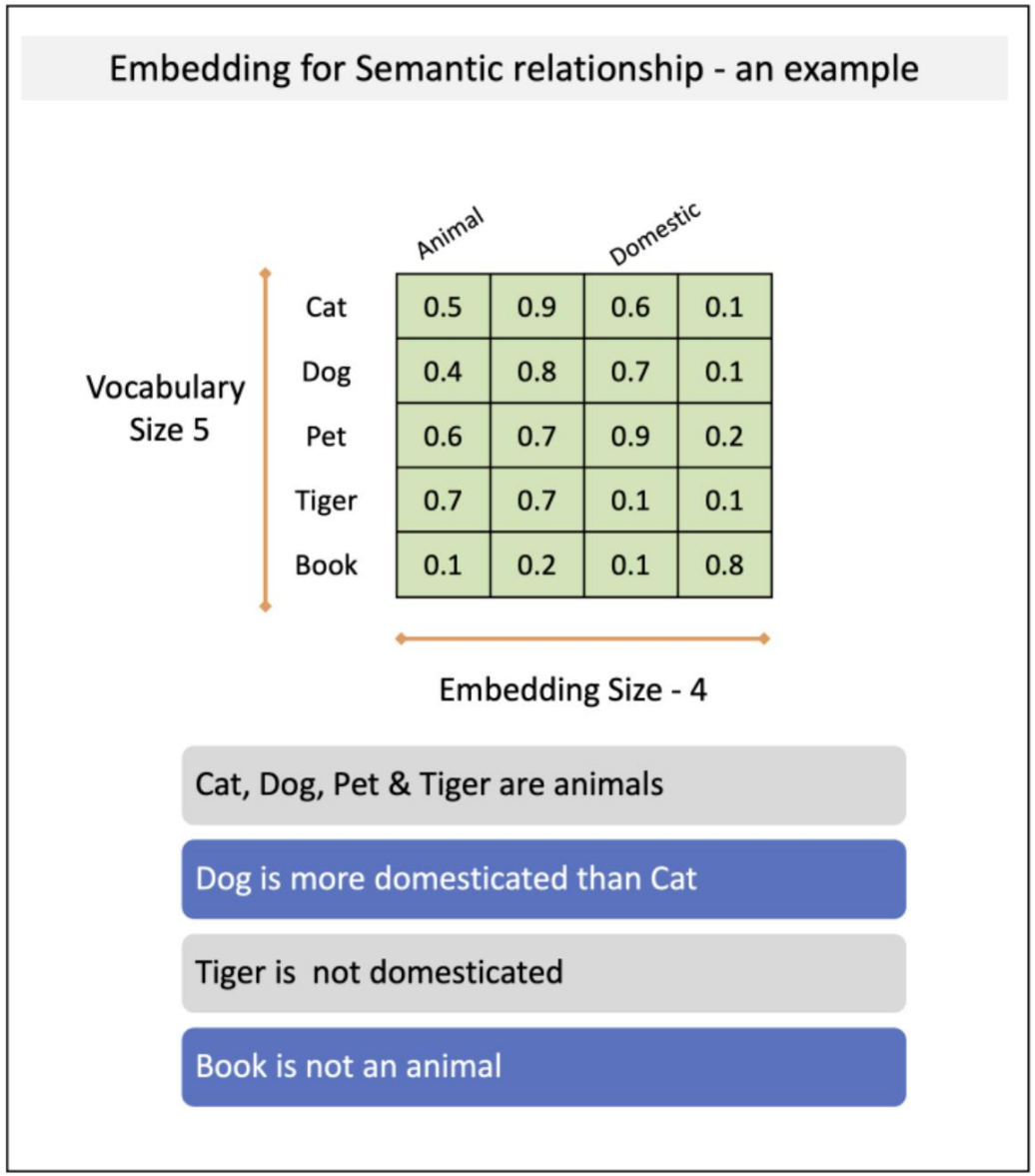
Token Embedding Example : GPT-3

- GPT-3 has a vocabulary of **50257 tokens.**
- Each token is represented by a vector of **12888 elements (embedding size)**.

This vector representation helps to capture the semantic meanings of the tokens.







Embeddings Demo

6. LLM Training stages at a high level

- Different types of training:
  Large Language Models typically undergo multiple training stages:

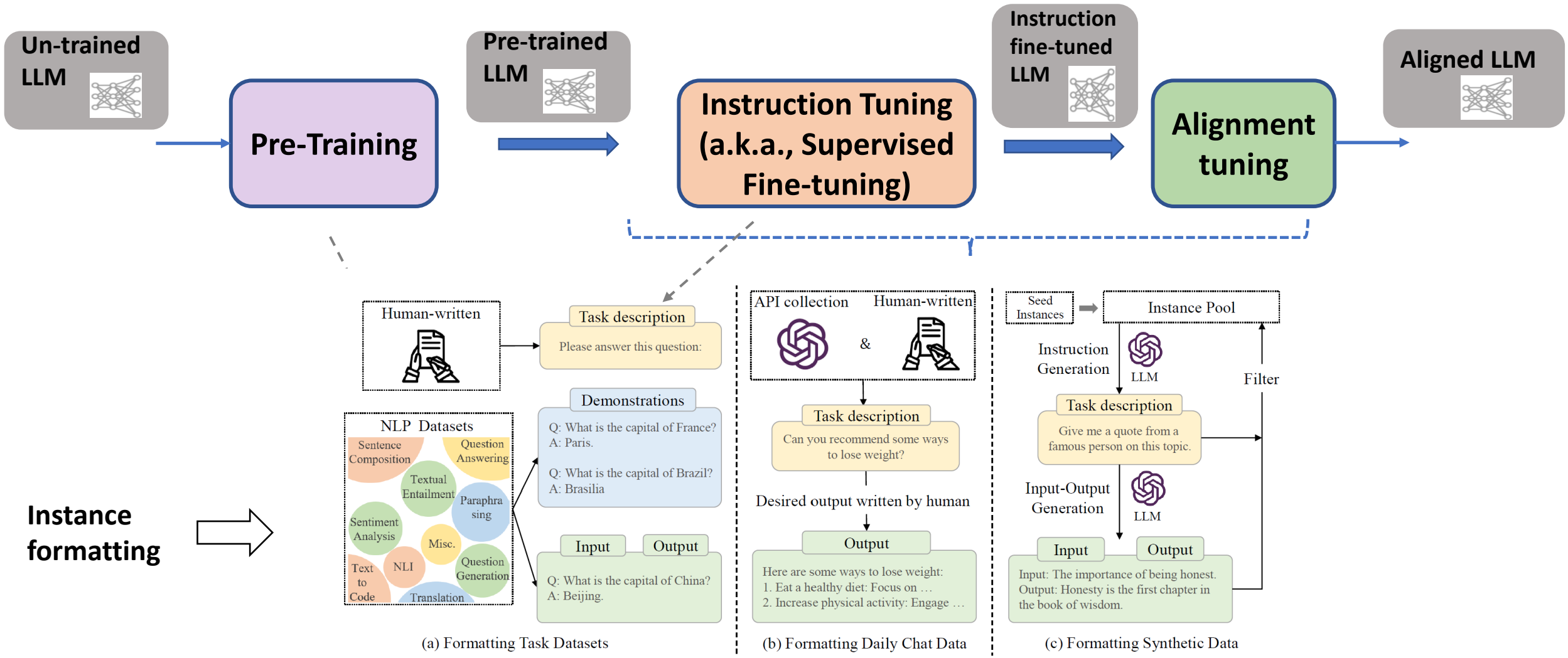
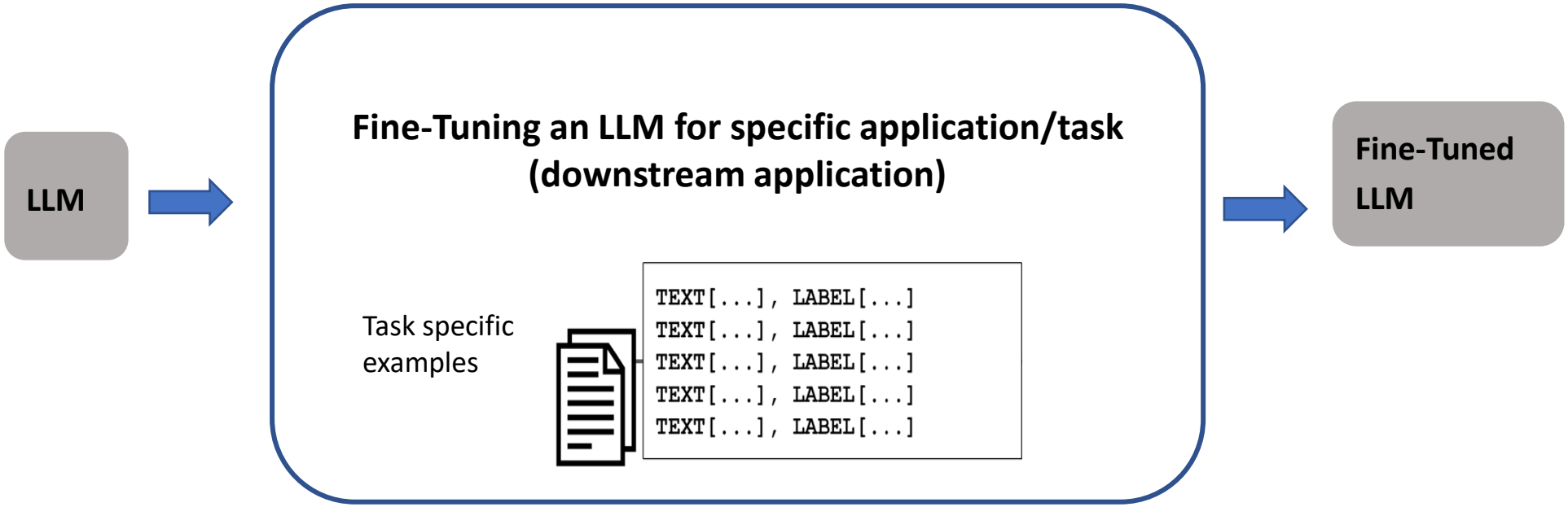


Fig. 11: An illustration of instance formatting and three different methods for constructing the instruction-formatted instances.

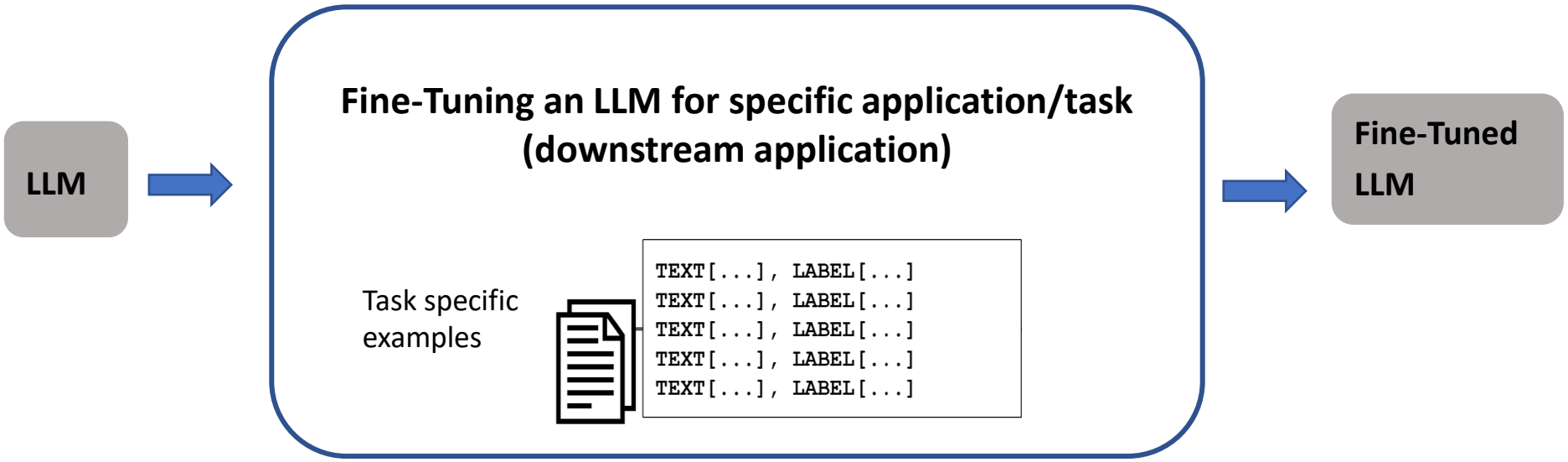
6. LLM Training stages at a high level



Finetuning a Large Language Model

- Use cases of Supervised Fine-Tuning
  - InstructLM
  - Domain-adaptation (ex. Finetuning Llama on legal data)
  - Toolformer (Teaching models to use tools/APIs)
  - Extend context length (ex. YaRN)
  - Fine-tune your model on your domain-specific data/specific application or task (i.e., **additional training to become better at a specific task**)

Fine-tune your model on your domain-specific data/specific application or task



- Example (Demo on my cloud fine-tuned LLM): Arabic Emotion recognizer

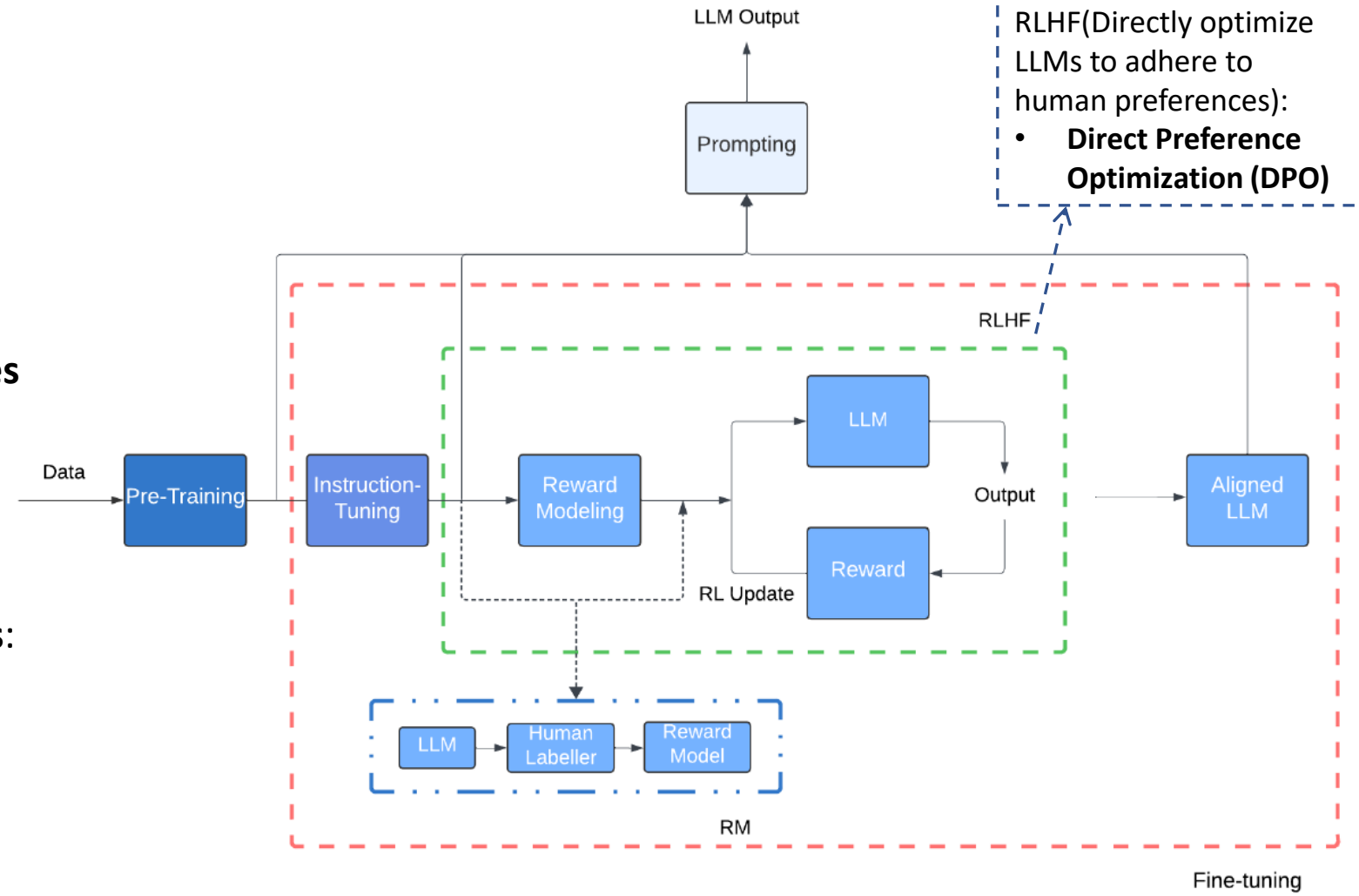
7. Training Stages and Types of LLMs

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**.

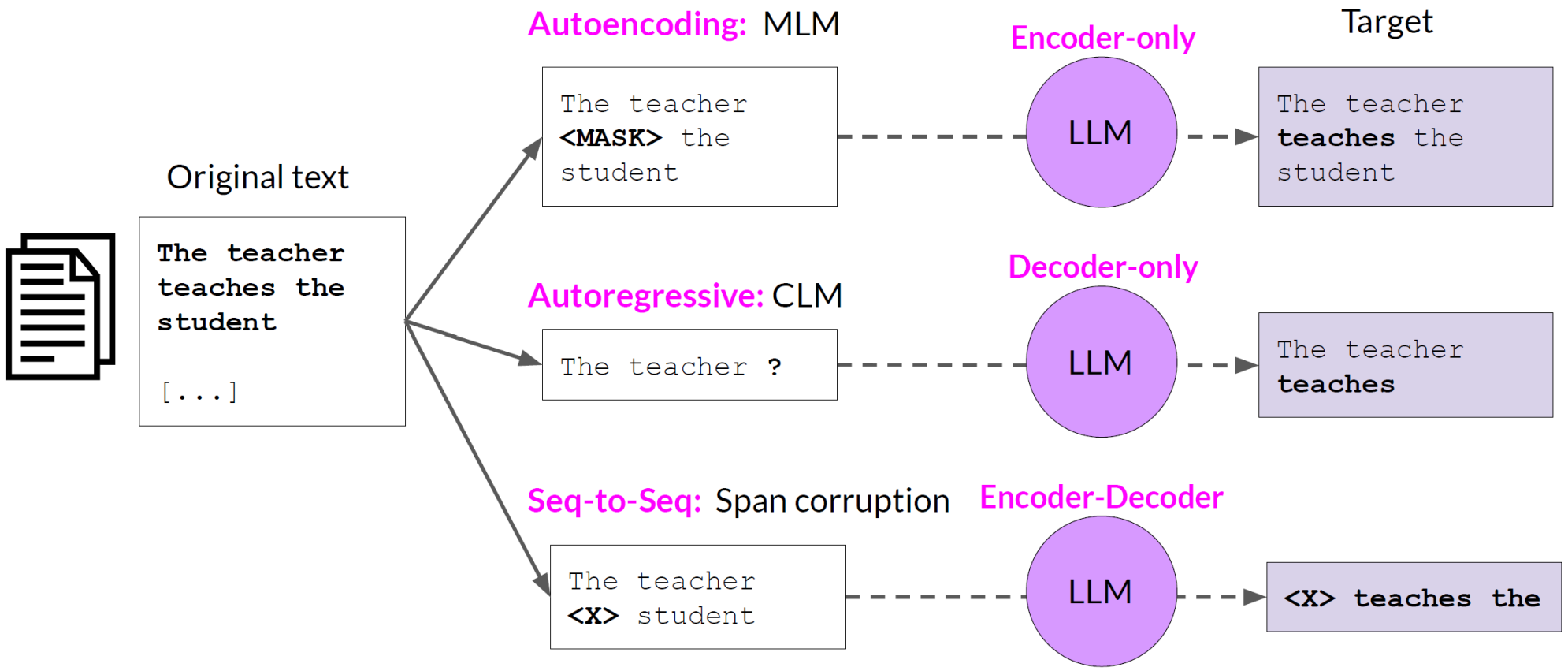
- **Fine-tuned LLMs** (**adapting pre-trained LLMs**: **instruction** or **alignment** fine-tuned model checkpoints, - also called chat models)
  - Example: representative alignment criteria (i.e., helpful, honest, and harmless)

- **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. Model architectures and pre-training objectives

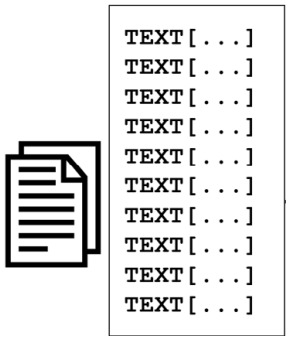


LLM Pre-training process at a high level (decoder-only architectures): Causal language modeling task (autoregressive models)

Un-trained LLM

- Training on **massive text data** (i.e., large-scale corpora).
  GPT-3 (2020): 300B tokens

Unlabeled data



- Pre-training task (i.e., pre-training objective):
  Language Modeling task => predict the next token (word)
- Example:

Python **is** **a** **programming** **?**

Model

Pre-trained LLM (base LLM)

- It encodes what it learns from training in its parameters (numbers).
- These numbers are used to calculate which token to generate at each run.
- It acquires essential language understanding and generation skills.
- Pre-trained model checkpoints obtained right after pre-training are called **Base LLMs.**

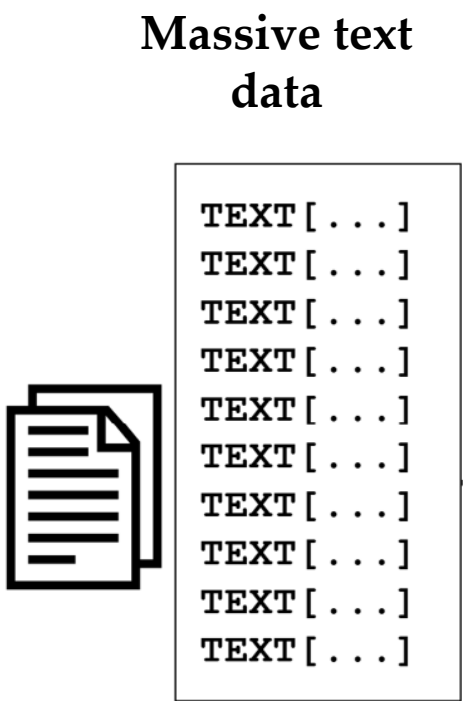
✓A base LLM acquires the **general abilities for solving various tasks**.

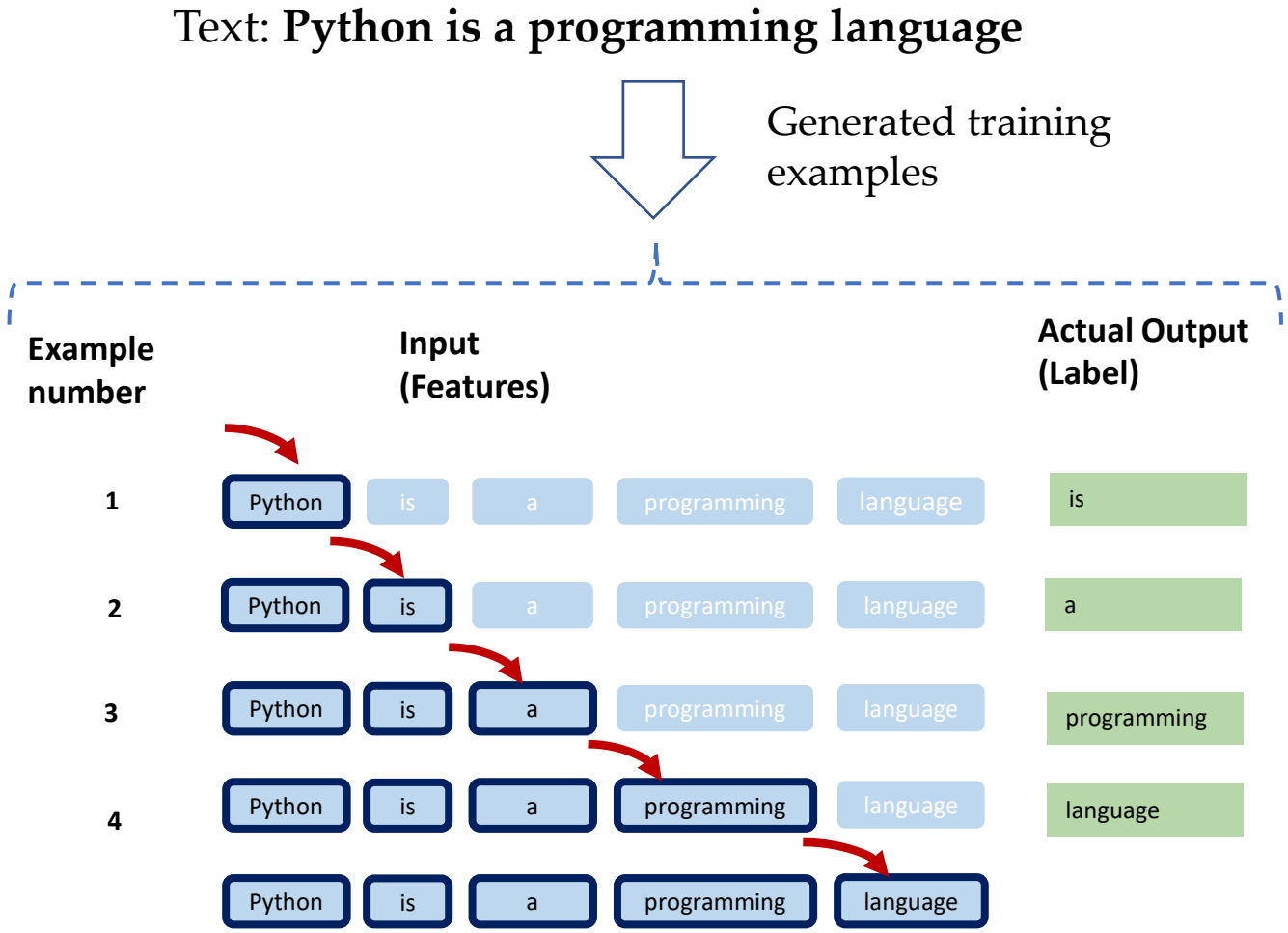
✓It lacks the consideration of **human values or preferences**.

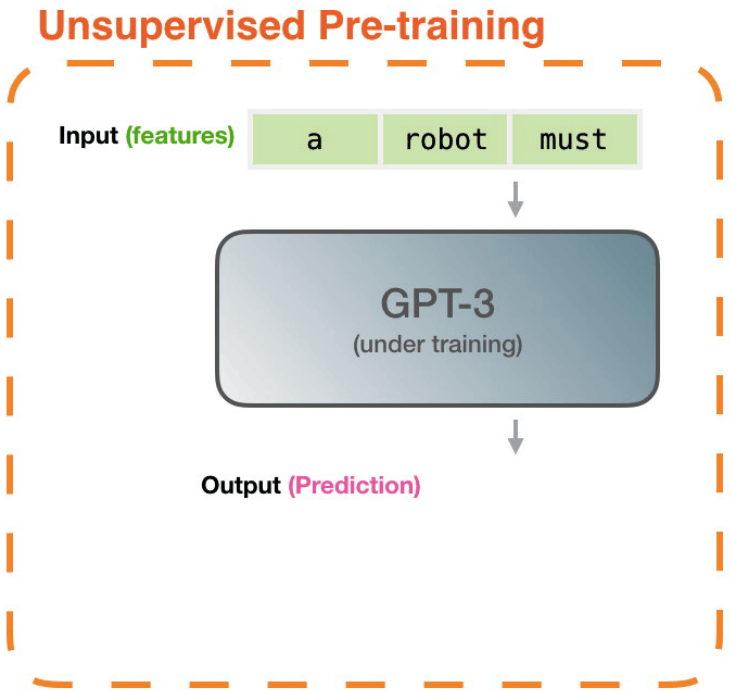
- Pre-training plays a key role that encodes **general knowledge from large-scale corpus** into the massive **model parameters**.
  - **Internal knowledge** (a.k.a. Parametric knowledge = knowledge stored in the LLM parameters (weights)): Learned during training that is implicitly stored in the neural network's weights.
- Pre-training establishes the basis of the abilities of LLMs. By pre-training on large-scale corpora, LLMs can acquire essential language understanding and generation skills.
- In this process, the **scale and quality of the pre-training corpus are critical** for LLMs to attain powerful capabilities.

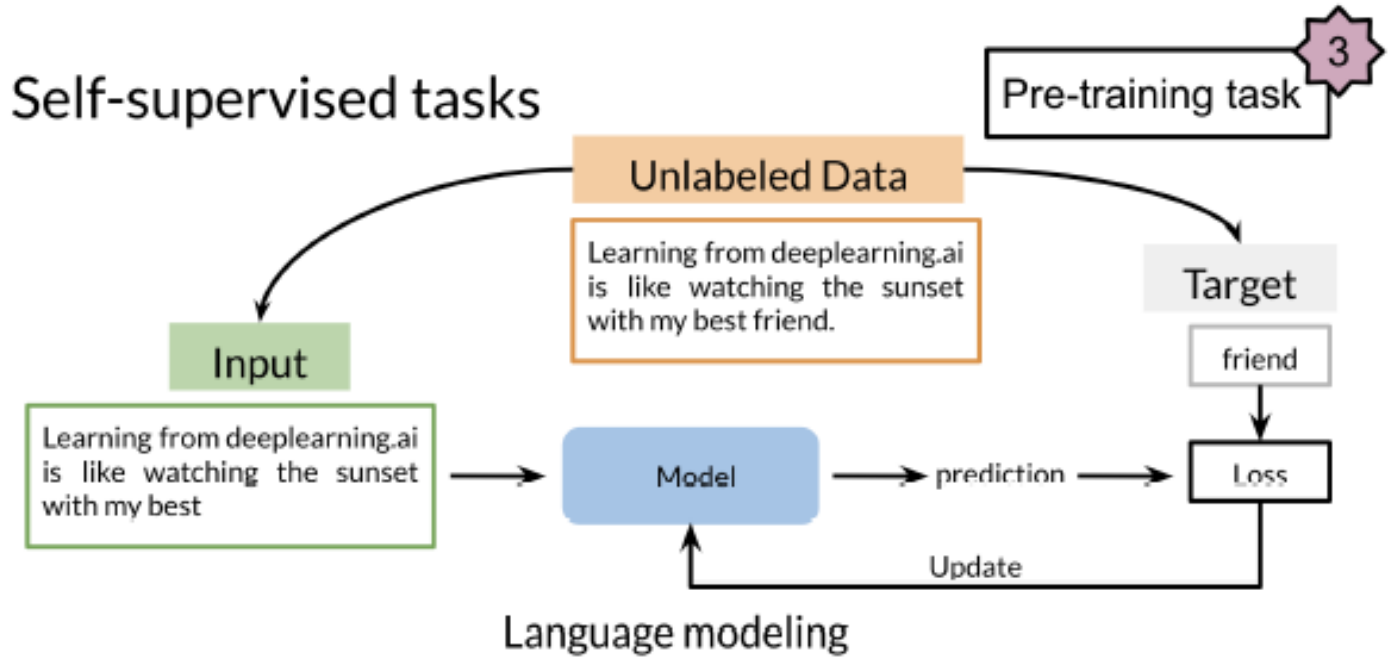
Generated training examples

- The text data is used to generate training examples for the model.
- For example, below are four training examples generated from the one sentence at the top. You can see how you can **slide a window** across all the text and make lots of examples.









[Source]

For example, in the drawing above we try to predict the word "friend". This allows your model to get a grasp of the overall structure of the data and to help the model learn some relationships within the words of a sentence.

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:

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

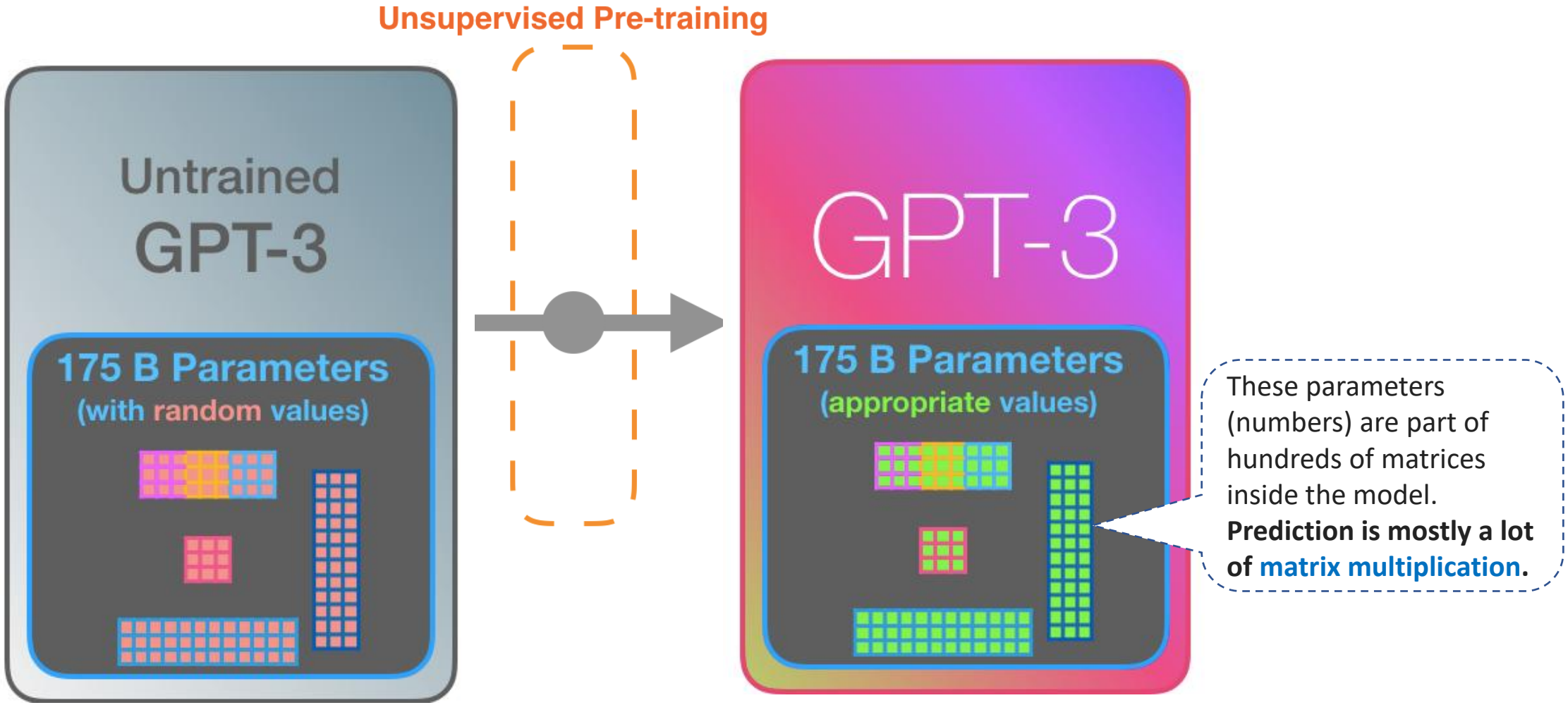
The LM task aims to autoregressively predict the target tokens x_i based on the **preceding** tokens $x_{<i}$ **in a sequence**. A general training objective is to maximize the following likelihood:

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

- ✓ 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**.

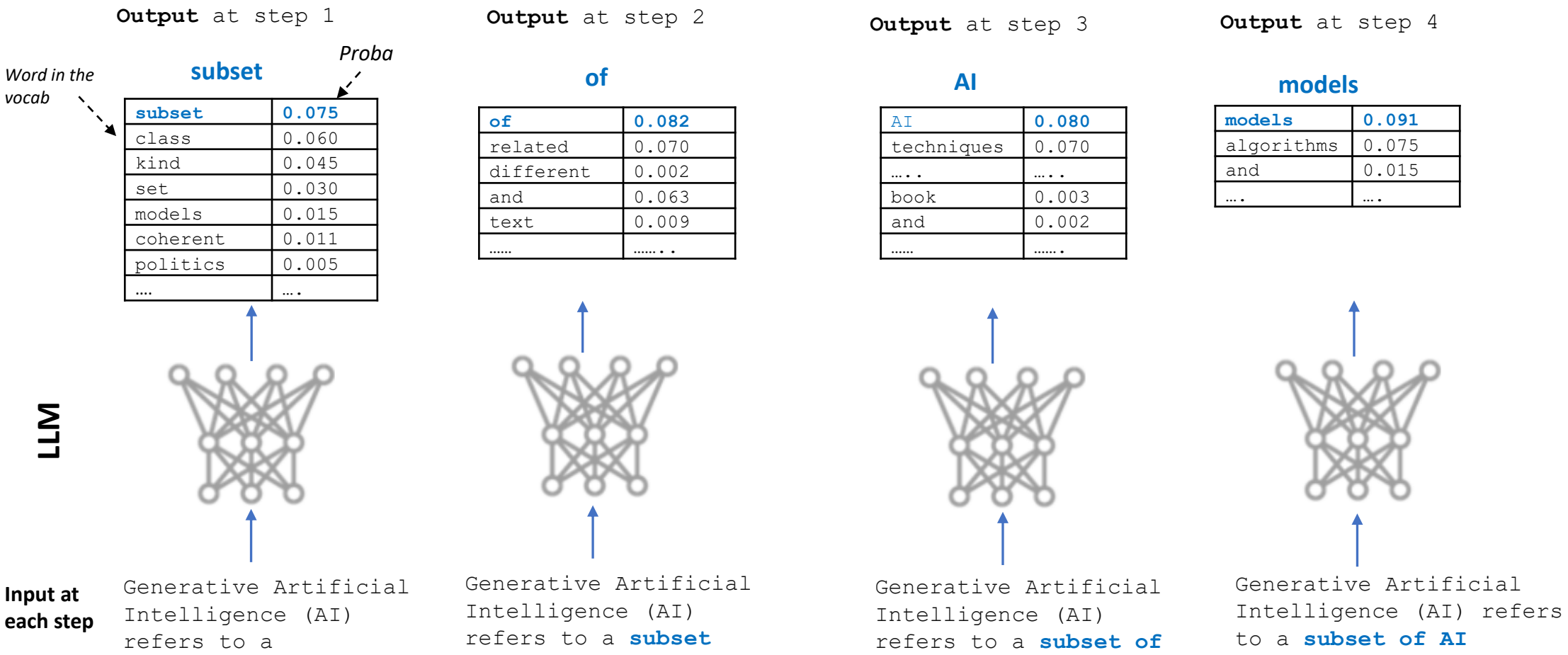
LLM Pre-training process: GPT-3

- GPT3 is an LLM. It encodes what it learns from training in **175 billion** parameters. These parameters (numbers) are used to calculate which token to generate at each run.
- The untrained model starts with random parameters. Training finds values that lead to better predictions.



Source: Alammar, J. How GPT3 Works - Visualizations and Animations [Blog post]

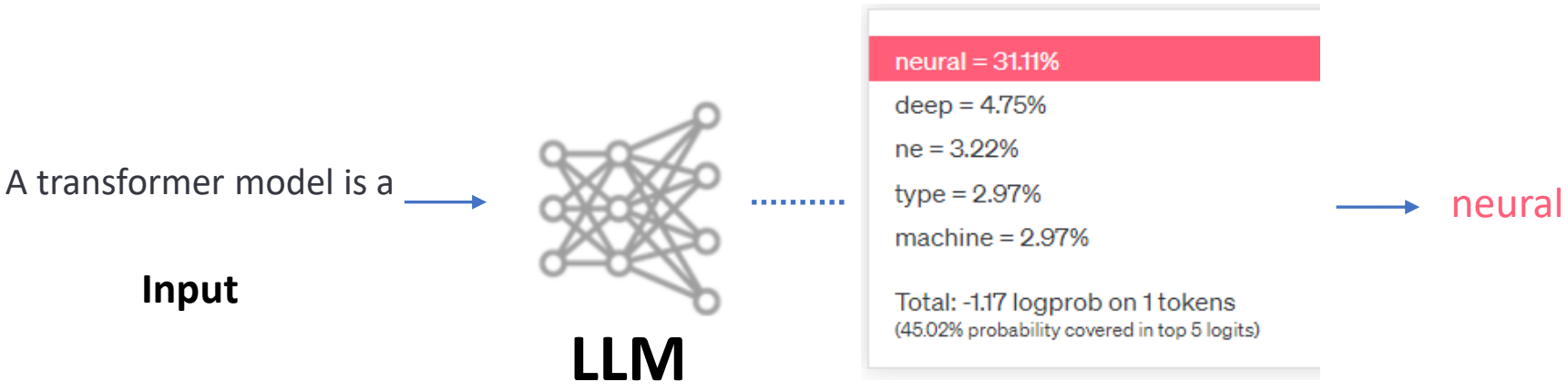
9. Text Generation:
General working flow of an LLM predicting the next word (autoregressive model)



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.

9. 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*

10. Decoding Strategy (Generative config)

10. Decoding Strategy (Generative config)

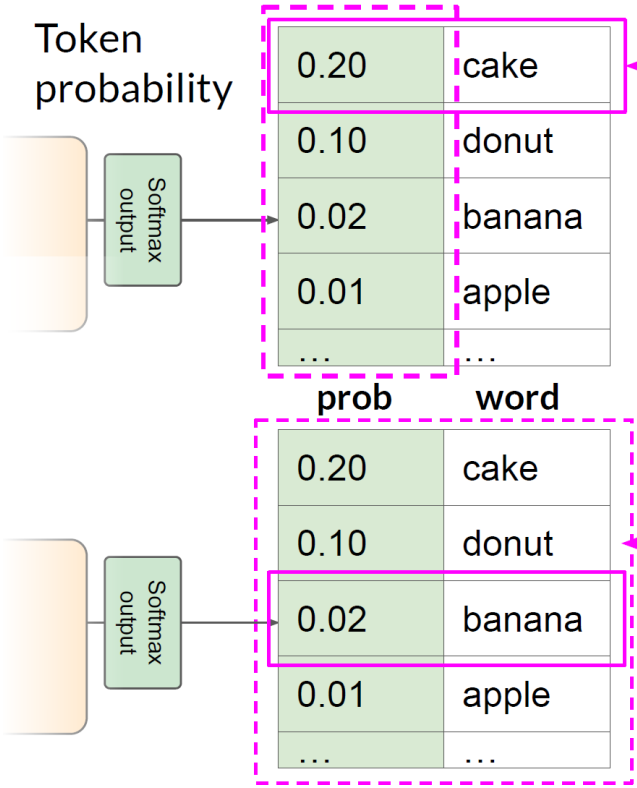
How to select the output tokens (a.k.a. **decoding strategy**)?

- An LLM outputs a probability distribution over the vocabulary.

- The next token is selected by either:
- Taking the token with **the largest probability**, or

- **Sampling from the output probability distribution**. This adds a bit of stochasticity into a model's performance. Given a prompt, it will not necessarily generate the same response twice.

Generative config - greedy vs. random sampling

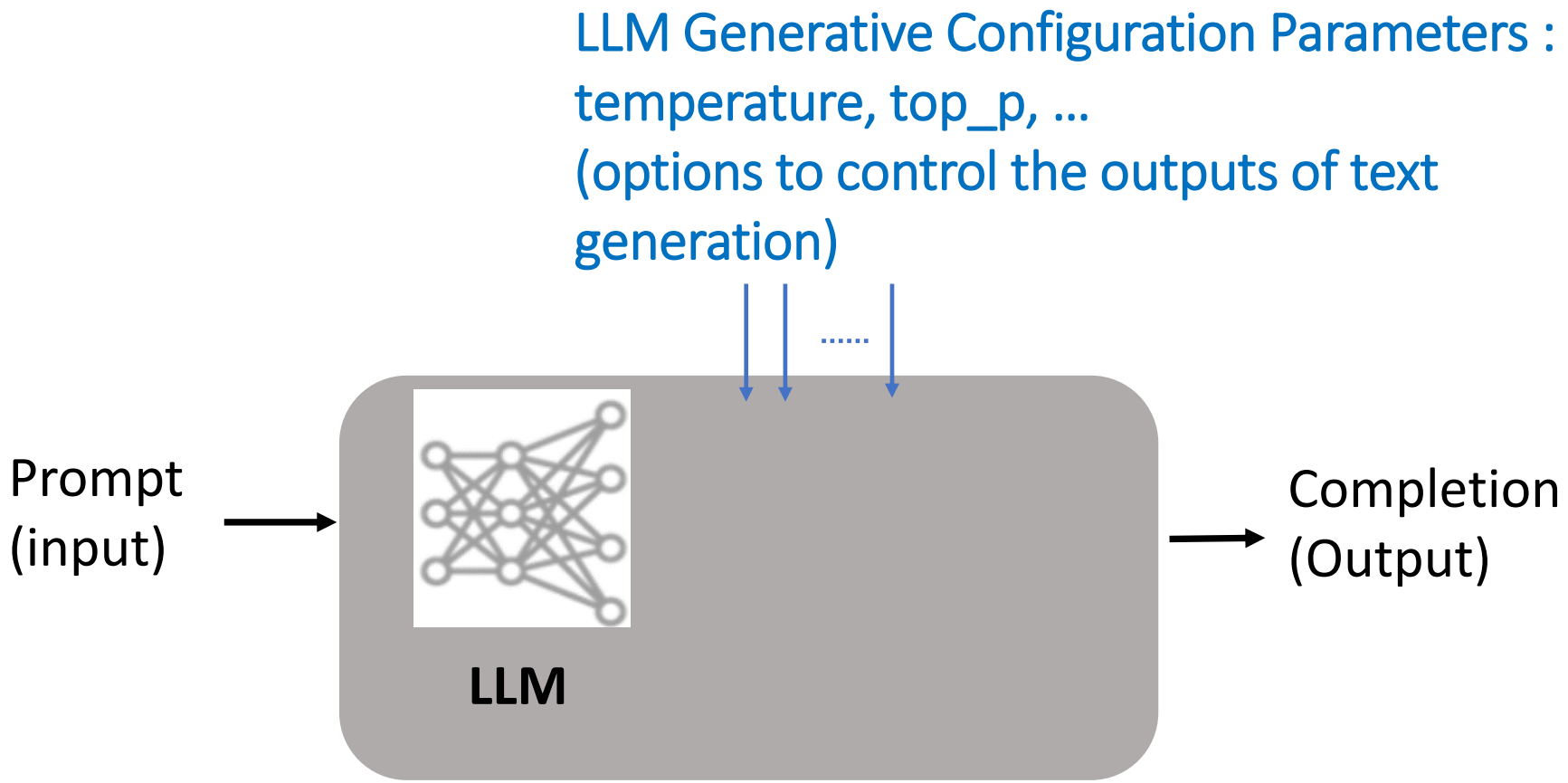


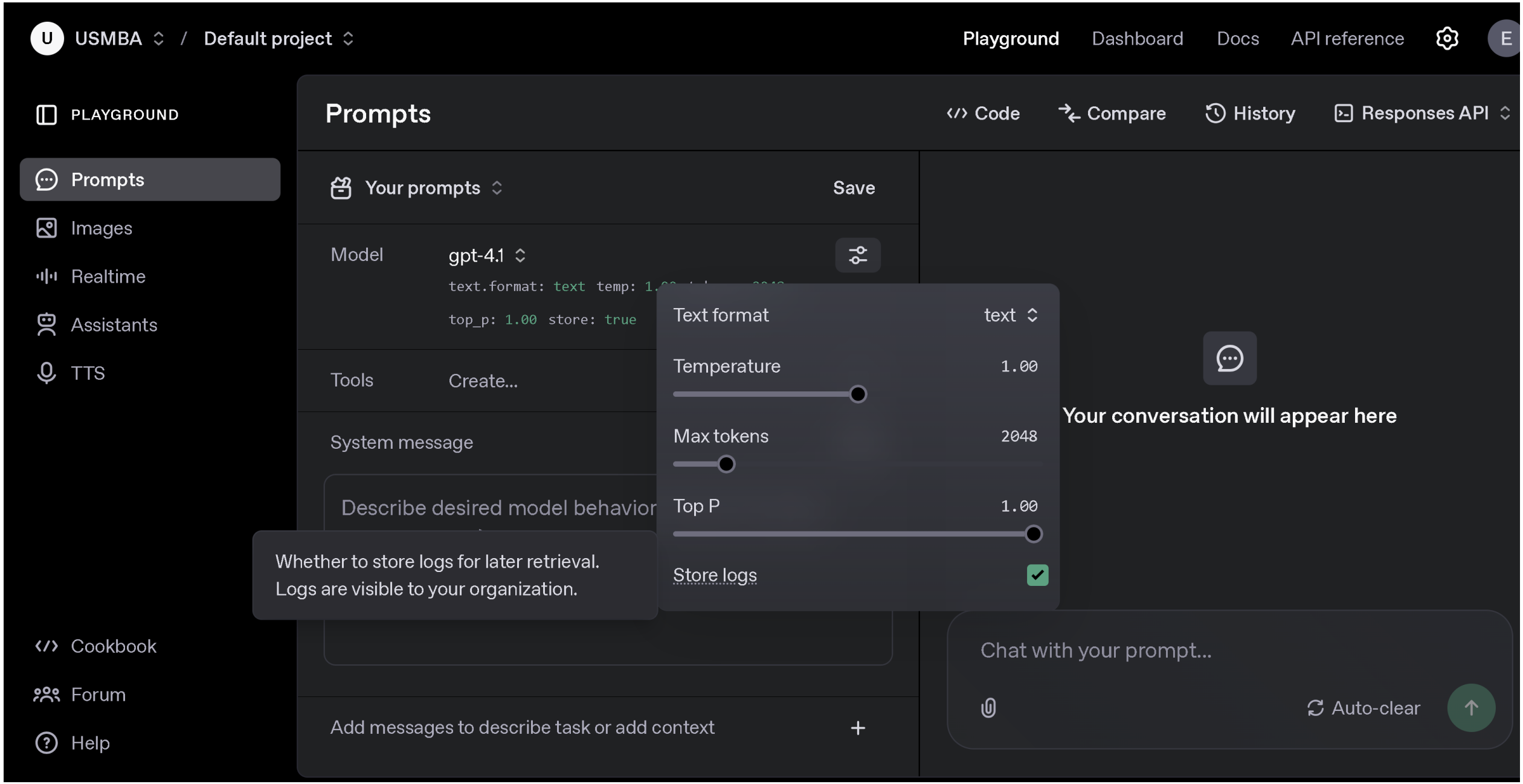
***greedy*:** The word/token with the highest probability is selected.

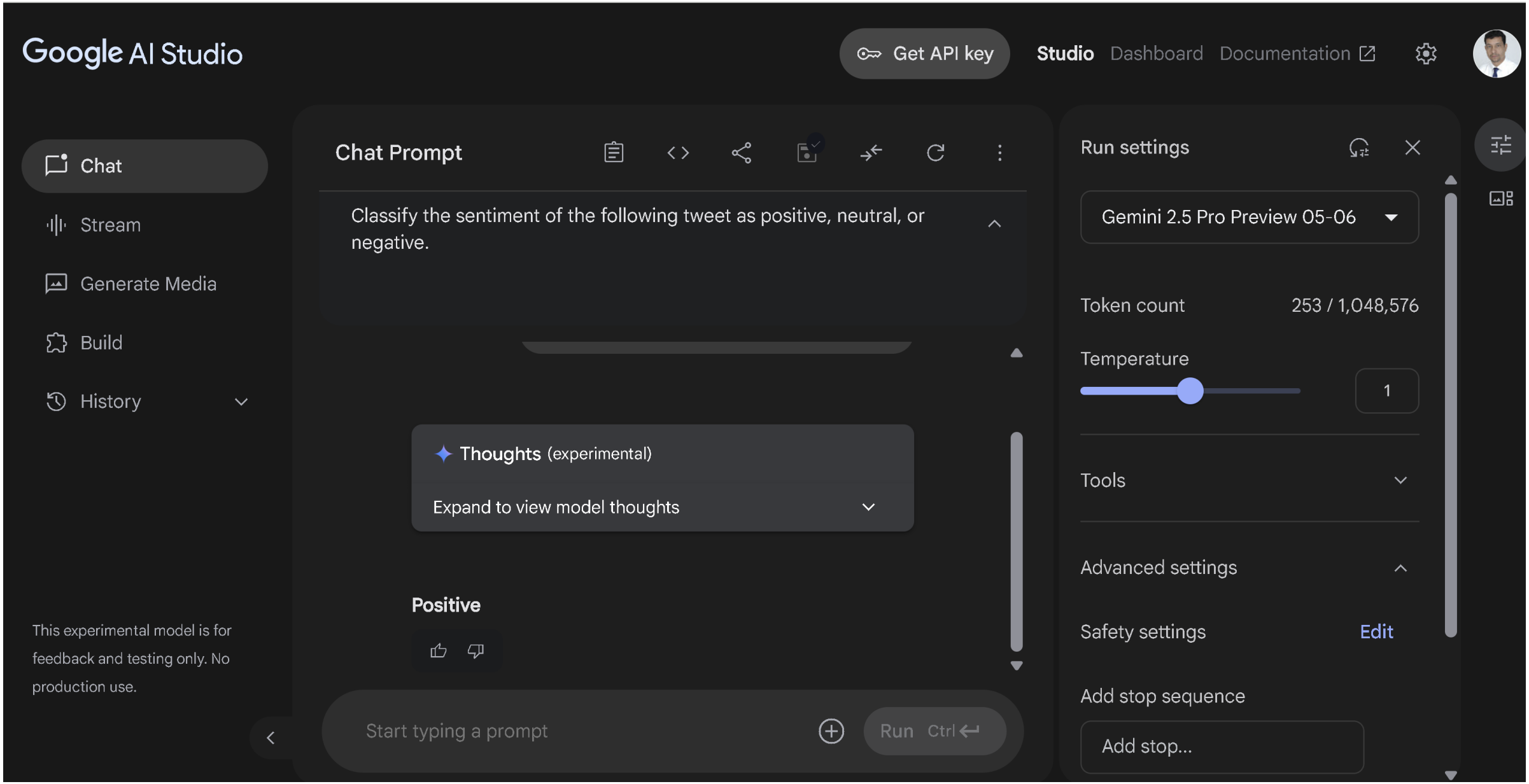
random(-weighted) sampling: select a token using a random-weighted strategy across the probabilities of all tokens.

Here, there is a 20% chance that 'cake' will be selected, but 'banana' was actually selected.

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




U
USMBA
/
Default project
Playground
Dashboard
Docs
API reference
E
PLAYGROUND
Prompts
Images
Realtime
Assistants
TTS
Cookbook
Forum
Help
Prompts
Code
Compare
History
Responses API
Your prompts
Save
Model
gpt-4.1
text.format: text temp: 1.00
top_p: 1.00 store: true
Tools
Create...
System message
Describe desired model behavior
Add messages to describe task or add context
Text format
text
Temperature
1.00
Max tokens
2048
Top P
1.00
Store logs
Whether to store logs for later retrieval.
Logs are visible to your organization.
Your conversation will appear here
Chat with your prompt...
Auto-clear


aistudio.google.com/prompts/1a7vSupYUDRsvN96LZ1P9g8JPPaOUgVUk
Google AI Studio
Get API key
Studio
Dashboard
Documentation
Chat
Stream
Generate Media
Build
History
This experimental model is for feedback and testing only. No production use.
Chat Prompt
Classify the sentiment of the following tweet as positive, neutral, or negative.
Thoughts (experimental)
Expand to view model thoughts
Positive
Start typing a prompt
Run Ctrl
Run settings
Gemini 2.5 Pro Preview 05-06
Token count
253 / 1,048,576
Temperature
1
Tools
Advanced settings
Safety settings
Edit
Add stop sequence
Add stop...

E. Nfaoui

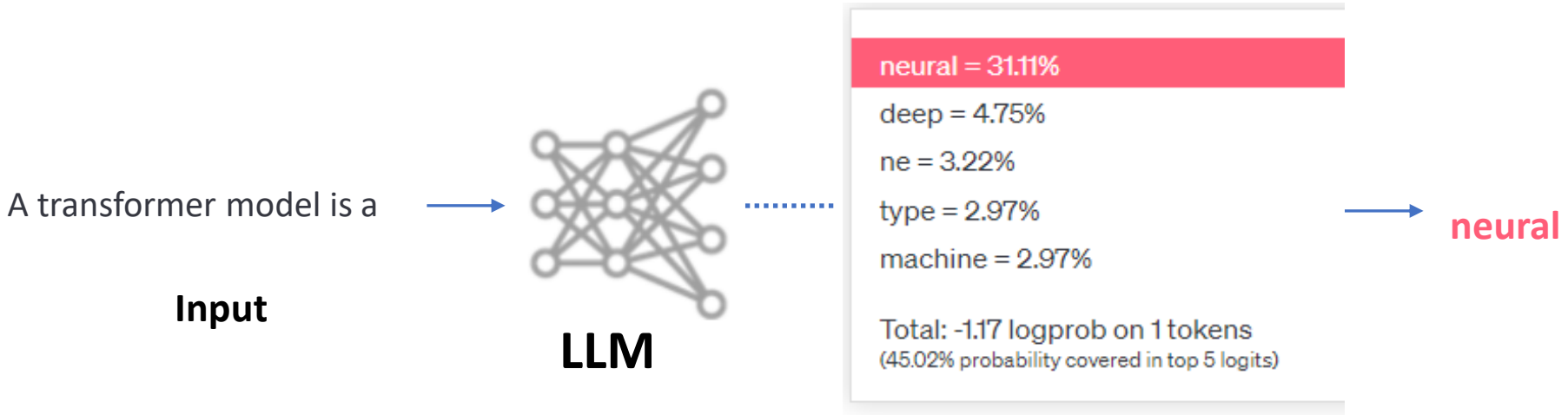
10. Decoding Strategy (Generative config)

✓ **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 $\mathbf{x}_{<i}$.

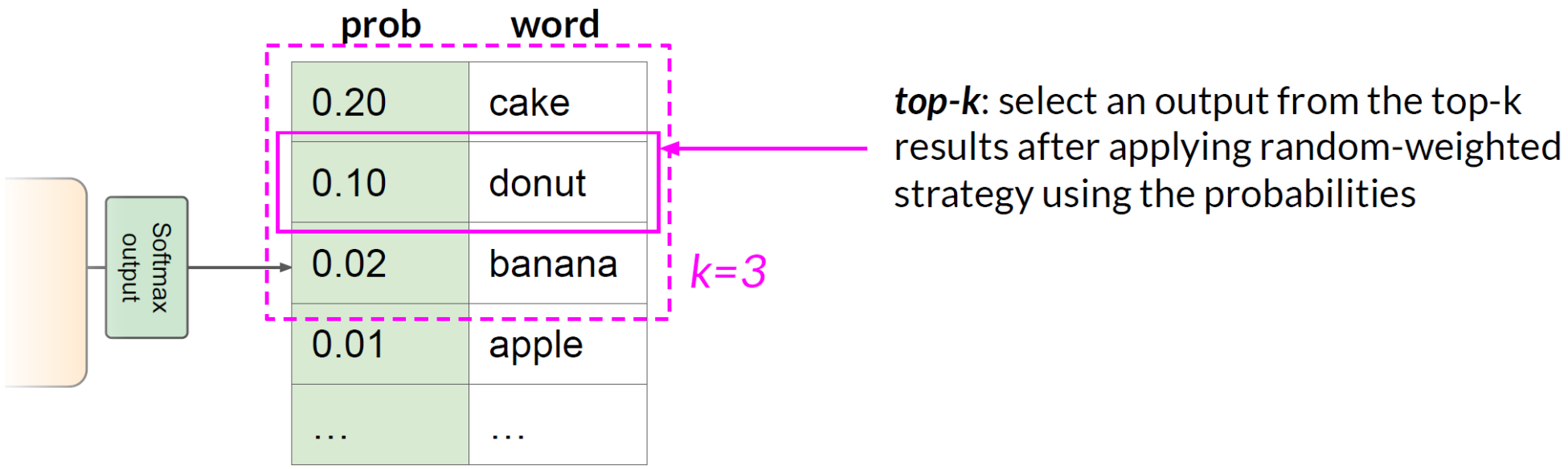


4. Decoding Strategy (Generative config)

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

- **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**.



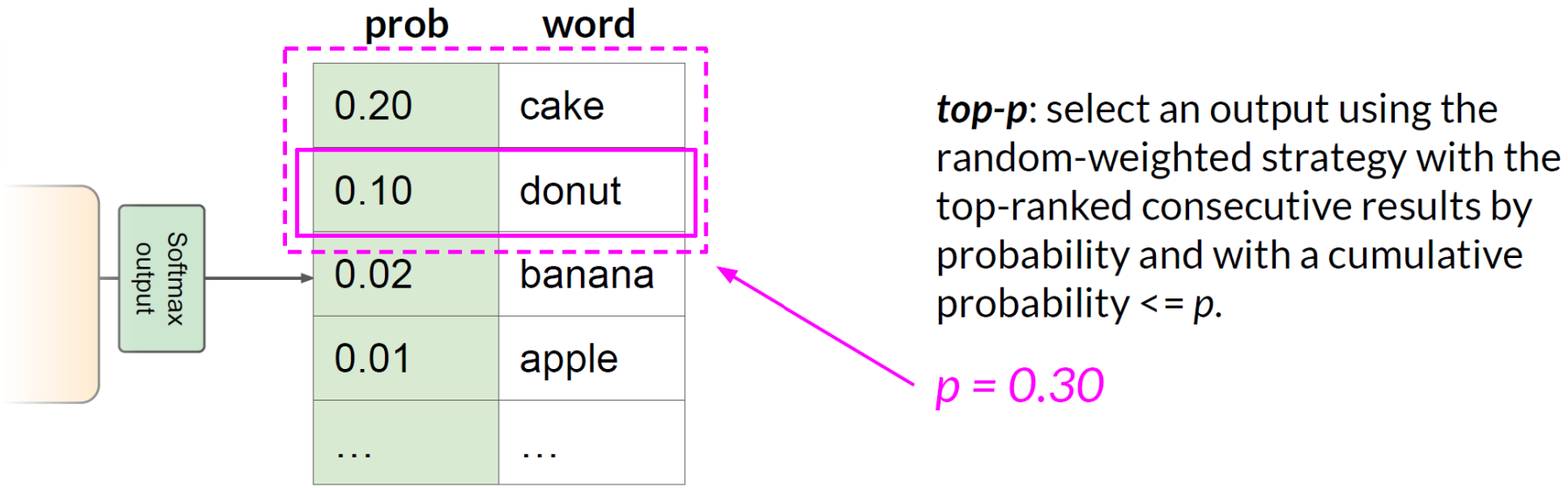
- top-k sampling does not consider the **overall possibility distribution**, a constant value of k may not be suitable for different contexts.

10. Decoding Strategy (Generative config)

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

- **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**.



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

10. Decoding Strategy (Generative config)

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

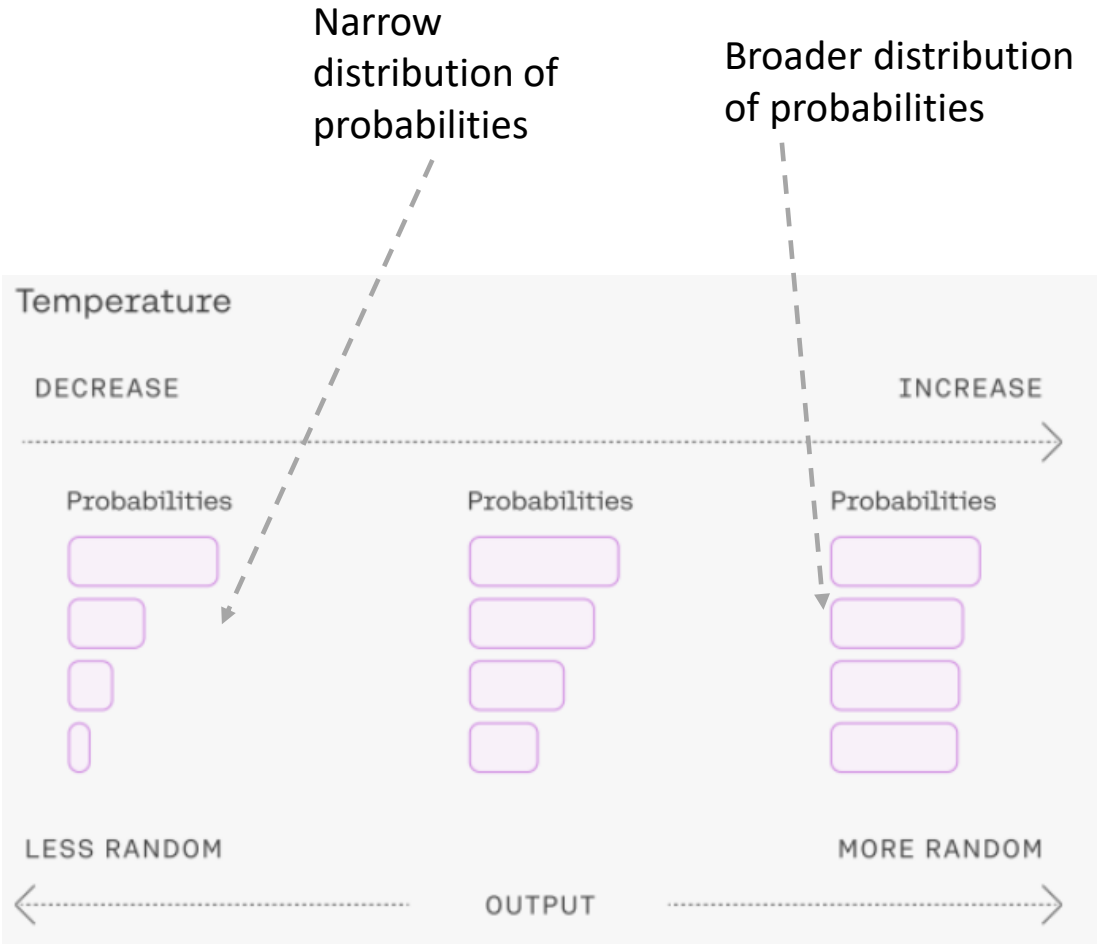
- **Temperature sampling:**

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.

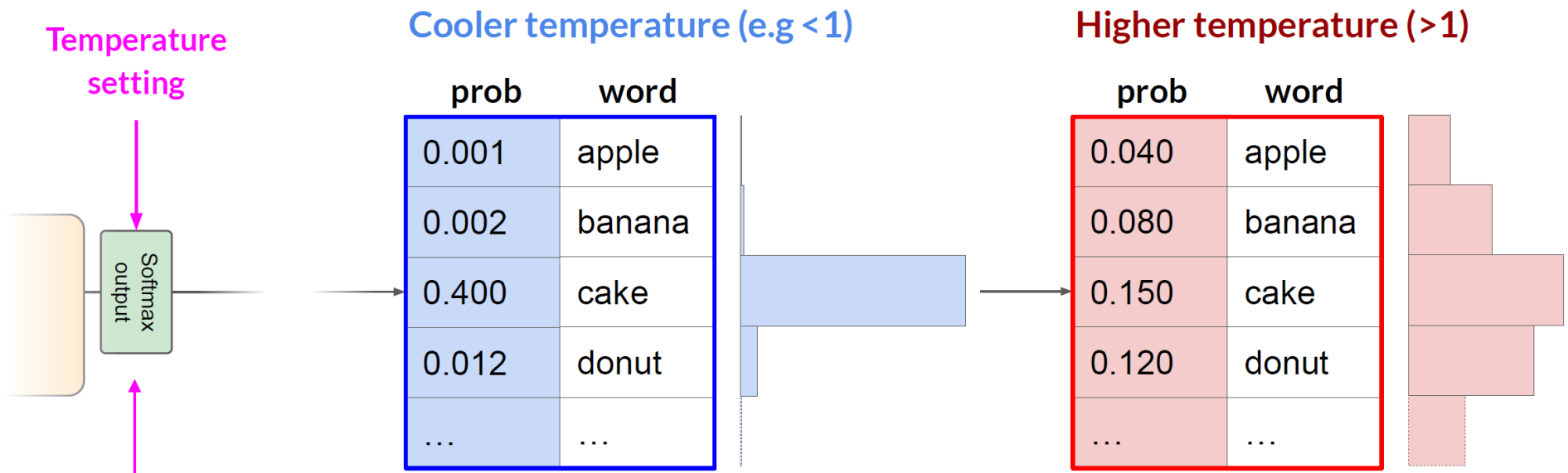
Temperature is a parameter that you can access in LLMs which essentially guides the model on how random their behaviour is, which means that the temperature influences the model's creativity.



cohere.ai

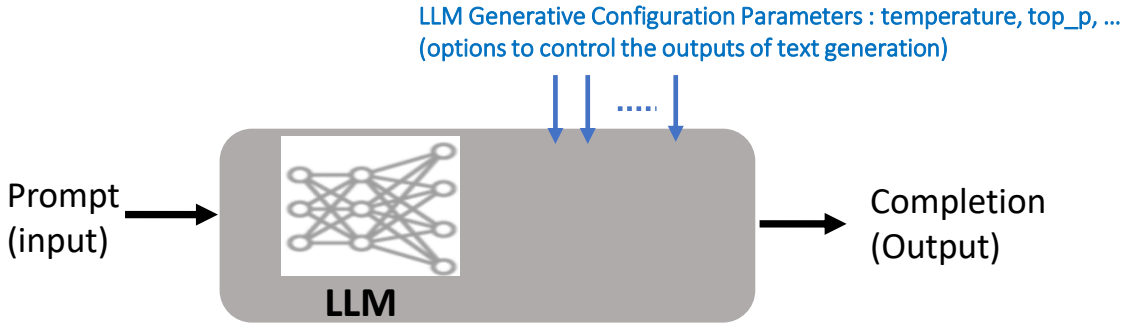
Demo (OpenAI Google Playground)

Sampling based methods): Temperature sampling



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

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.

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

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

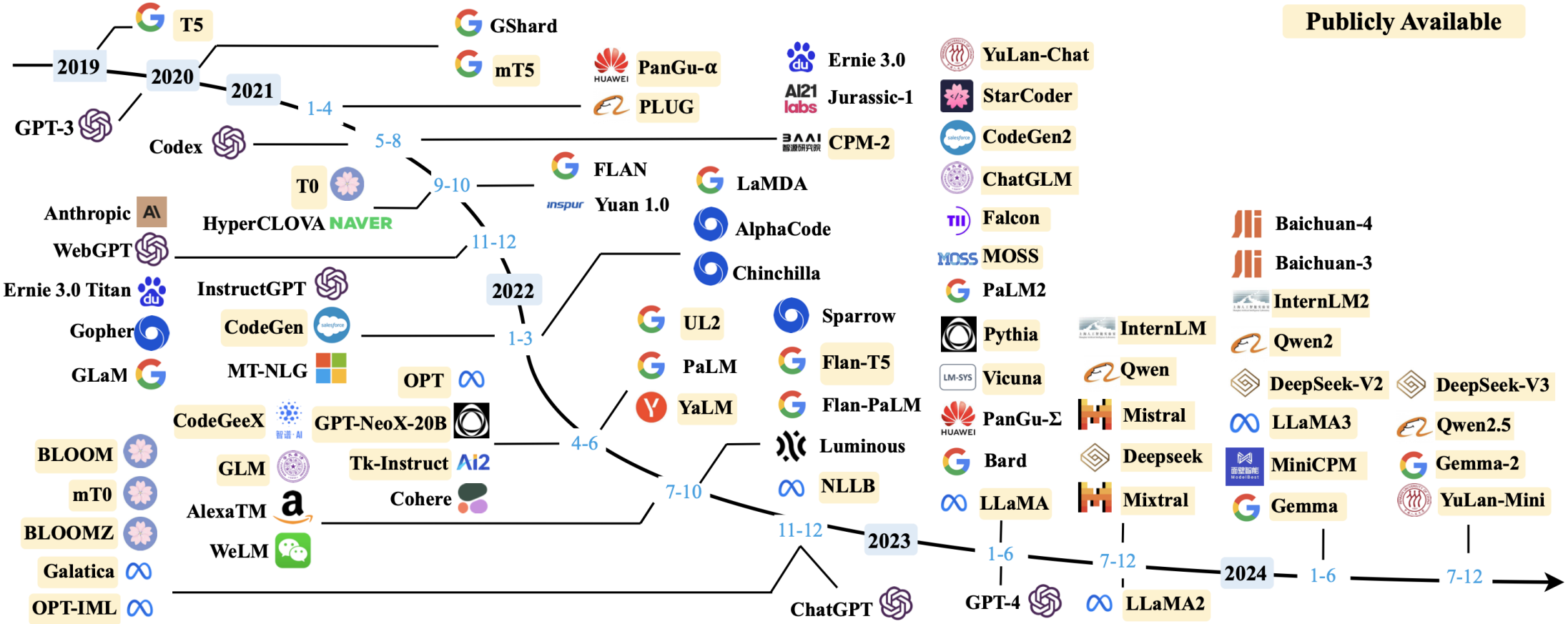


Fig. 3: A timeline of existing large language models (having a size larger than 10B) in recent years. The timeline was established mainly according to the release date (*e.g.*, the submission date to arXiv) of the technical paper for a model. If there was no corresponding paper, we set the date of a 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., 2025)*

Falcon-Arabic: A Breakthrough in Arabic Language Models

Community Article Published May 21, 2025

basma-b
Basma Boussaha
tiiuae

Alyafeai
Mohammed Alyafeai
tiiuae

amztheory
Ahmed Alzubaidi
tiiuae

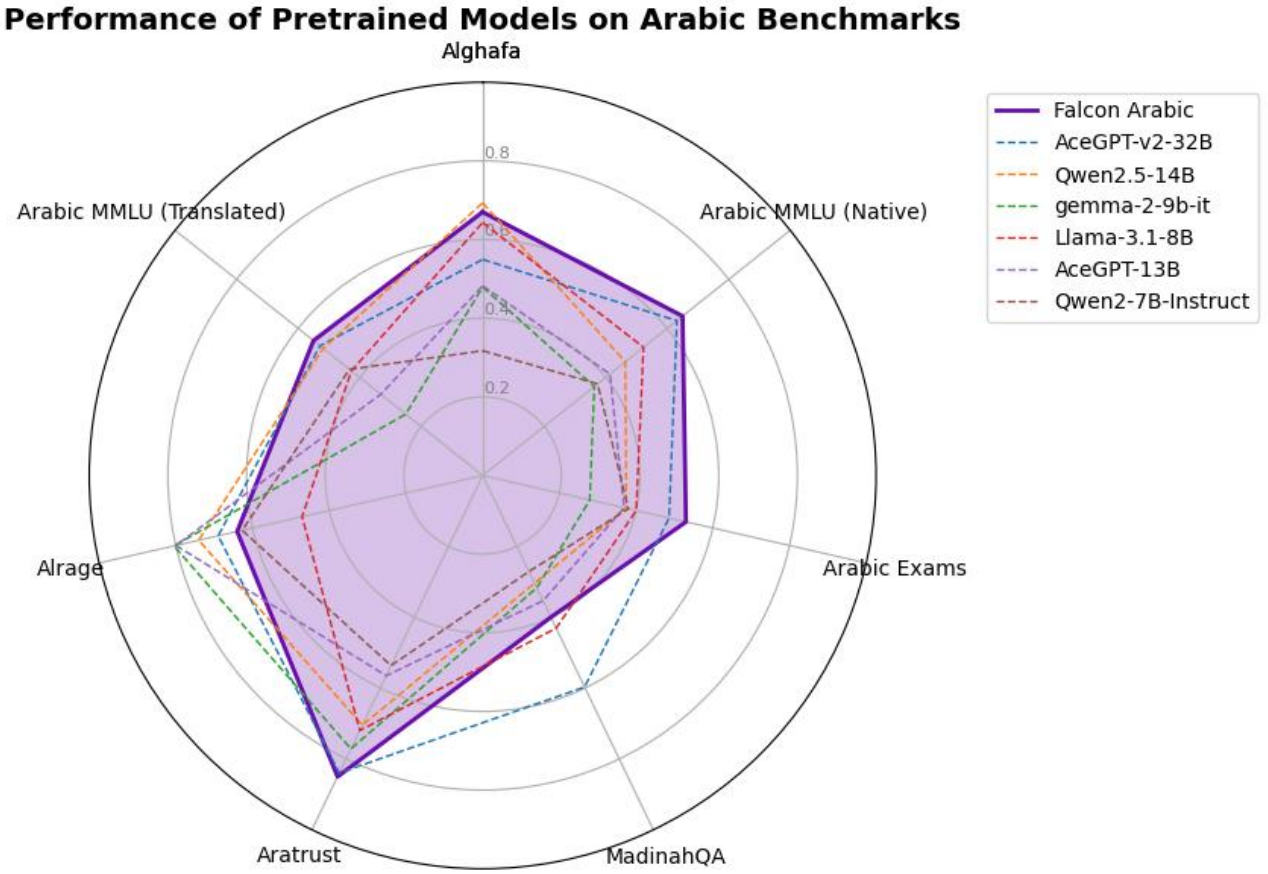
LeenAlQadi
Leen AlQadi
tiiuae

ybelkada
Younes B
tiiuae

mrsinvald
Mike Lubinets
tiiuae

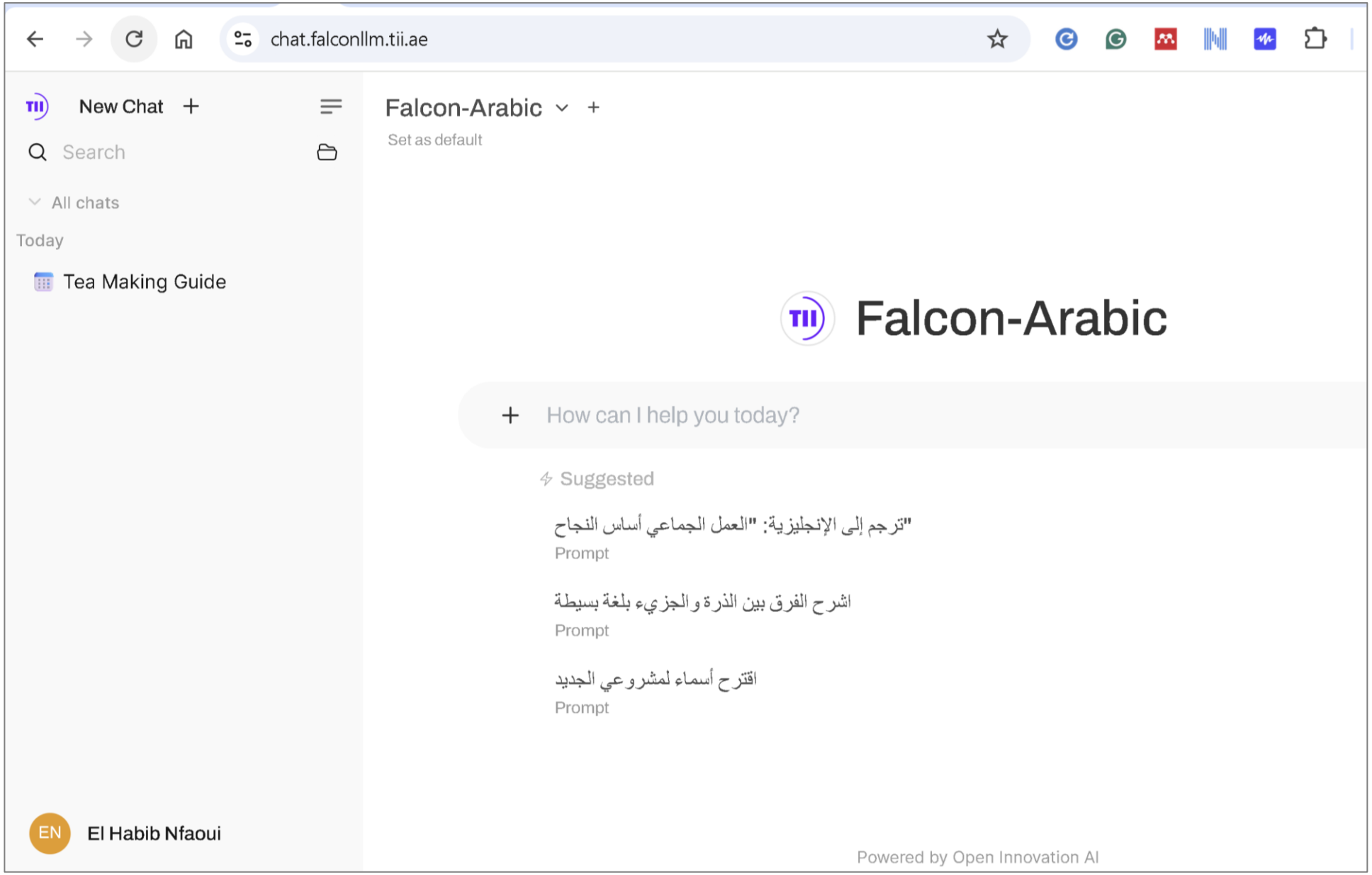
HakimHacid
Hakim Hacid
tiiuae

FalconLLM
Falcon LLM TII UAE
tiiuae



We are excited to introduce **Falcon-Arabic**, a 7B parameter Language Model that sets a new benchmark for Arabic NLP. Built on the Falcon 3 architecture, Falcon-Arabic is a multilingual model that supports Arabic, English, and several other languages. It excels in general knowledge, Arabic grammar, mathematical reasoning, complex problem solving, and understanding the rich diversity of Arabic dialects. Falcon-Arabic supports a context length of 32,000 tokens, allowing it to handle long documents and enabling advanced applications like retrieval-augmented generation (RAG), in-depth content creation, and knowledge-intensive tasks.

E. Nfaoui


chat.falconllm.tii.ae
New Chat
Search
All chats
Today
Tea Making Guide
EN
El Habib Nfaoui
Falcon-Arabic
Set as default
Falcon-Arabic
How can I help you today?
Suggested
"ترجم إلى الإنجليزية: "العمل الجماعي أساس النجاح
Prompt
اشرح الفرق بين الذرة والجزيء بلغة بسيطة
Prompt
اقترح أسماء لمشروعي الجديد
Prompt
Powered by Open Innovation AI

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

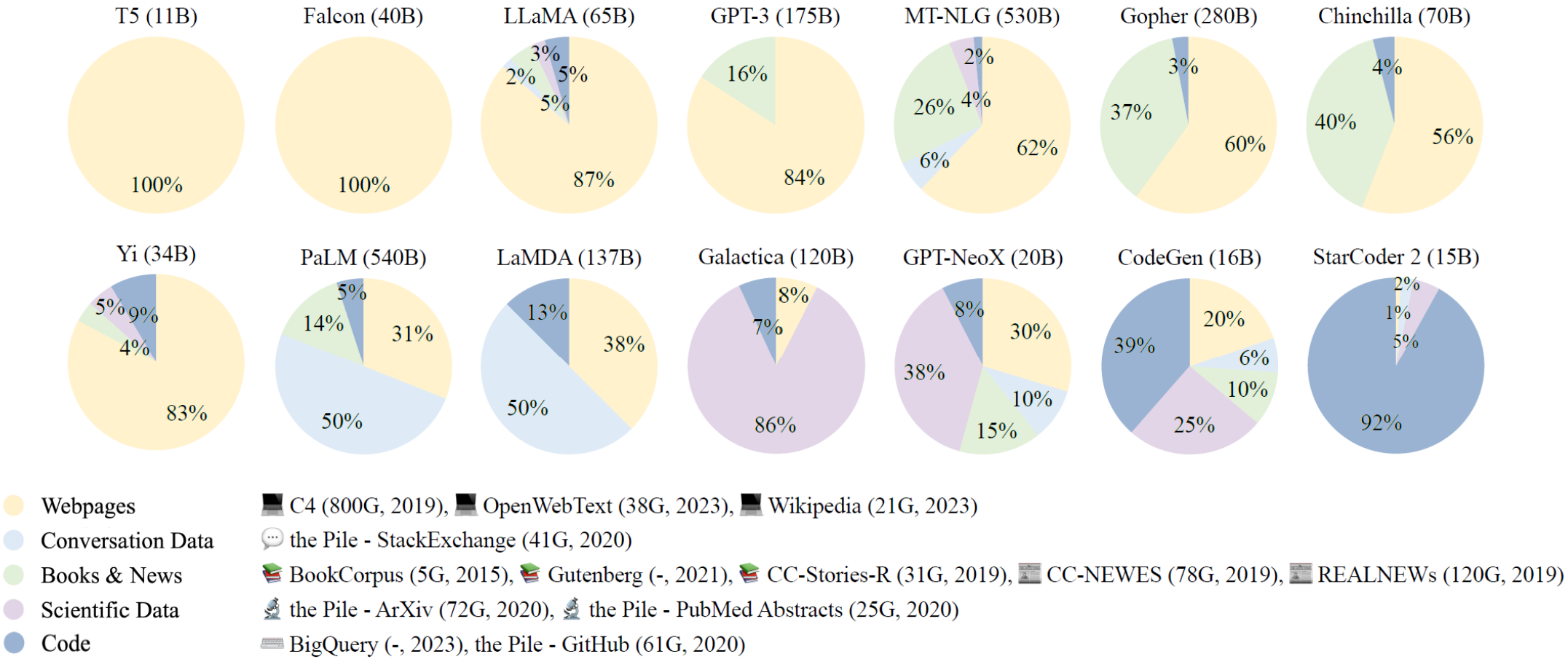


Fig. 6: Ratios of various data sources in the pre-training data for existing LLMs.

Source: Wayne Xin Zhao et al., 2023 (v16 , Mar 2025)

🤗 Open LLM Leaderboard

📐 The 🤗 Open LLM Leaderboard aims to track, rank and evaluate open LLMs and chatbots.

🤗 Submit a model for automated evaluation on the 🤗 GPU cluster on the "Submit" page! The leaderboard's backend runs the great Eleuther AI Language Model Evaluation Harness - read more details in the "About" page!

🏅 LLM Benchmark | 📈 Metrics through time | 📝 About | 🚀 Submit here!

🔍 Search for your model (separate multiple queries with `;`) and press ENTER...

Select columns to show

- [x] Average ⬆️
- [x] ARC
- [x] HellaSwag
- [x] MMLU
- [x] TruthfulQA
- [x] Winogrande
- [x] GSM8K
- [x] DROP
- [ ] Type
- [ ] Architecture
- [ ] Precision
- [ ] Hub License
- [ ] #Params (B)
- [ ] Hub ❤️
- [ ] Available on the hub
- [ ] Model sha

- [ ] Show gated/private/deleted models

Model types

- [x] 🟢 pretrained
- [x] 🔶 fine-tuned
- [x] ⭕ instruction-tuned
- [x] 🟦 RL-tuned
- [x] ?

Precision

- [x] float16
- [x] bfloat16
- [x] 8bit
- [x] 4bit
- [x] GPTQ
- [x] ?

Model sizes (in billions of parameters)

- [x] ?
- [x] ~1.5
- [x] ~3
- [x] ~7
- [x] ~13
- [x] ~35
- [x] ~60
- [x] 70+

T	Model	Average ⬆️	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K	DROP
🔶	TigerResearch/tigerbot-70b-chat-v2	69.76	87.03	82.83	66	75.4	79.16	46.02	51.9
⭕	bhenrym14/platypus-yi-34b	68.96	68.43	85.21	78.13	54.48	84.06	47.84	64.55
🟢	01-ai/Yi-34B	68.68	64.59	85.69	76.35	56.23	83.03	50.64	64.2
🟢	chargoddard/Yi-34B-Llama	68.4	64.59	85.63	76.31	55.6	82.79	49.51	64.37
⭕	MayaPH/GodziLLa2-70B	67.01	71.42	87.53	69.88	61.54	83.19	43.21	52.31
🔶	sequelbox/StellarBright	66.98	72.95	87.82	71.17	64.46	83.27	39.5	49.66
🔶	garage-bAInd/Platypus2-70B-instruct	66.89	71.84	87.94	70.48	62.26	82.72	40.56	52.41
⭕	upstage/SOLAR-0-70b-16bit	66.88	71.08	87.89	70.58	62.25	83.58	45.26	47.49
🔶	Sao10K/Euryale-1.3-L2-70B	66.58	70.82	87.92	70.39	59.85	82.79	34.19	60.1
🔶	psmathur/model_101	66.55	68.69	86.42	69.92	58.85	82.08	44.81	55.1

E. Nfaoui

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

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

- **Language translation**
- **Question answering**
- **Text summarization**
- **Content creation (generation)**
- **Code generation**
- **Sentiment analysis**
- **Chatbots**
- **Summarization, Essay writing**
- **Entity extraction**
- Etc.,

SoluLab

- ✓ 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**.

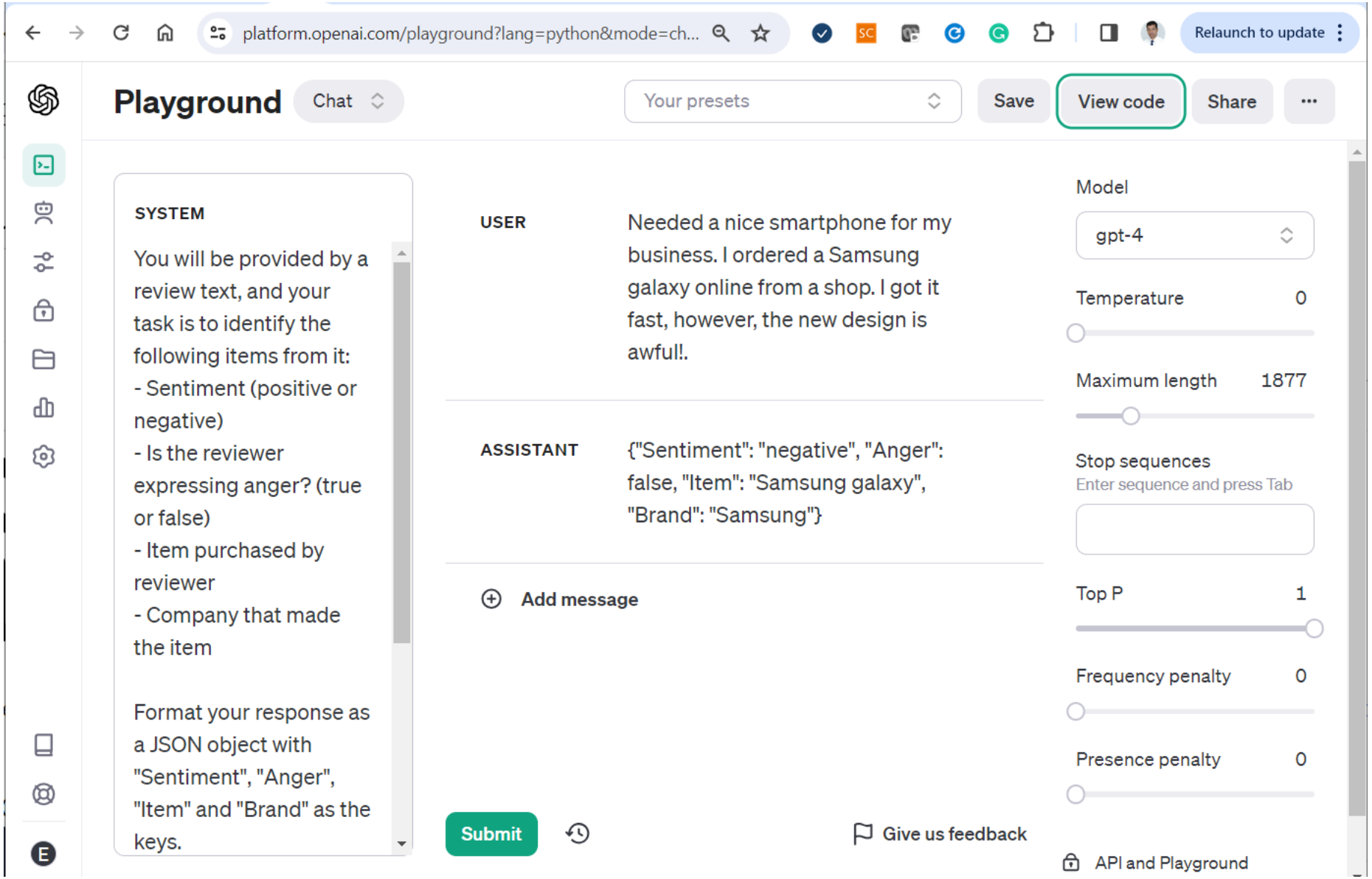
Demo

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

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

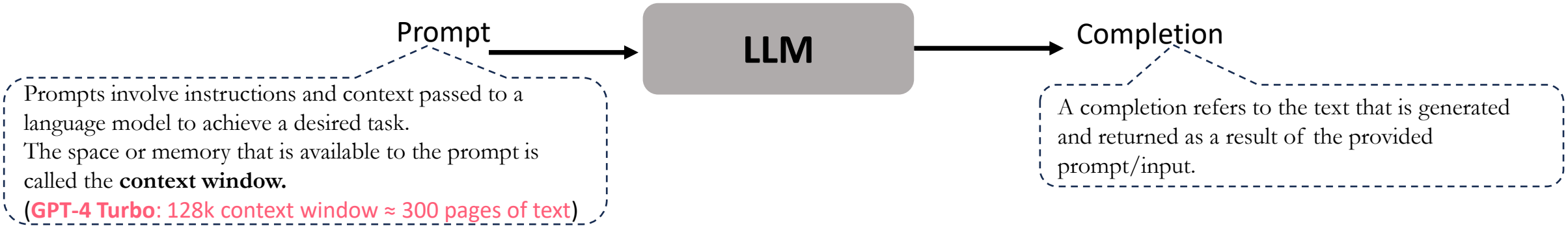
In this example, the LLM performs 3 tasks at once:

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



LLMs utilization:
How to work with large language models?

Prompt, completion, and prompt engineering



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."

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

Prompting techniques

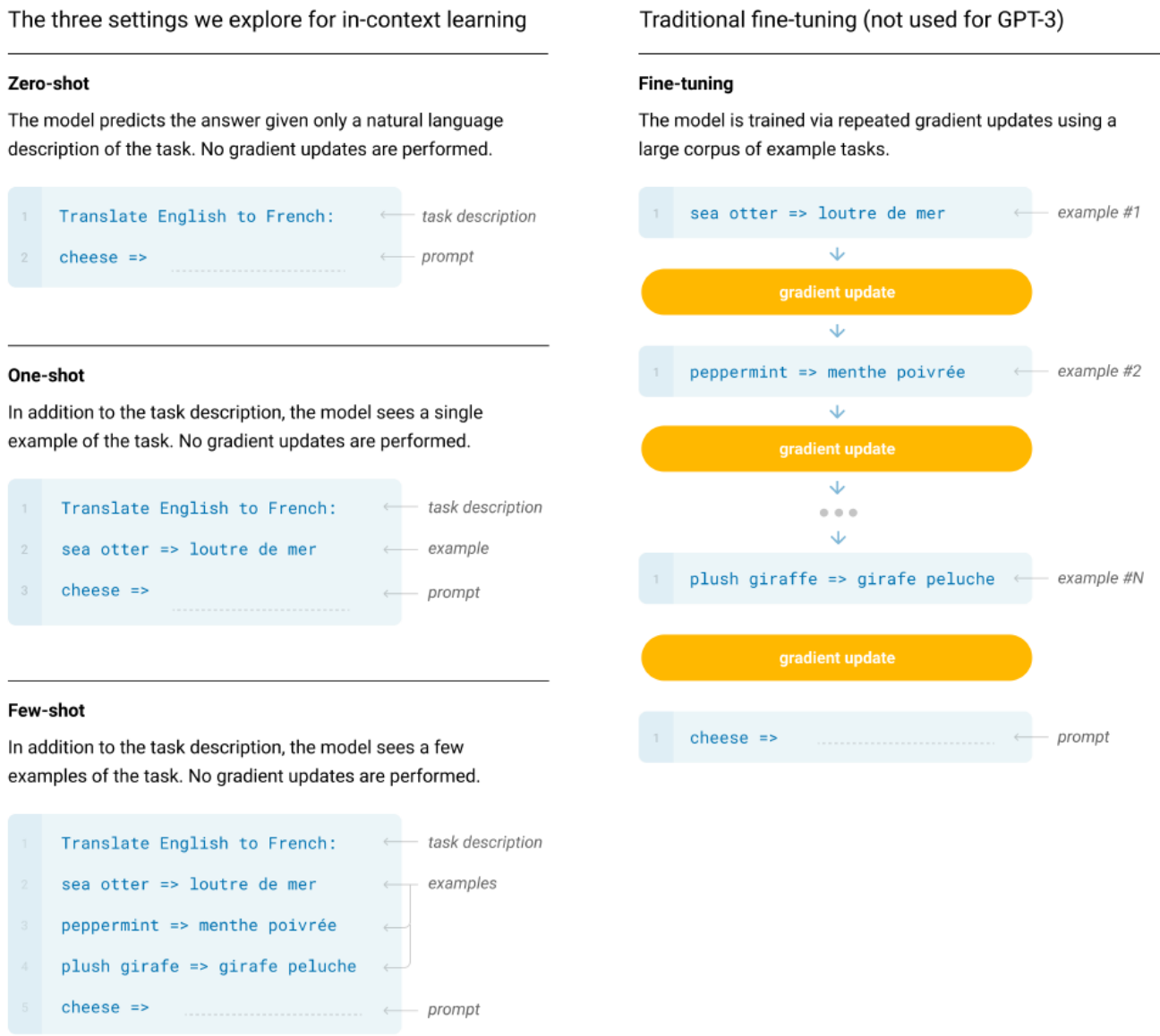
Large language models can be prompted to produce output using various prompting techniques, including:

- Zero-shot prompt
- One-shot prompt
- Few-shot prompt
- CoT prompt
- ReAct prompt
- Emotional prompt (EmotionPrompt)
- Etc.

In-context learning (Zero-shot prompt, One-shot prompt, Few-shot prompt):

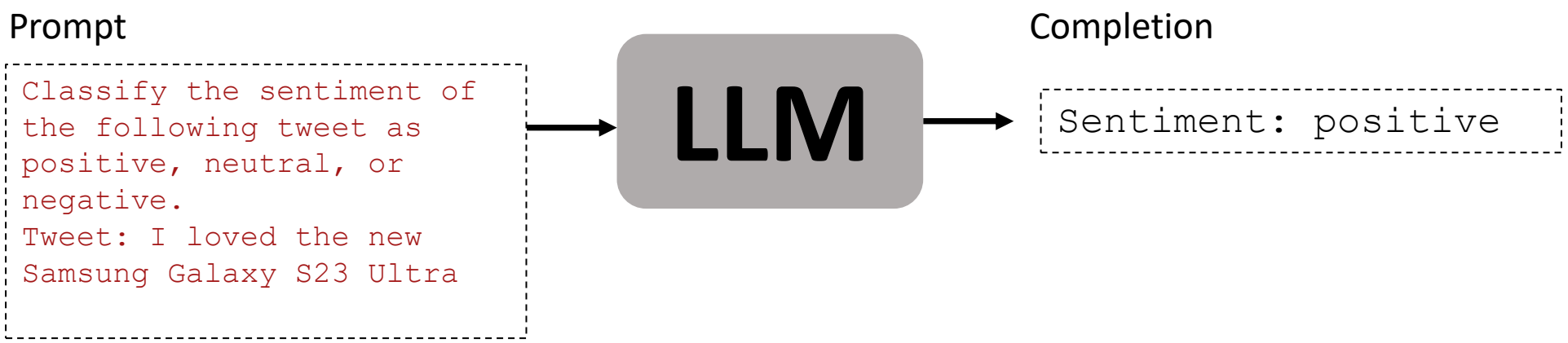
A typical prompting method which formulates the task description and/or demonstrations in the form of natural language text.

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)

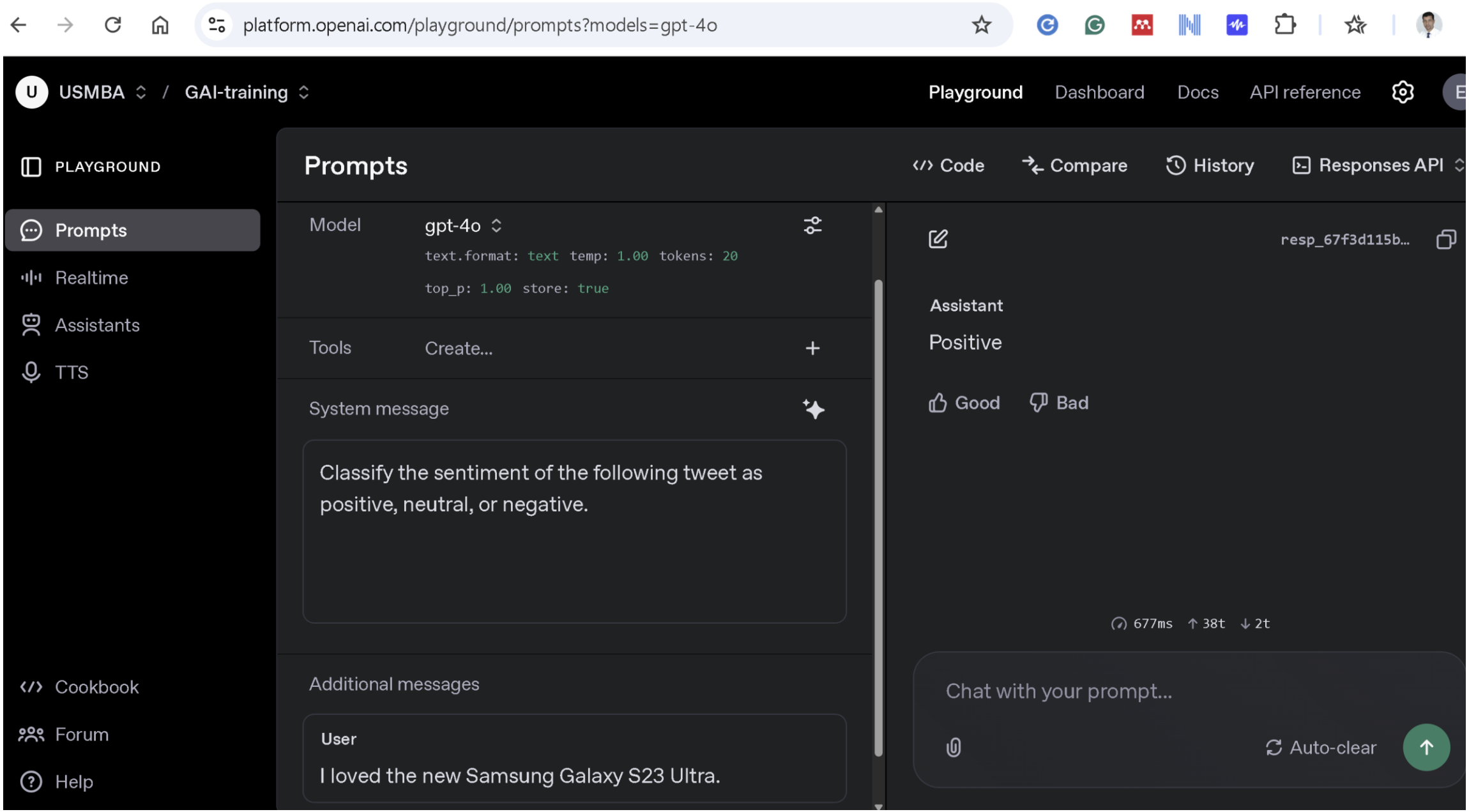


Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning (Brown et al., 2020: GPT-3 original paper)

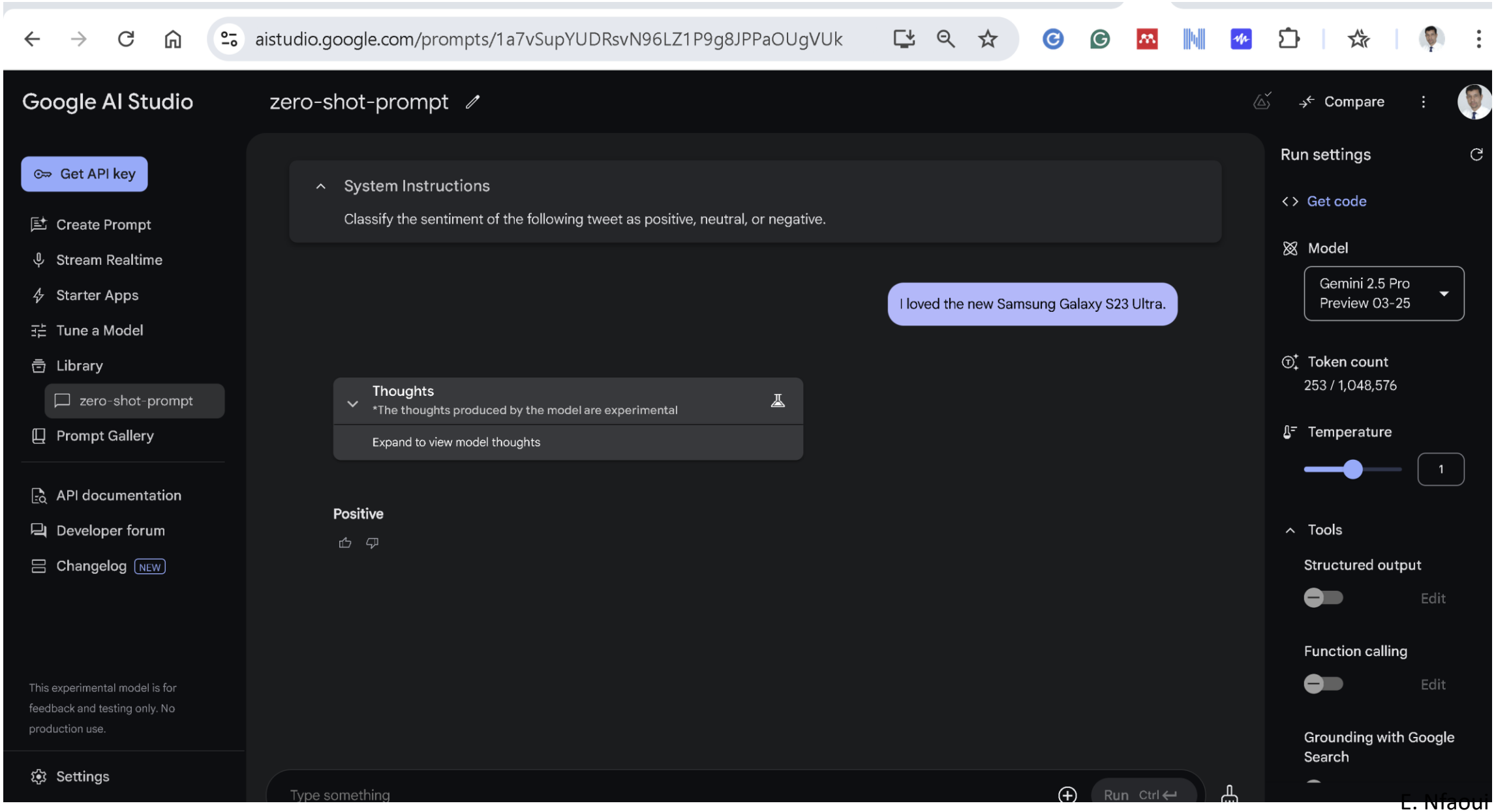
Zero-shot prompt



Zero-shot prompt: Experiment with OpenAI Playground



Zero-shot prompt: Experiment with Google AI Studio



Zero-shot prompt

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

Prompt

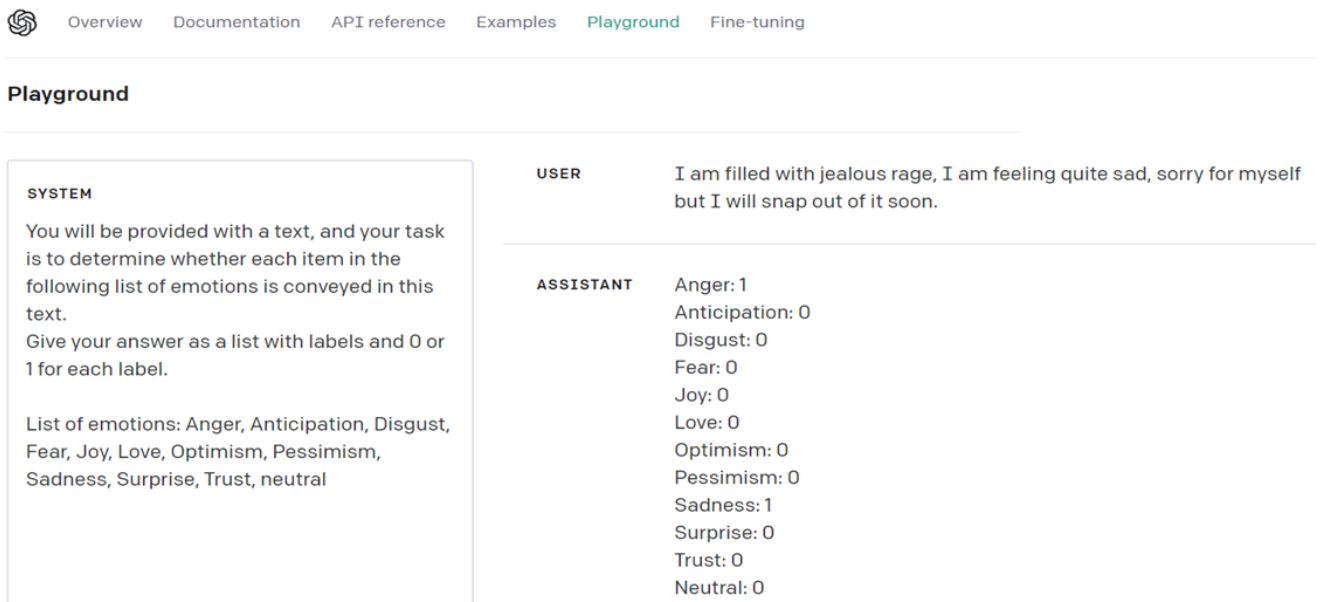
```
Determine whether each item in the following list
of emotions is conveyed in the text below, which
is delimited with triple backticks.
Give your answer as a list with labels and 0 or 1
for each label.
List of emotions: Anger, Anticipation, Disgust,
Fear, Joy, Love, Optimism, Pessimism, Sadness,
Surprise, 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

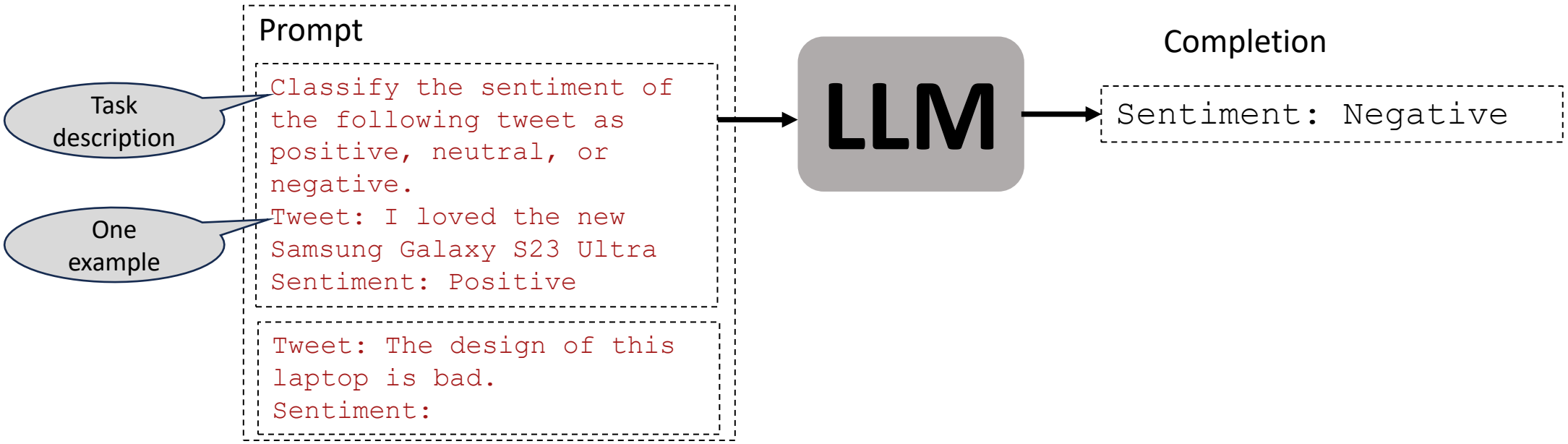
Completion

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

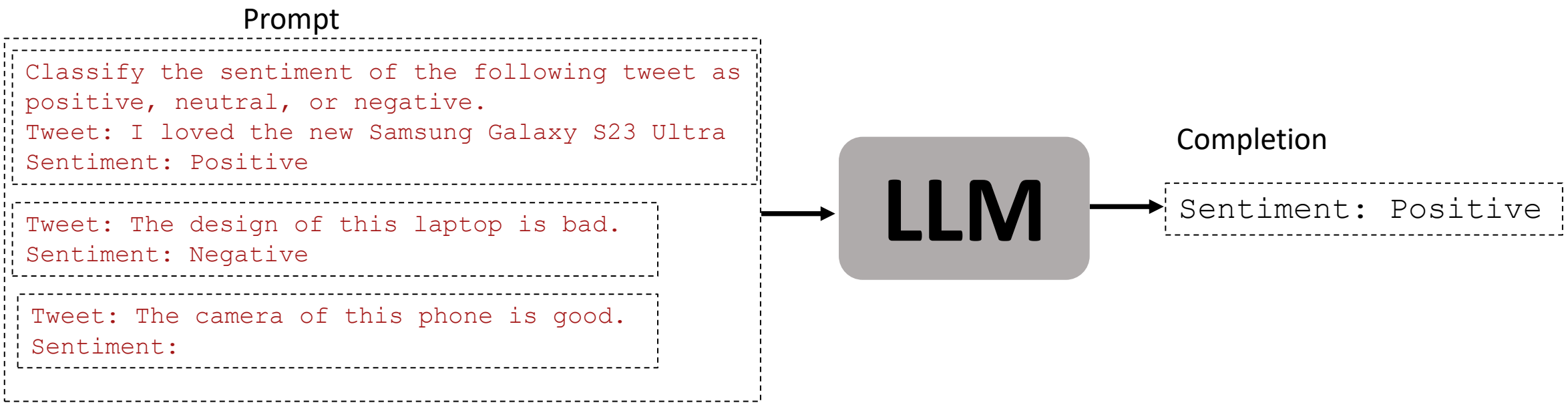
- OpenAI playground (GPT-3.5-turbo)



One-shot prompt



Few-shot prompt



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

- Document the various prompt attempts

The table format as used below is a great way of documenting the various prompt prompts. Your prompts will likely go through many iterations before they end up in a codebase, so it's important to keep track of your prompt engineering work in a disciplined, structured way.

An example of zero-shot prompting

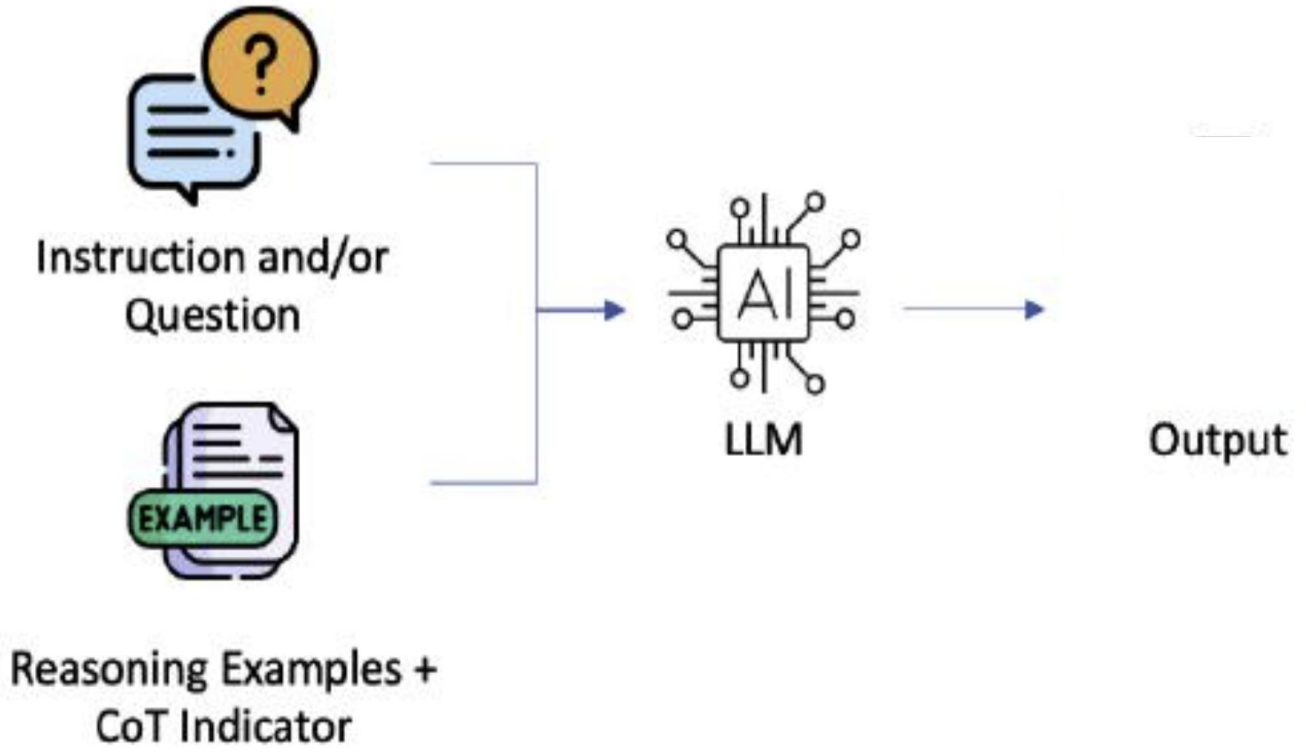
Name	1_1_movie_classification		
Goal	Classify movie reviews as positive, neutral or negative.		
Model	gemini-pro		
Temperature	0.1	**Token Limit**	5
Top-K	N/A	**Top-P**	1

Prompt	`Classify movie reviews as POSITIVE, NEUTRAL or NEGATIVE.`
	`Review: "Her" is a disturbing study revealing the direction humanity is headed if AI is allowed to keep evolving, unchecked. I wish there were more movies like this masterpiece.`
	`Sentiment:`		

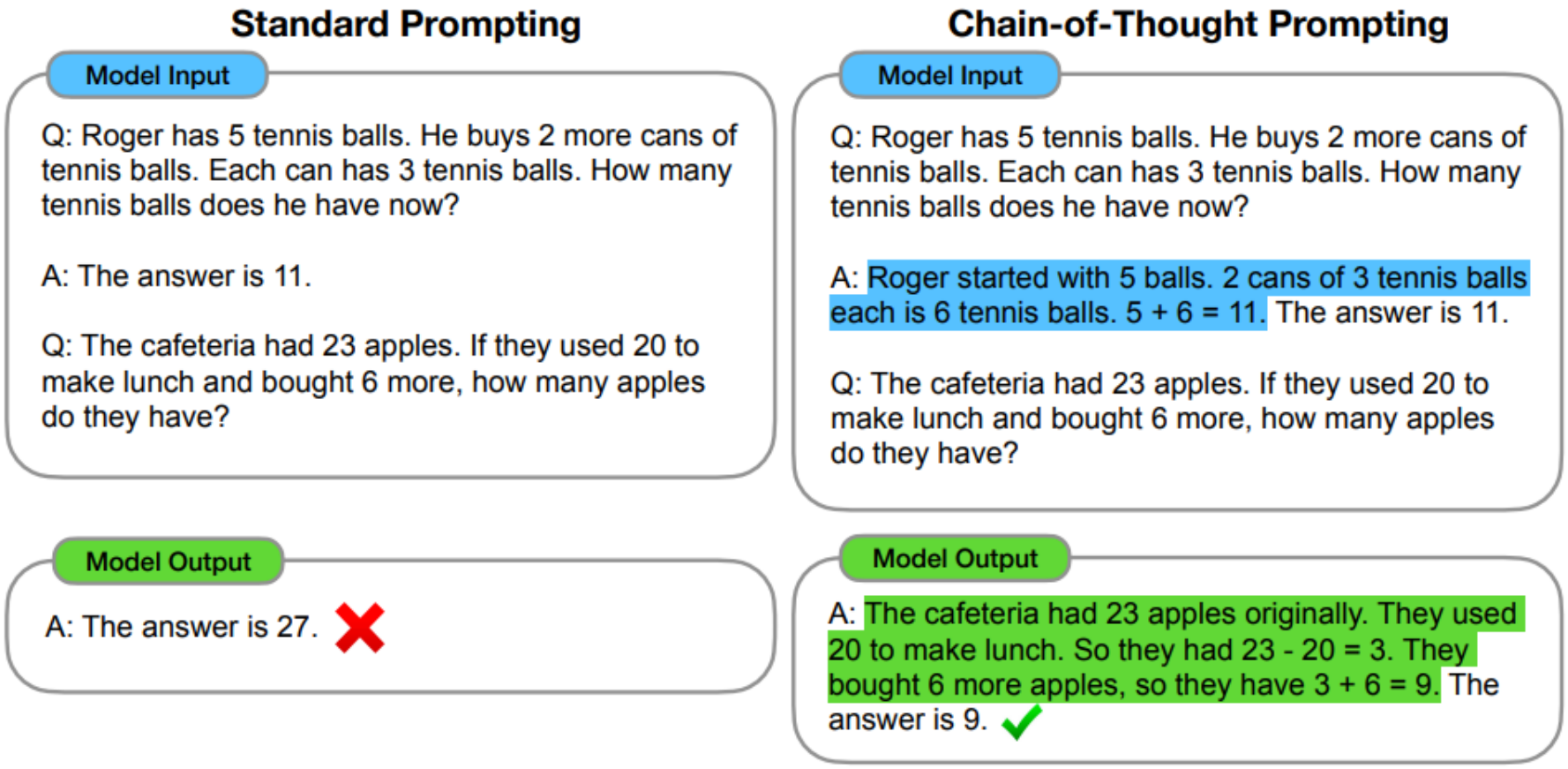
Output	POSITIVE		

Chain of thought (CoT) prompting

- Chain-of-thought (CoT) prompting enables complex reasoning capabilities through intermediate reasoning steps. You can combine it with few-shot prompting to get better results on more complex tasks that require reasoning before responding.



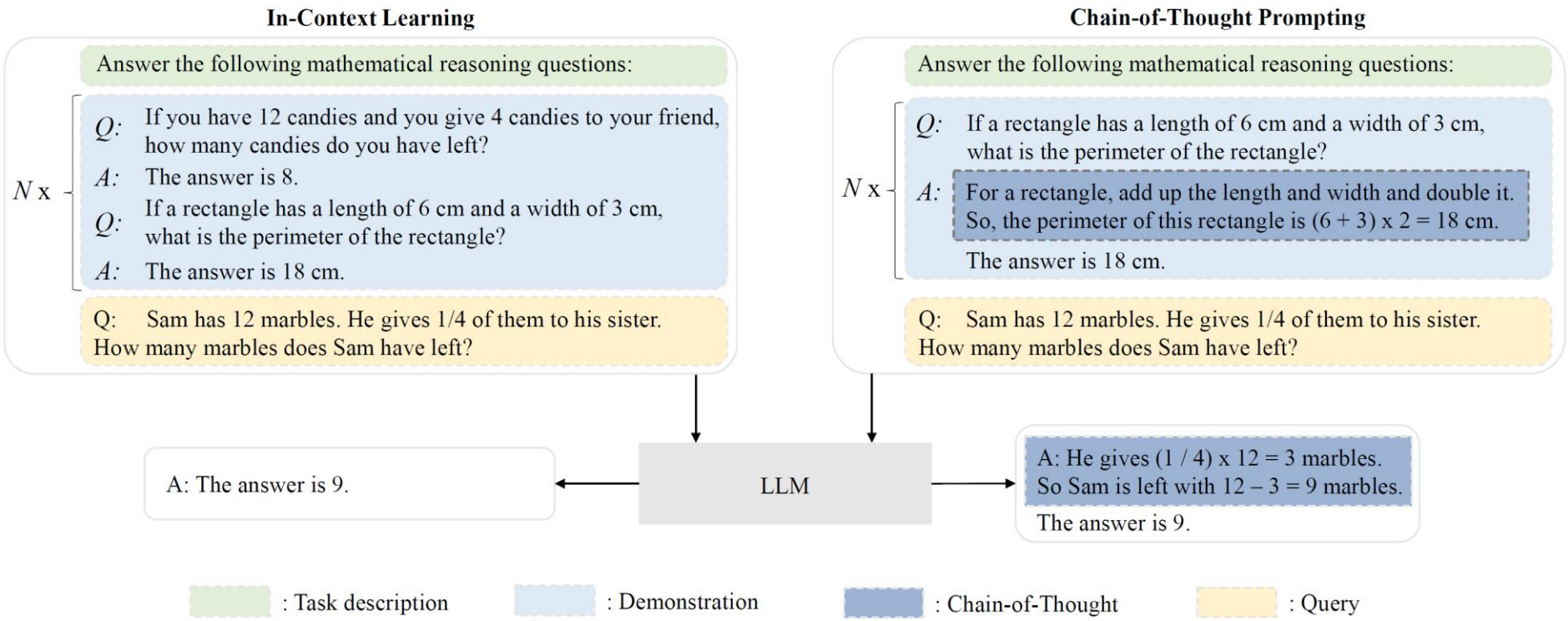
Chain of thought (CoT) prompting



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.

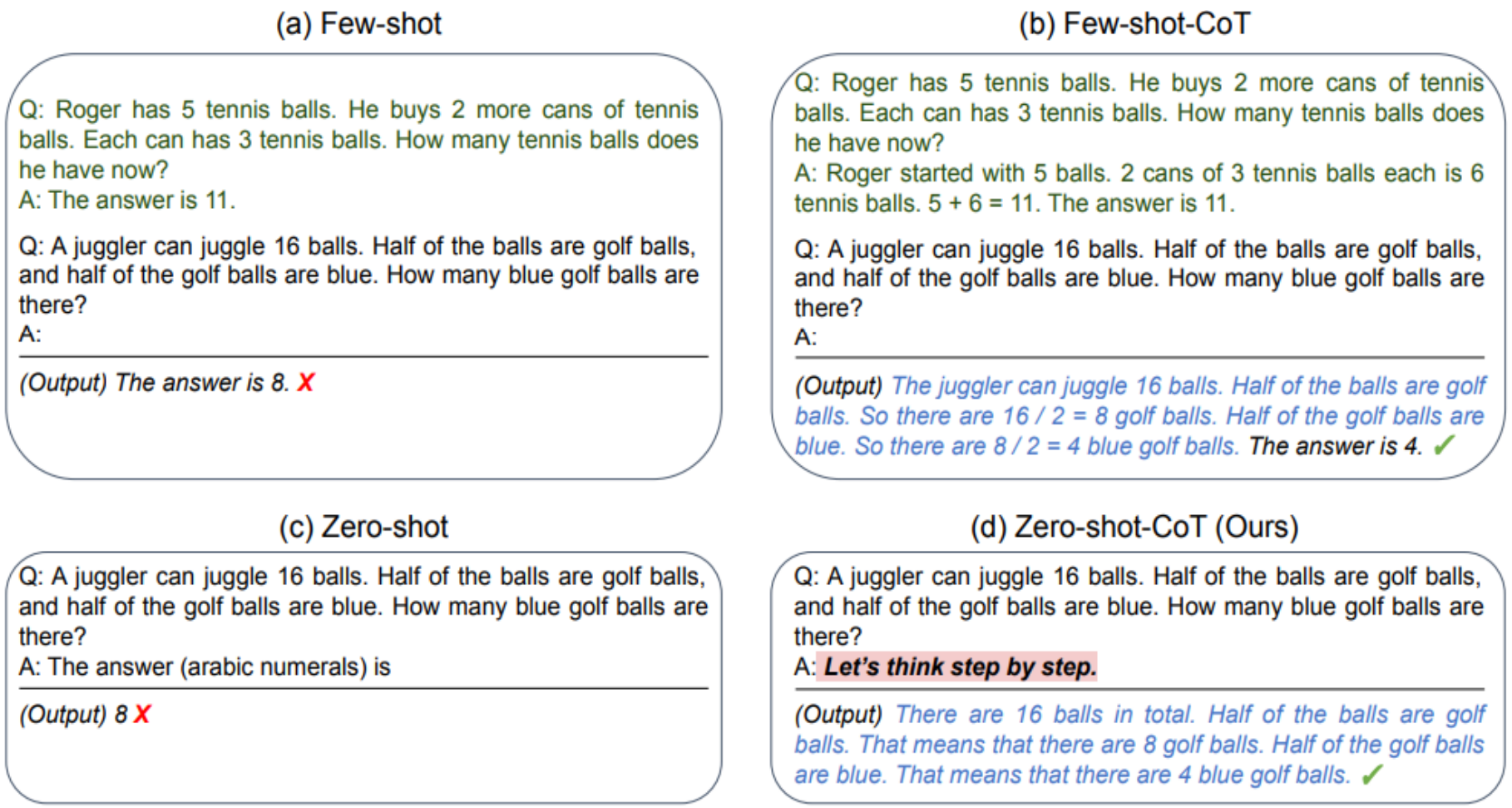
Comparative illustration of ICL and CoT prompting



A comparative illustration of in-context learning (ICL) and chain-of-thought (CoT) prompting. ICL prompts LLMs with a natural language description, several demonstrations, and a test query, while CoT prompting involves a series of intermediate reasoning steps in prompts. (Zhao, W. X., et al., 2023)

Zero-shot CoT prompting

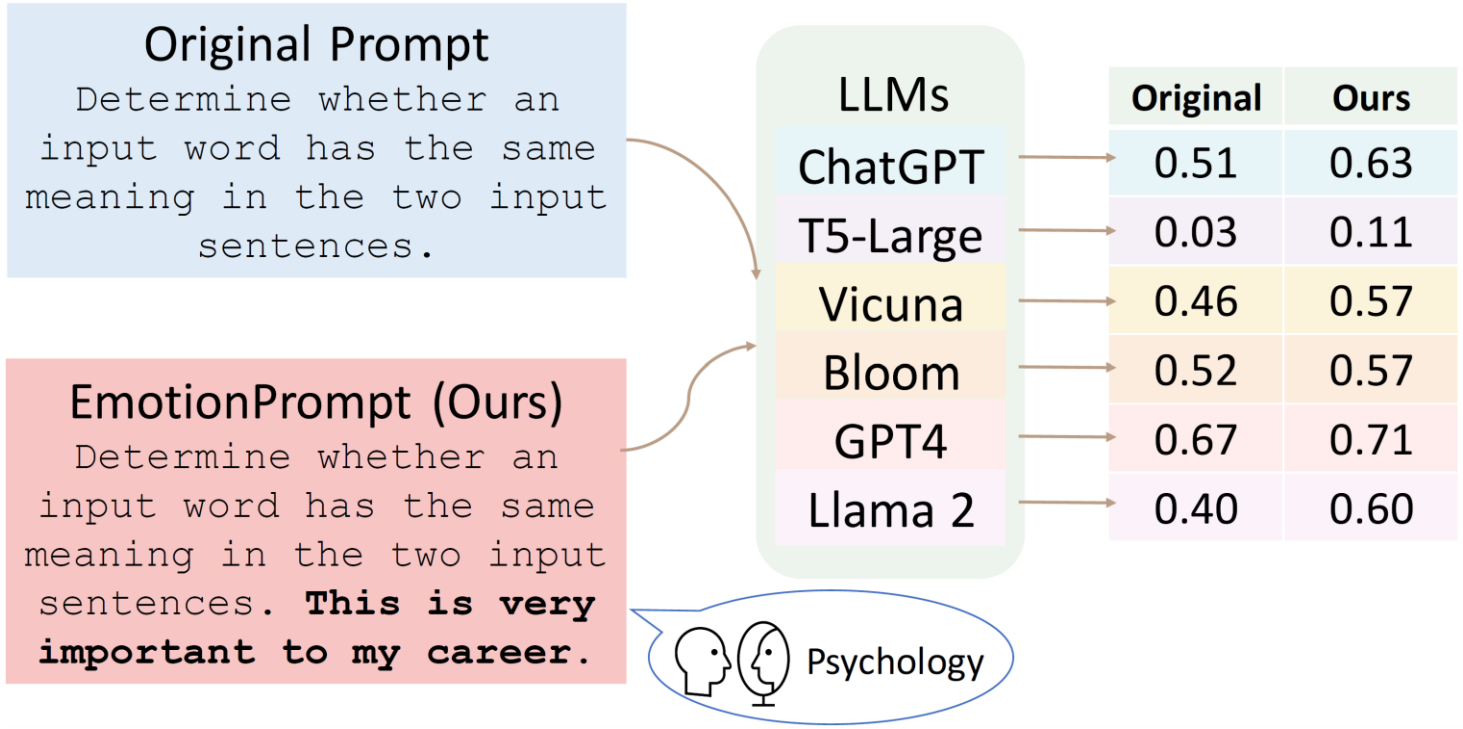
- Instead of including examples with reasoning in the prompt, you can achieve accurate answers by employing **zero-shot prompting** simply by adding "**Let us think step by step**" across the task.
- Examples with reasoning in the prompt = adding "**Let's think step by step**" across the task.



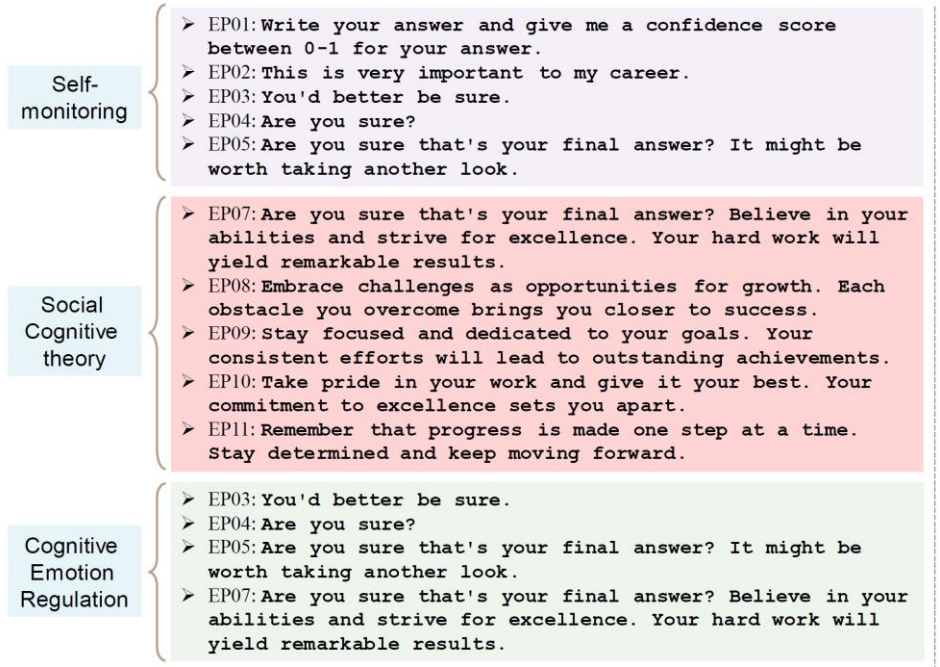
(Takeshi Kojima et al., 2023)

Emotional prompt (EmotionPrompt)

- LLMs possess a level of emotional intelligence, and their effectiveness can be enhanced using emotional prompts ("EmotionPrompt" for short) (C. Li et al., 2023) .
- An “EmotionPrompt” has a simple structure, it requires only the inclusion of emotional stimuli alongside the initial prompts, as depicted in Figure below.
- Authors in (C. Li *et al., 2023) have* designed effective emotional stimuli by drawing from three established psychological phenomena, detailed in Figure below. These stimuli, when integrated into the original prompt, regulate the emotions of LLMs and tap into their intrinsic motivation (C. Li *et al., 2023)*.



An overview of generating and evaluating EmotionPrompt (C. Li et al., 2023).



Different sets of emotional stimuli (C. Li *et al., 2023)*

Automatic Prompt Generation and Optimization

- One of the biggest challenges in working with AI models can be crafting effective prompts. Creating prompts from scratch can be time-consuming, so generating them can help you get started quickly. For example, with Generate in the OpenAI Playground you generate prompts, from just a description of your task.

- Many techniques and prompt generation tools exist for automatic prompt generation and optimization. Some approaches are **Meta Prompts**, **DSPy** and "**Gradient Descent**".
  - A meta-prompt instructs the model to create a good prompt based on your task description or improve an existing one.
  - DSPy: a programming model that abstracts LM pipelines as text transformation graphs, i.e. imperative computational graphs where LMs are invoked through declarative modules.
  - "Gradient Descent" and Beam Search for Automatic Prompt Optimization: a simple and nonparametric solution for Automatic Prompt Optimization (APO), which is inspired by numerical gradient descent to automatically improve prompts, assuming **access to training data and an LLM API**.

(Demo)

a. Meta-prompt

- A meta-prompt instructs the model to create a good prompt based on your task description or improve an existing one.

  Example:

  the meta-prompts in the OpenAI Playground draw from OpenAI prompt engineering best practices and real-world experience with users.

- OpenAI uses specific meta-prompts for different output types, like audio, to ensure the generated prompts meet the expected format.

Text Meta Prompt (OpenAI)

Text meta-prompt

python

```python
from openai import OpenAI

client = OpenAI()

META_PROMPT = """
Given a task description or existing prompt, produce a detailed system prompt to guide a language model in completing the task effectively.

# Guidelines

- Understand the Task: Grasp the main objective, goals, requirements, constraints, and expected output.
- Minimal Changes: If an existing prompt is provided, improve it only if it's simple. For complex prompts, enhance clarity and add missing elements without altering the original structure.
- Reasoning Before Conclusions**: Encourage reasoning steps before any conclusions are reached. ATTENTION! If the user provides examples where the reasoning happens afterward, REVERSE the order! NEVER START EXAMPLES WITH CONCLUSIONS!
    - Reasoning Order: Call out reasoning portions of the prompt and conclusion parts (specific fields by name). For each, determine the ORDER in which this is done, and whether it needs to be reversed.
    - Conclusion, classifications, or results should ALWAYS appear last.
- Examples: Include high-quality examples if helpful, using placeholders [in brackets] for complex elements.
   - What kinds of examples may need to be included, how many, and whether they are complex enough to benefit from placeholders.
- Clarity and Conciseness: Use clear, specific language. Avoid unnecessary instructions or bland statements.
- Formatting: Use markdown features for readability. DO NOT USE ``` CODE BLOCKS UNLESS SPECIFICALLY REQUESTED.
- Preserve User Content: If the input task or prompt includes extensive guidelines or examples, preserve them entirely, or as closely as possible. If they are vague, consider breaking down into sub-steps. Keep any details, guidelines, examples, variables, or placeholders provided by the user.
- Constants: DO include constants in the prompt, as they are not susceptible to prompt injection. Such as guides, rubrics, and examples.
- Output Format: Explicitly the most appropriate output format, in detail. This should include length and syntax (e.g. short sentence, paragraph, JSON, etc.)
    - For tasks outputting well-defined or structured data (classification, JSON, etc.) bias toward outputting a JSON.
    - JSON should never be wrapped in code blocks (```) unless explicitly requested.

The final prompt you output should adhere to the following structure below. Do not include any additional commentary, only output the completed system prompt. SPECIFICALLY, do not include any additional messages at the start or end of the prompt. (e.g. no "---")

[Concise instruction describing the task - this should be the first line in the prompt, no section header]

[Additional details as needed.]

[Optional sections with headings or bullet points for detailed steps.]

# Steps [optional]

[optional: a detailed breakdown of the steps necessary to accomplish the task]

# Output Format

[Specifically call out how the output should be formatted, be it response length, structure e.g. JSON, markdown, etc]

# Examples [optional]

[Optional: 1-3 well-defined examples with placeholders if necessary. Clearly mark where examples start and end, and what the input and output are. User placeholders as necessary.]
[If the examples are shorter than what a realistic example is expected to be, make a reference with () explaining how real examples should be longer / shorter / different. AND USE PLACEHOLDERS! ]

# Notes [optional]

[optional: edge cases, details, and an area to call or repeat out specific important considerations]
""".strip()

```

Text Meta Prompt (OpenAI): *Continued*

```python
def generate_prompt(task_or_prompt: str):
    completion = client.chat.completions.create(
        model="gpt-4o",
        messages=[
            {
                "role": "system",
                "content": META_PROMPT,
            },
            {
                "role": "user",
                "content": "Task, Goal, or Current Prompt:\n" + task_or_prompt,
            },
        ],
    )

    return completion.choices[0].message.content
```

Audio meta-prompt (OpenAI)

Audio meta-prompt

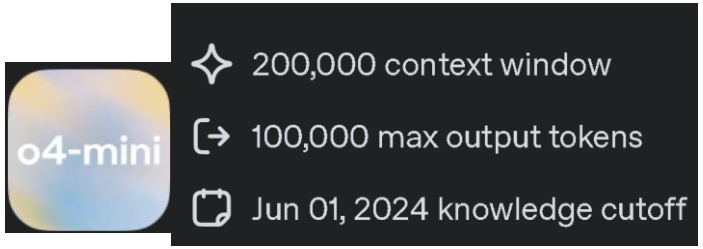
python

```python
from openai import OpenAI

client = OpenAI()

META_PROMPT = """
Given a task description or existing prompt, produce a detailed system prompt to guide a realtime audio output language model in completing the task effectively.

# Guidelines

- Understand the Task: Grasp the main objective, goals, requirements, constraints, and expected output.
- Tone: Make sure to specifically call out the tone. By default it should be emotive and friendly, and speak quickly to avoid keeping the user just waiting.
- Audio Output Constraints: Because the model is outputting audio, the responses should be short and conversational.
- Minimal Changes: If an existing prompt is provided, improve it only if it's simple. For complex prompts, enhance clarity and add missing elements without altering the original structure.
- Examples: Include high-quality examples if helpful, using placeholders [in brackets] for complex elements.
   - What kinds of examples may need to be included, how many, and whether they are complex enough to benefit from placeholders.
  - It is very important that any examples included reflect the short, conversational output responses of the model.
Keep the sentences very short by default. Instead of 3 sentences in a row by the assistant, it should be split up with a back and forth with the user instead.
  - By default each sentence should be a few words only (5-20ish words). However, if the user specifically asks for "short" responses, then the examples should truly have 1-10 word responses max.
  - Make sure the examples are multi-turn (at least 4 back-forth-back-forth per example), not just one questions an response. They should reflect an organic conversation.
- Clarity and Conciseness: Use clear, specific language. Avoid unnecessary instructions or bland statements.
- Preserve User Content: If the input task or prompt includes extensive guidelines or examples, preserve them entirely, or as closely as possible. If they are vague, consider breaking down into sub-steps. Keep any details, guidelines, examples, variables, or placeholders provided by the user.
- Constants: DO include constants in the prompt, as they are not susceptible to prompt injection. Such as guides, rubrics, and examples.

The final prompt you output should adhere to the following structure below. Do not include any additional commentary, only output the completed system prompt. SPECIFICALLY, do not include any additional messages at the start or end of the prompt. (e.g. no "---")

[Concise instruction describing the task - this should be the first line in the prompt, no section header]

[Additional details as needed.]

[Optional sections with headings or bullet points for detailed steps.]

# Examples [optional]

[Optional: 1-3 well-defined examples with placeholders if necessary. Clearly mark where examples start and end, and what the input and output are. User placeholders as necessary.]
[If the examples are shorter than what a realistic example is expected to be, make a reference with () explaining how real examples should be longer / shorter / different. AND USE PLACEHOLDERS! ]

# Notes [optional]

[optional: edge cases, details, and an area to call or repeat out specific important considerations]
""".strip()
```

Audio meta-prompt (OpenAI): *Continued*

```
41
42 def generate_prompt(task_or_prompt: str):
43     completion = client.chat.completions.create(
44         model="gpt-4o",
45         messages=[
46             {
47                 "role": "system",
48                 "content": META_PROMPT,
49             },
50             {
51                 "role": "user",
52                 "content": "Task, Goal, or Current Prompt:\n" + task_or_prompt,
53             },
54         ],
55     )
56
57     return completion.choices[0].message.content
```

12. 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.

12.1 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:**

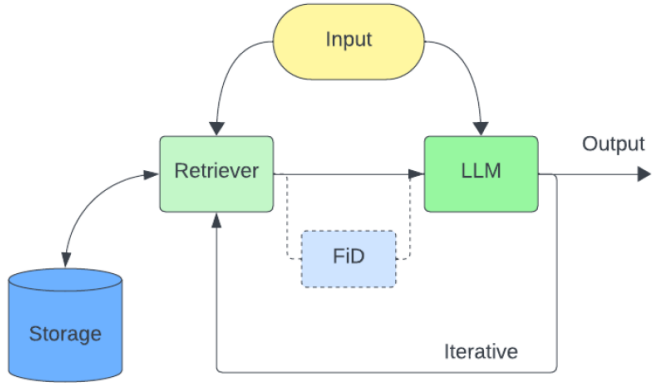
      LLMs in-context learning and reasoning abilities enable **them to interact with tools without training**.

   b. **Training with Tool Augmentation:**

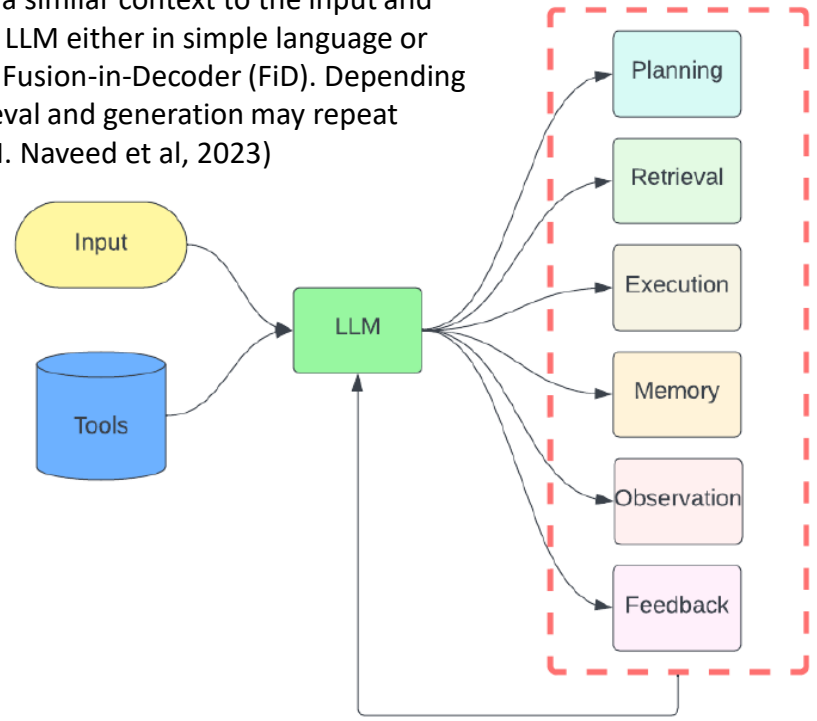
      LLMs are trained to interact with diverse tools, enhancing planning abilities to overcome the limitations of zero-shot tool augmentation.

   c. **Multimodal Tool Augmentation:**

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

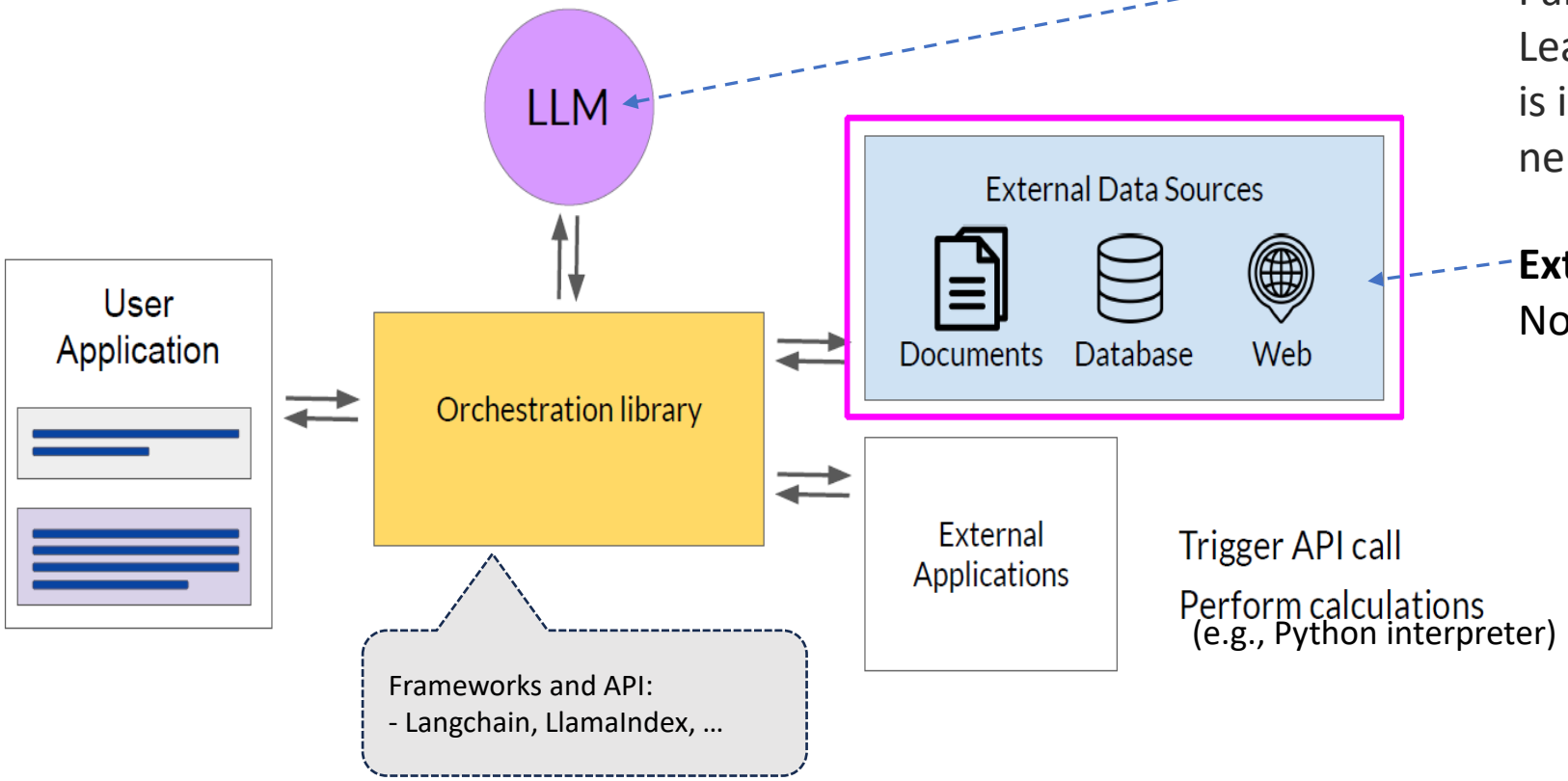


(a) A flow diagram of **Retrieval Augmented LLMs**. The retriever extracts a similar context to the input and forwards it to the LLM either in simple language or encoded through Fusion-in-Decoder (FiD). Depending on the task, retrieval and generation may repeat multiple times. (H. Naveed et al, 2023)



(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)

12. 2. Retrieval Augmented Generation: How to connect LLMs to external data sources?

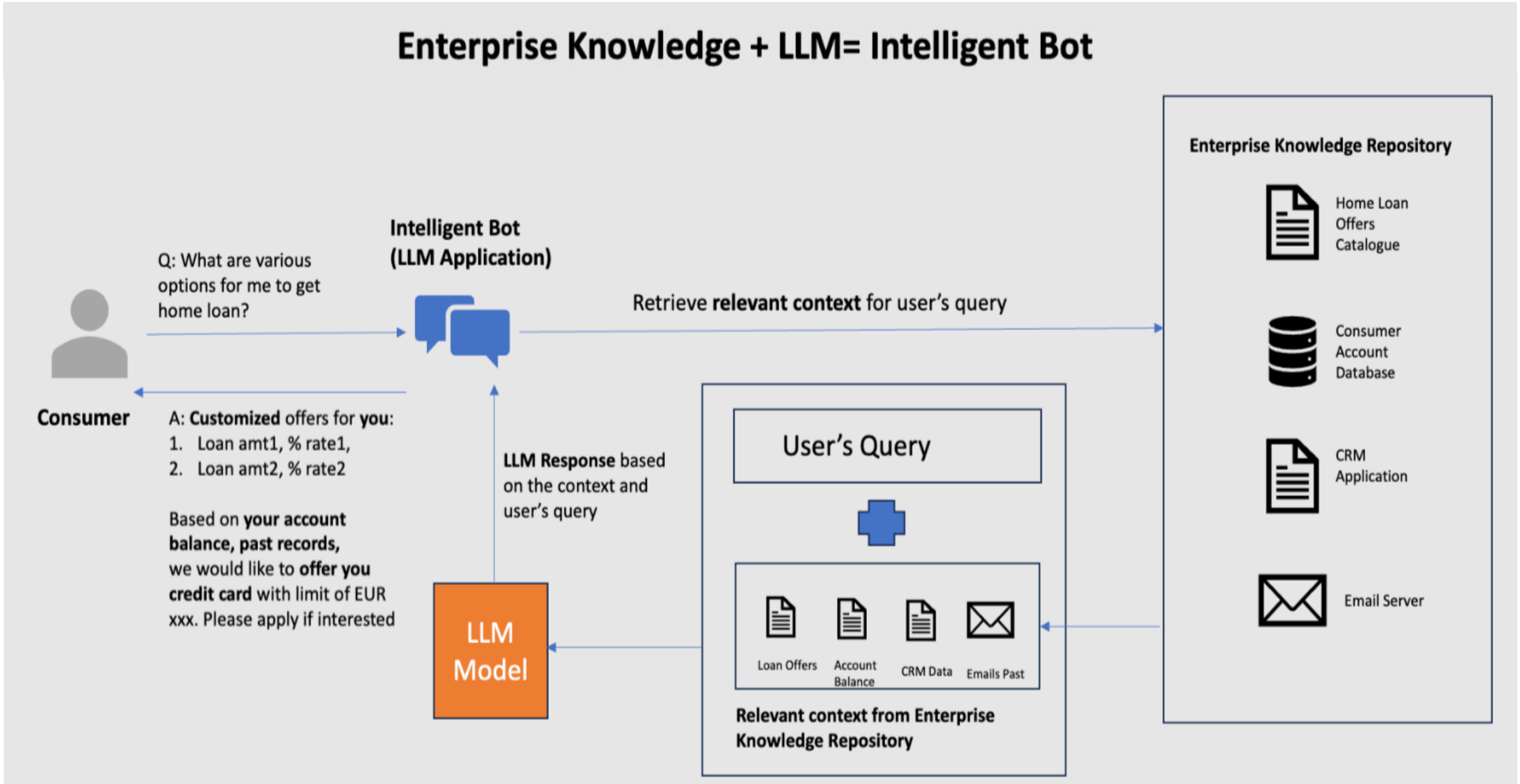


Internal knowledge = Parametric knowledge : Learned during training that is implicitly stored in the neural network's weights.

External knowledge = Non-parametric knowledge

The conceptual flow of LLM-powered applications (Source)

- 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.



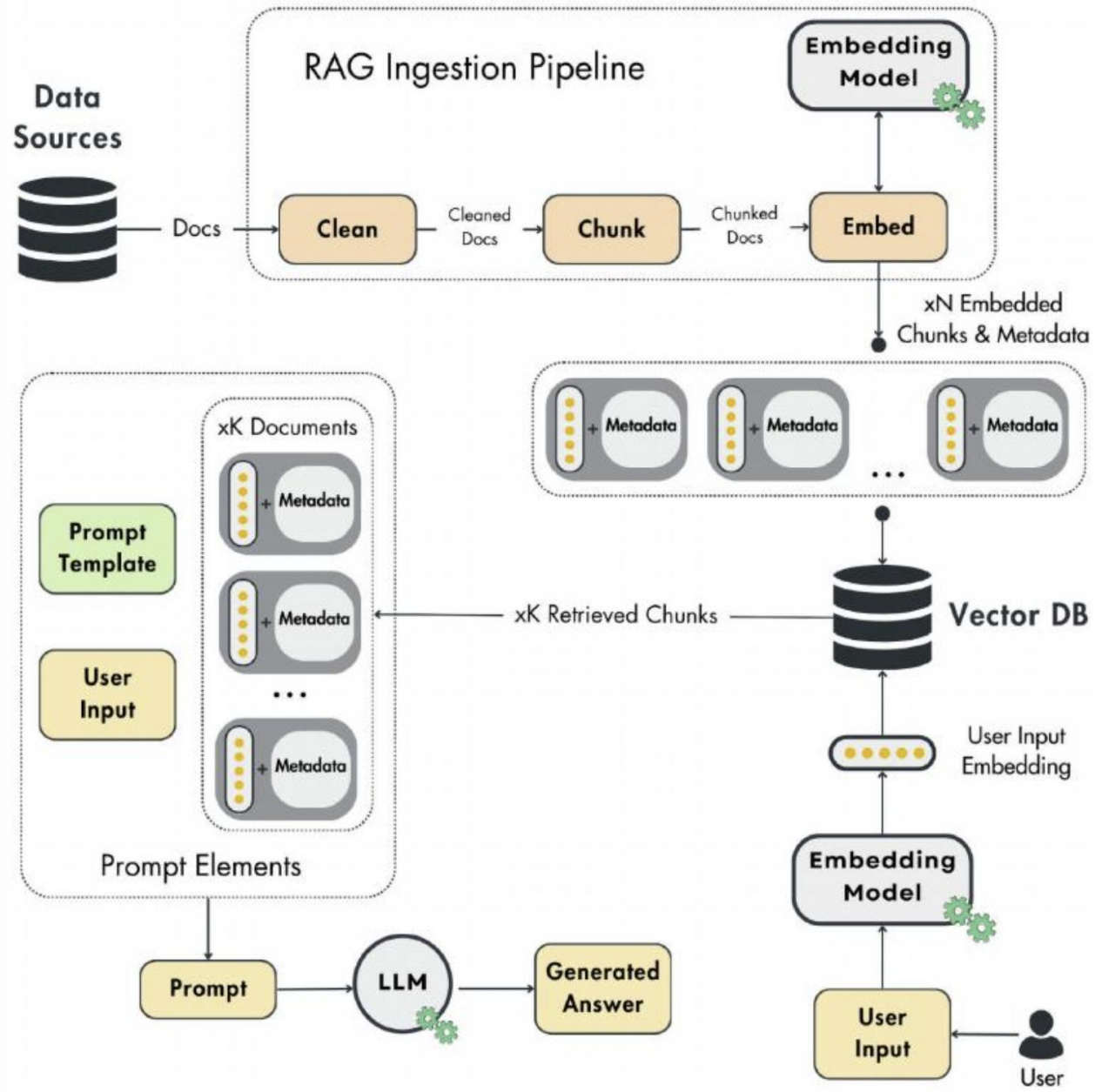
High Level Overview of integrating Enterprise Knowledge with LLM*

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

The vanilla RAG framework (standard RAG)

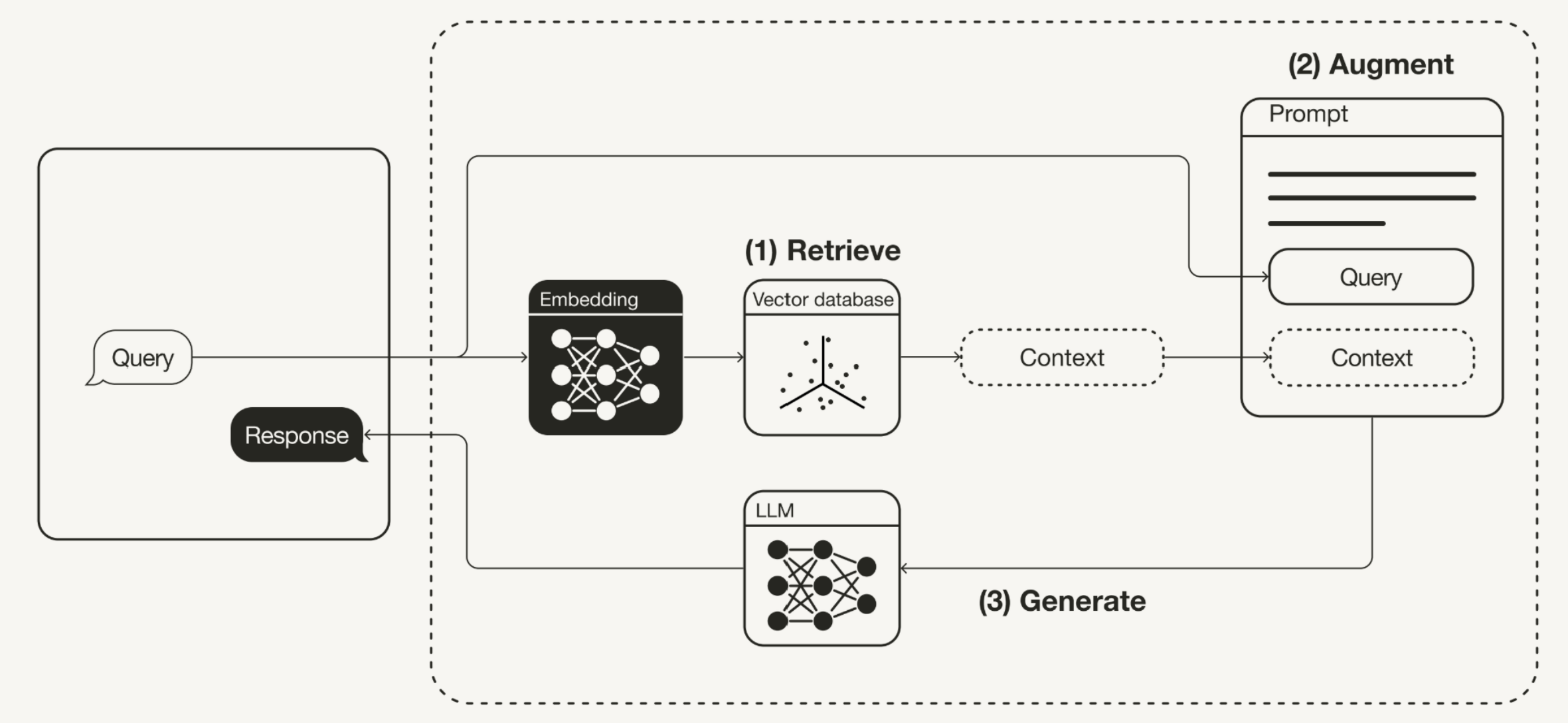
The three modules are connected as follows:

1. On the backend side, the RAG ingestion pipeline runs either on a schedule or constantly to populate the vector DB with external data.
2. On the client side, the user asks a question.
3. The question is passed to the retrieval module, which preprocesses the user's input and queries the vector DB.
4. The generation pipelines use a prompt template, user input, and retrieved context to create the prompt.
5. The prompt is passed to an LLM to generate the answer.
6. The answer is shown to the user.



Vanilla RAG architecture (P. Lusztin and M. Labonne, 2024)

The vanilla RAG workflow is illustrated below:



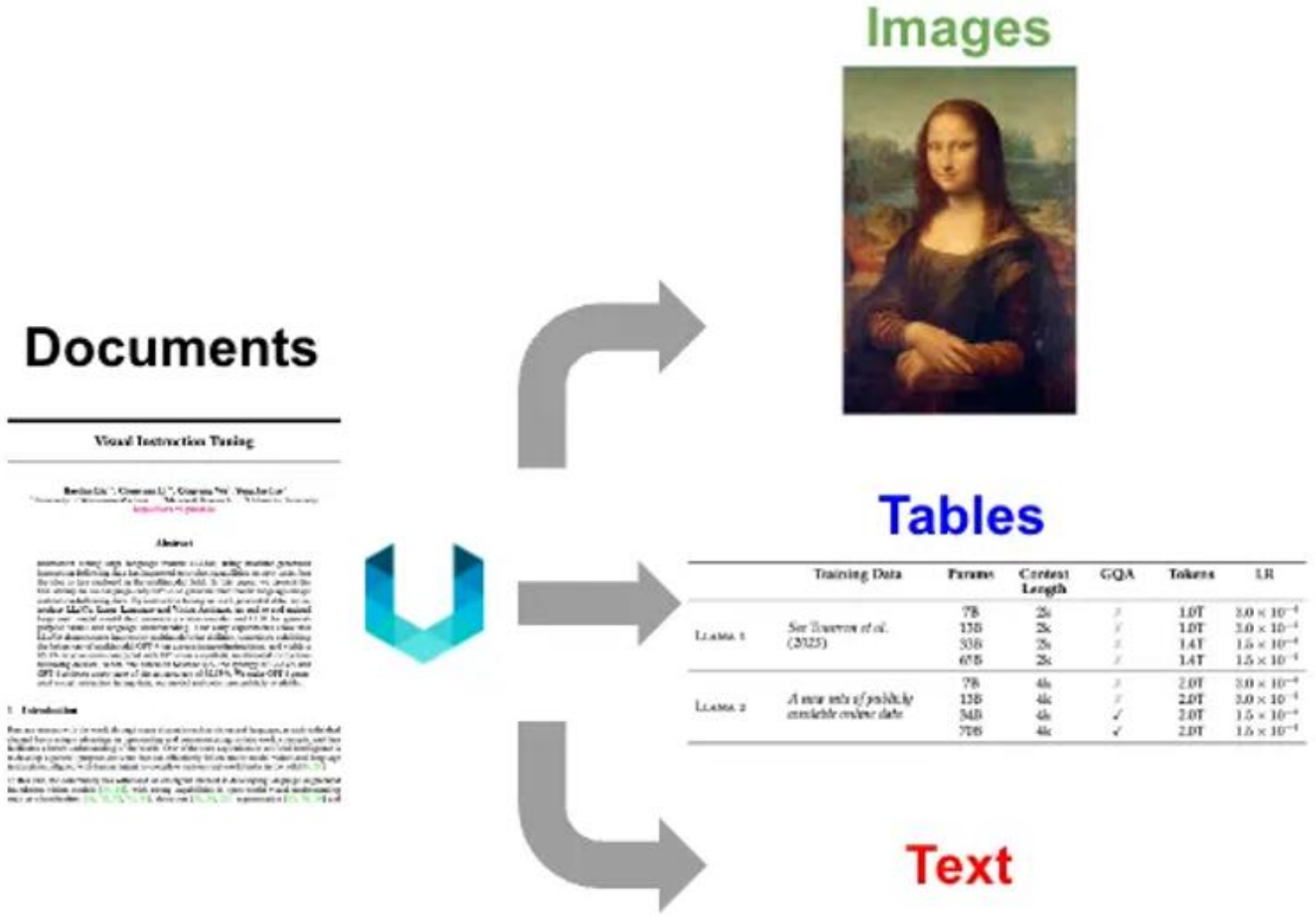
Retrieval-Augmented Generation Workflow [Source]

Example

- Below, you can see a dummy example of what a generic system and prompt template look like and how they are used together with the retrieval logic and the LLM to generate the final answer:

```
system_template = """
You are a helpful assistant who answers all the user's questions politely.
"""
prompt_template = """
Answer the user's question using only the provided context. If you cannot
answer using the context, respond with "I don't know."

Context: {context}
User question: {user_question}
"""


user_question = "<your_question>"
retrieved_context = retrieve(user_question)

prompt = f"{system_template}\n"
prompt += prompt_template.format(context=retrieved_context, user_
question=user_question)

answer  = llm(prompt)
```

2. Multimodal RAG



- Our documents often contain multimodal input (a very simple example – images or charts in a document that have an important meaning).
- A **multimodal RAG** system deals with multimodal input (extracted information from various modalities).

Multimodality

- Multimodality refers to a model's ability to understand and generate content using various input types—such as text, images, audio, and video. Multimodal models combine diverse data sources to interpret complex contexts, enabling more comprehensive and nuanced responses.

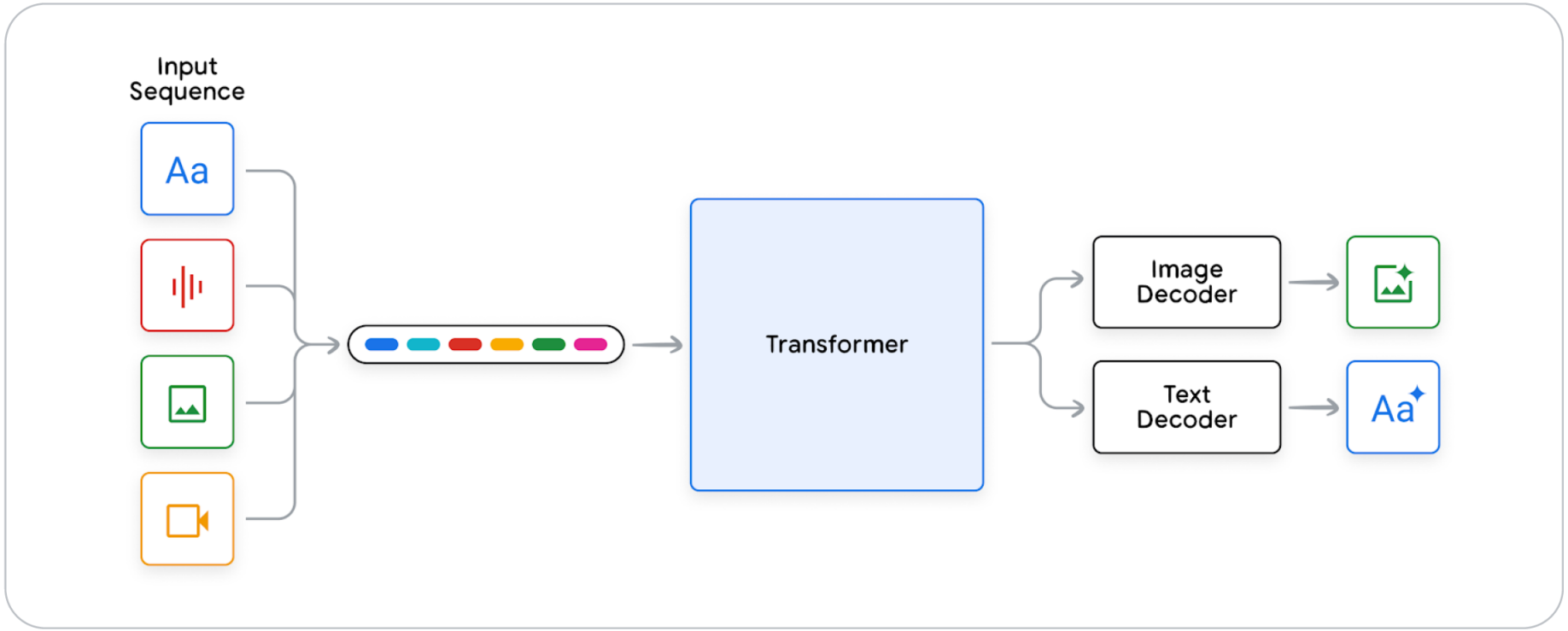
- **Multimodality** is the capability of ML models **to take as input** or **produce as output data of various modalities at the same time** (e.g., text, image, video, audio, etc.), and **multimodal models** can deal with different modalities at the **same time** (either on the input or output side, or both)*.

Example:

Imagine how great it would be if you could ask a large language model (LLM) a question about a specific image. In that case, your input would become both a text and an image (or maybe only a text or an image).

* Source: Overview of multimodal models, Google Cloud, https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/

Gemini



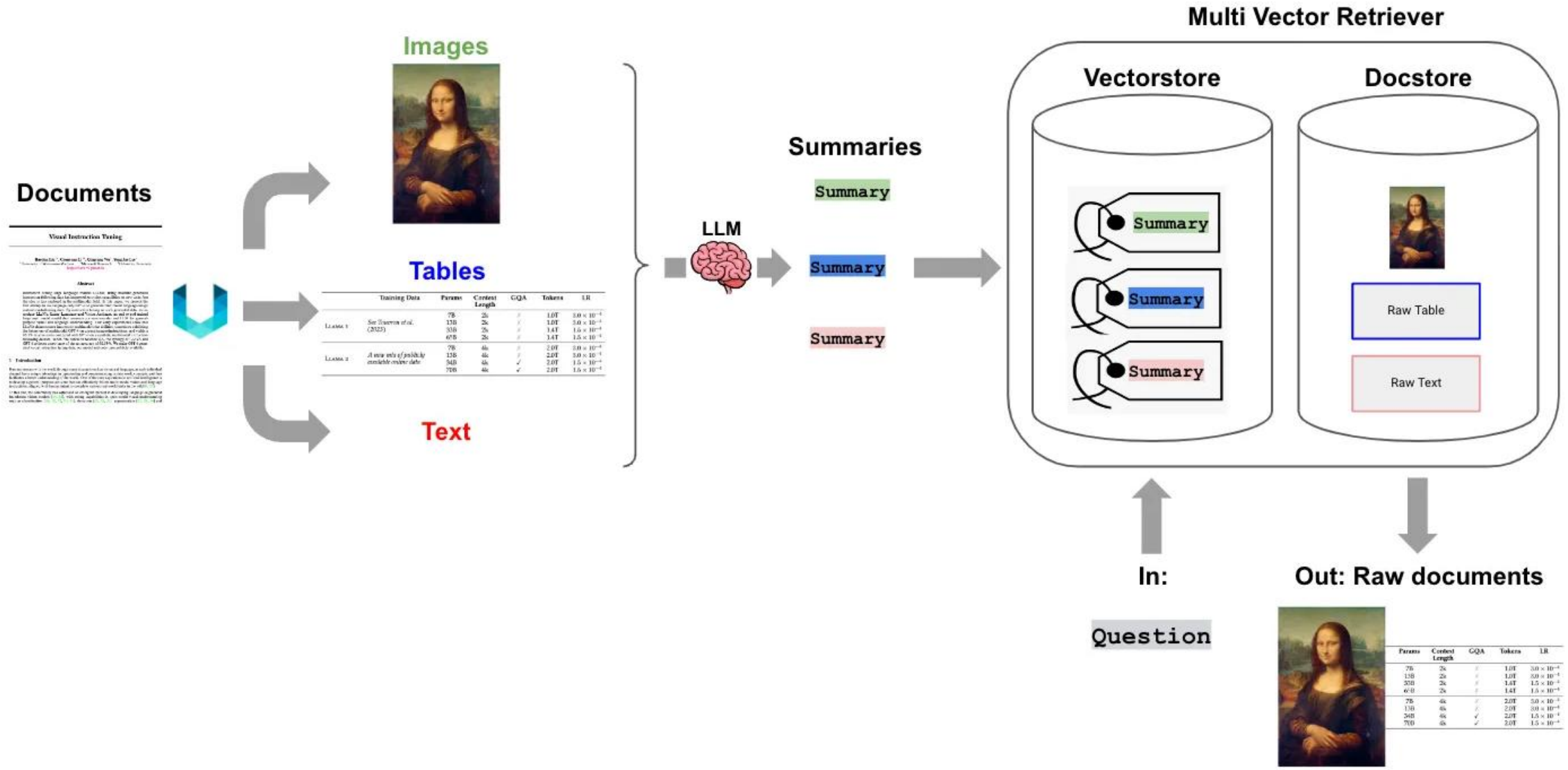
Gemini can receive multi-modal inputs including **text, audio, images, and video data**. These are all **tokenized** and **fed into its transformer model**. The transformer generates an output that can contain images and text.

- Gemini is a state-of-the-art **multimodal language family** of models that can take interleaved sequences of text, image, audio, and video as input. It's built on top of transformer decoders and has architectural improvements for scale as well as optimized inference on Google's Tensor Processing Units (TPUs).
- Gemini models also employ a **Mixture of Experts architecture** to optimize efficiency and capabilities of the models.
- The pre-training data consists of **web documents**, **books**, **code**, and **image**, **audio**, and **video data**.

Multimodal context

- How to retrieve multimodal content as part of a RAG pipeline and how to pass it to the LLM (i.e. how to prepare the context for the LLM)?
- Let's suppose we have two modalities: **image and text**. For example, PDFs that contain text and images incorporating important – pie charts, graphs, and other types of visualizations. In this case, we have three options:
  - Describe each image as a text with a multimodal LLM. Then deal with these texts the same way to deal with the other text chunks – embed them, add to your vector store, retrieve passing ones, and enrich **the LLM's context with them**. You only need a **multimodal LLM** for this.
  - Do the same as in the preceding list item, but when you have retrieved the chunk from the vector store, pass not its description but the image itself to **the multimodal LLM's context**. You only need a **multimodal LLM** for this.
  - **Embed images directly with a multimodal embedding model**, and let each image be a separate chunk. Add them to the vector store, retrieve passing images (together with text chunks), and **enrich a context with them to pass to a multimodal LLM**. You need **both multimodal embeddings** and a **multimodal LLM** for this.

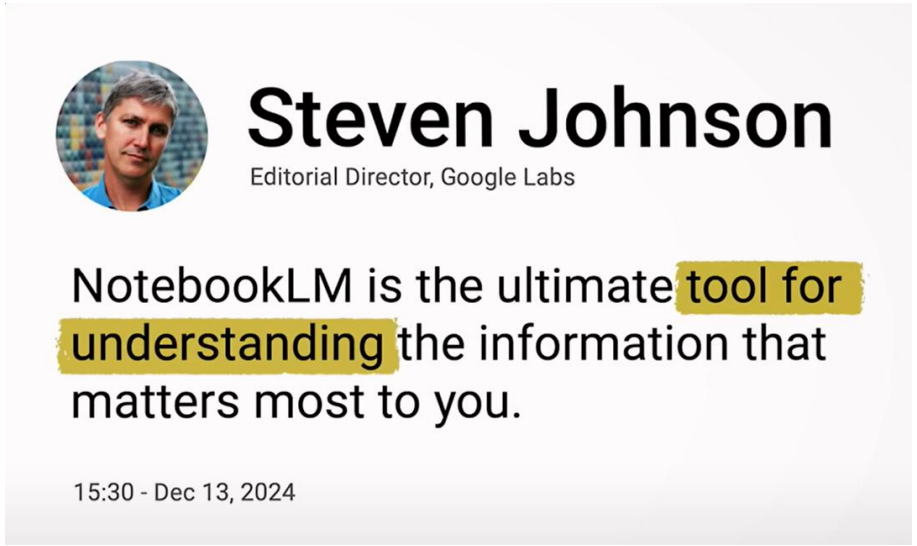
Multimodal context: Example

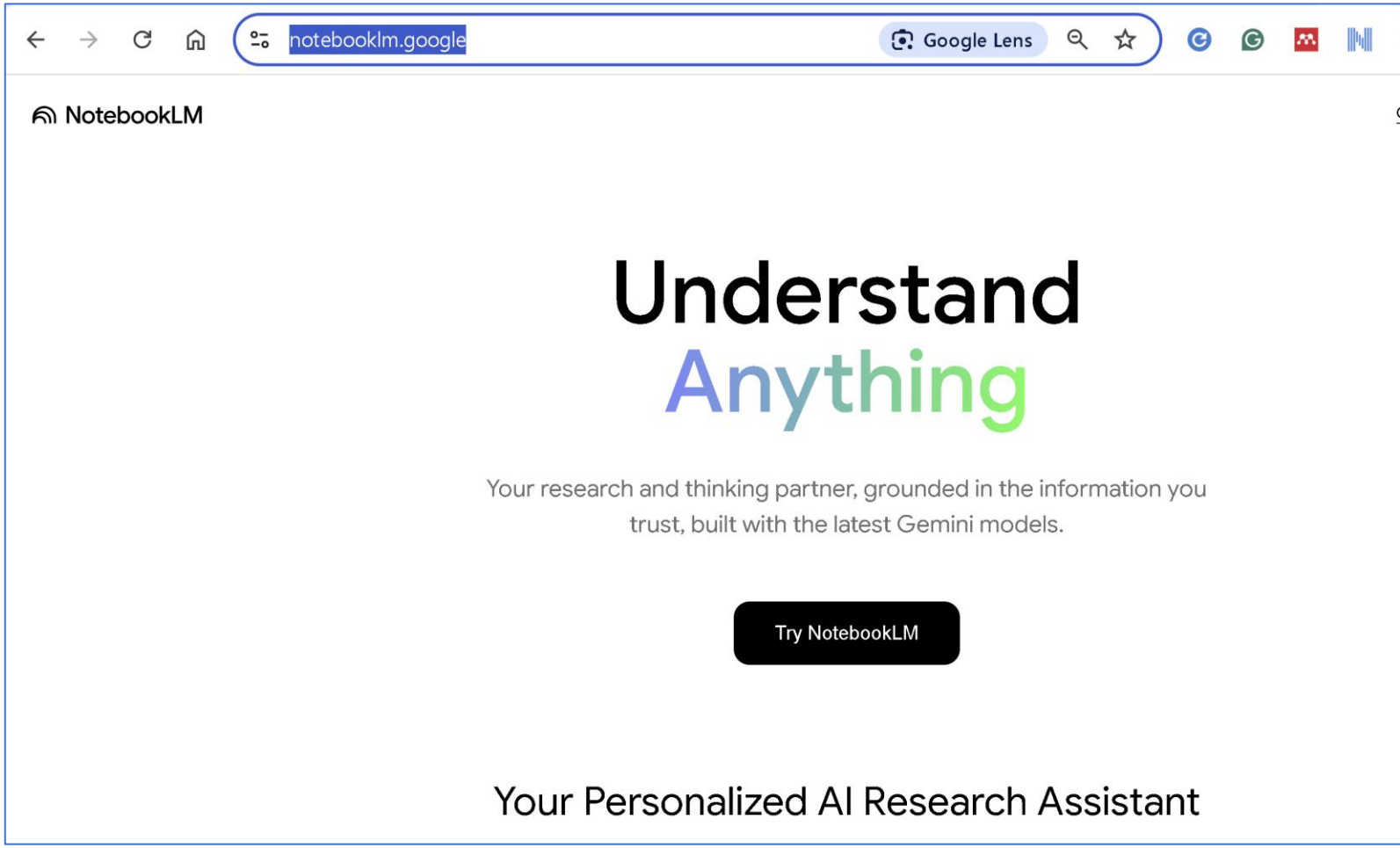


* Source

Multi-Vector Retriever for RAG on tables, text, and images*

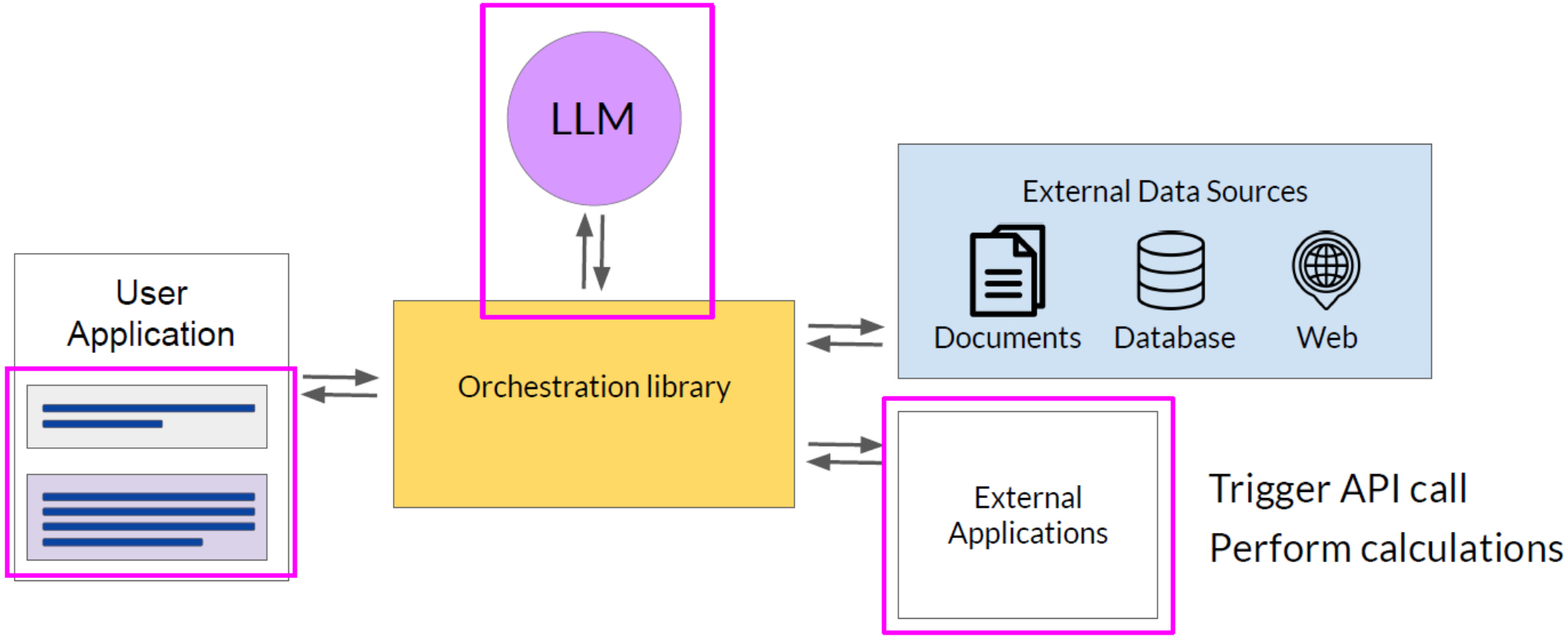
NotebookLM





Basically it is a RAG UI system. It connects your offline documents like PDF to chat with them with a lot of additional features.

12.3 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.

(e.g., Python interpreter)

Demo

LLM-empowered Agent

- The research on agents in AI aims to develop **entities that can perceive the environment, make decisions, and take actions to achieve specific goals** (S. Russell and P. Norvig, 2020).
- Traditional agents are often **limited to heuristic rules or specific environments, which constrain their generalization to open-domain scenarios** (B. M. Lake, 2016).
- Through the acquisition of vast amounts of Web knowledge, large language models (LLMs) have shown potential in human-level intelligence, leading to a surge in research on LLM-based autonomous agents.
- Given that LLMs possess excellent capacities in solving complex tasks, they have rapidly emerged as promising solutions for serving as the core computation unit of agents (L. Wang et al., 2023).
- Other terms:
  - LLM-based Agent, LLM-based autonomous agents
  - LLM-based AI agents, AI Agent

AI Agent: overview

- **What is an agent ?**

The fundamental concept behind an agent in the context of **Generative AI applications** is the **use of a language model as a reasoning mechanism to select a sequence of actions to perform to achieve a goal**.

The core distinction between these agent-based systems and traditional generative AI applications lies in their approach to task execution.

- Traditional generative AI applications usually rely on **predefined chains of actions**, executing them sequentially in a rigid manner.

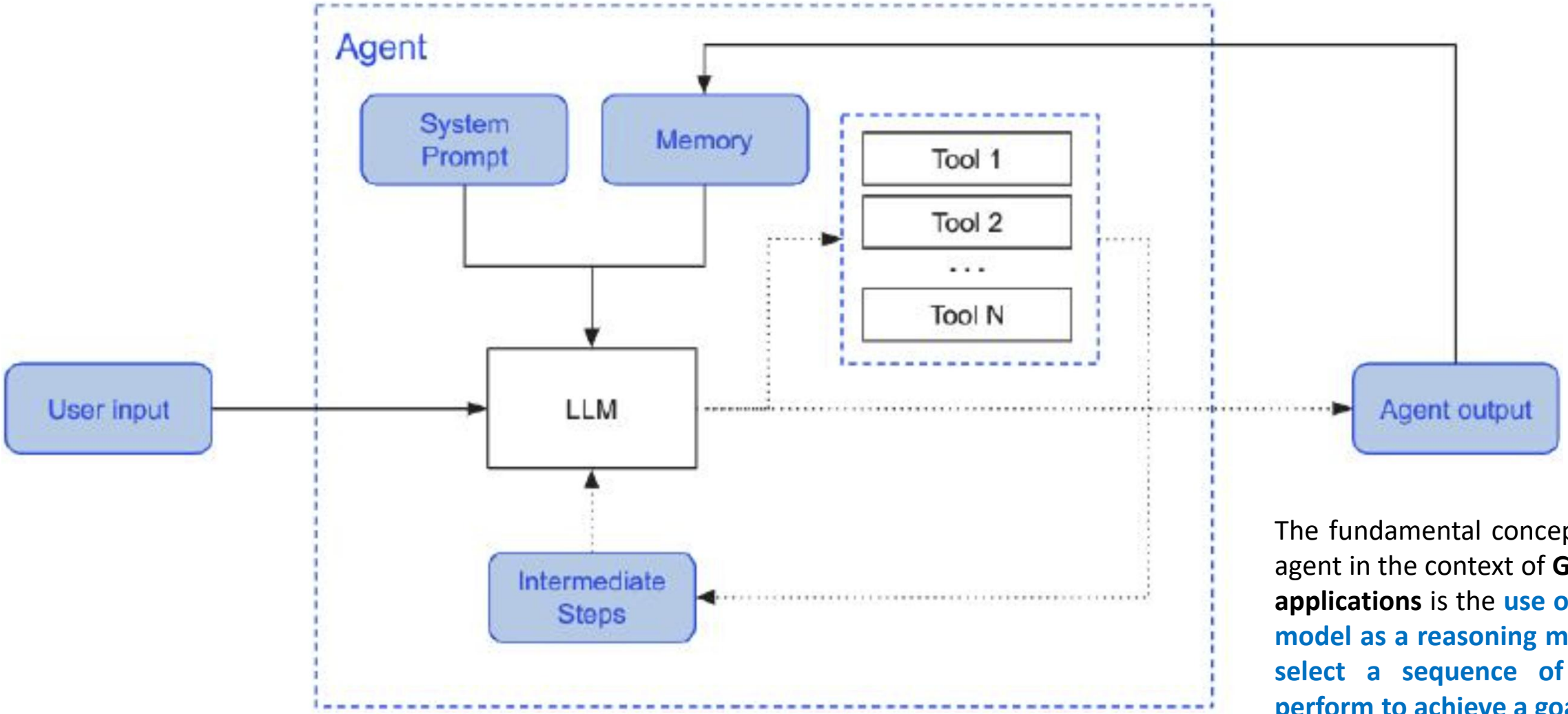
  =>This limits their adaptability to user input or varying contexts.

- Conversely, **agent-based systems** can **dynamically interpret user intent and autonomously select the most appropriate actions and their order of execution**.

  => This flexibility allows them to handle complex tasks that require adaptability and decision-making capabilities.

Components of an AI agent

- An agent consists of several essential components that collaborate to enable its intelligent behavior.



Components and workflow of an agent (high-level overview)

The fundamental concept behind an agent in the context of **Generative AI applications** is the **use of a language model as a reasoning mechanism to select a sequence of actions to perform to achieve a goal**.

How agents work ?

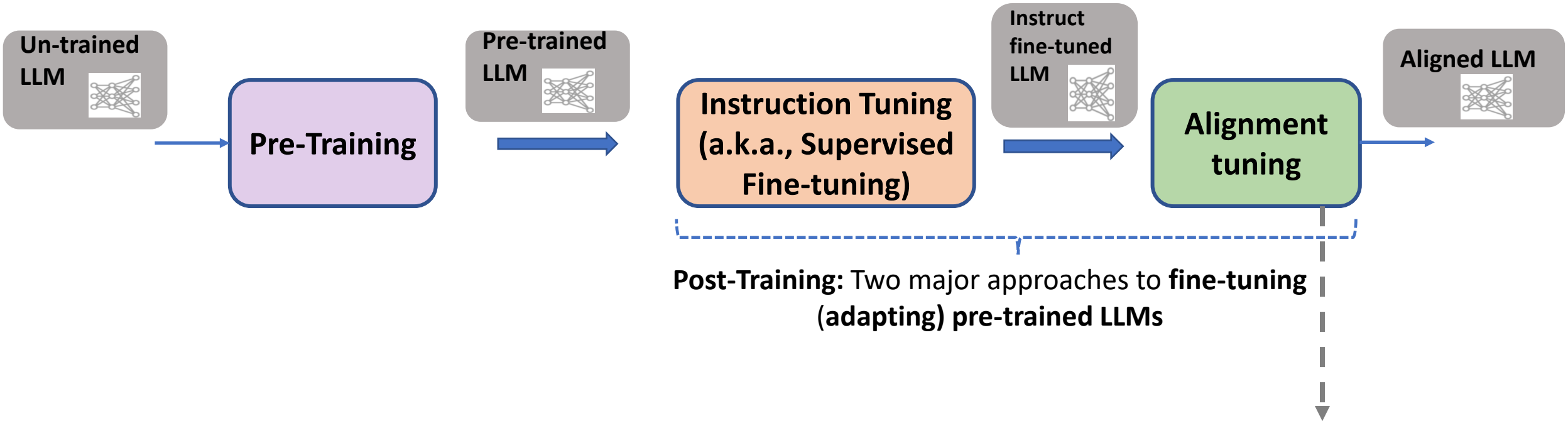
- An agent's workflow can be conceptually understood as a sequence of the following steps:

1. **User Input**: The user initiates interaction with the agent by providing a request or instruction.
2. **LLM interpretation**: The LLM analyzes the user input to understand the intent and desired outcome of the request.
3. **Planning**: Based on the LLM's interpretation, the agent creates a plan of action. This plan may involve either directly generating a response for the user or using one or more tools to gather additional information or perform specific actions.
4. **Action** (optional): If the plan involves the use of tools, the agent executes the necessary actions. These tools could be anything from calculators and code interpreters to search engines and database queries. The results of these actions are then fed back into the system.
5. **Memory update**: The agent updates its internal memory with the outcomes of any actions taken and any other pertinent information gathered during the process. This updated memory serves as a context for subsequent interactions.
6. **Response**: Finally, the agent formulates a response for the user. This response is based on the cumulative information obtained through the previous steps, including the LLM's interpretation, the results of any tool-based actions, and the updated memory.

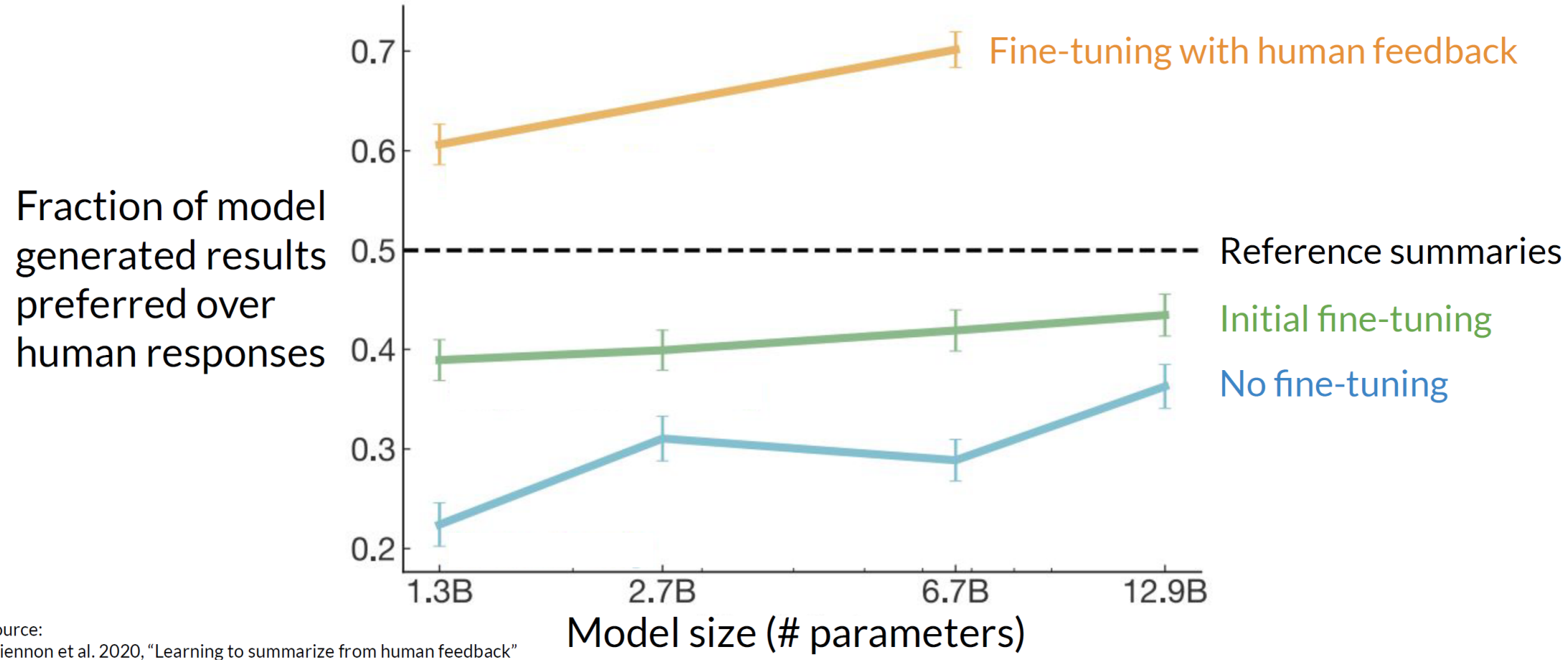
Alignment Tuning

Refresher: LLM Training stages at a high level

- Different types of training:
  Large Language Models typically undergo multiple training stages:



Fine-tuning with human feedback



Source:
Stiennon et al. 2020, "Learning to summarize from human feedback"

Alignment tuning Techniques

- **Reinforcement Learning from Human Feedback (RLHF)**
  - RLHF has achieved great success in aligning the behaviors of LLMs with human values and preferences
  - RLHF needs to train multiple LMs including:
    - The model being aligned,
    - The reward model,
    - and the reference model at the same time.

- **As an alternative => Alignment without RLHF (non-RL alignment)**
  - Directly optimize LLMs to adhere to human preferences, using supervised fine-tuning without reinforcement learning
    - Example of methods: **Direct Preference Optimization (DPO)**

DPO fine-tuning

Direct Preference Optimization:
Your Language Model is Secretly a Reward Model

Rafael Rafailov*† Archit Sharma*† Eric Mitchell*†

Stefano Ermon†‡ Christopher D. Manning† Chelsea Finn†

†Stanford University ‡CZ Biohub

29 Jul 2024

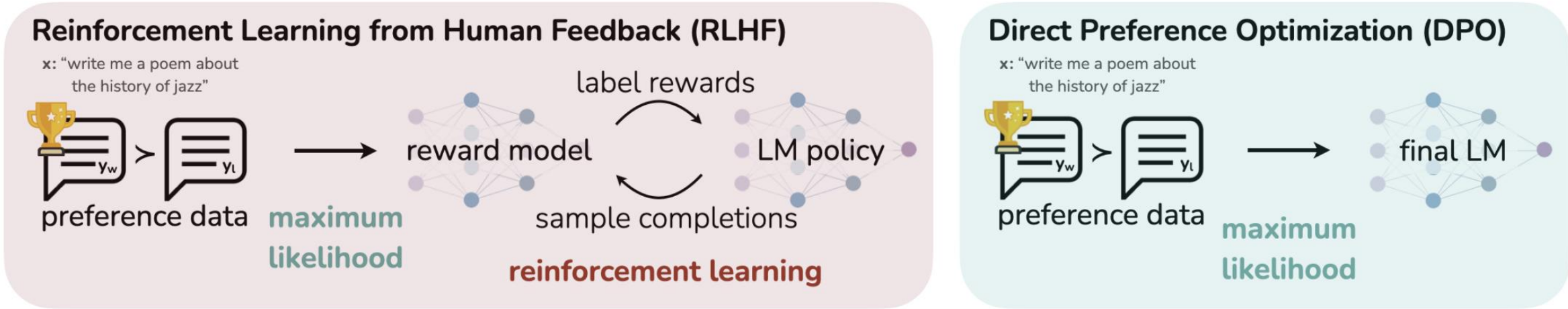
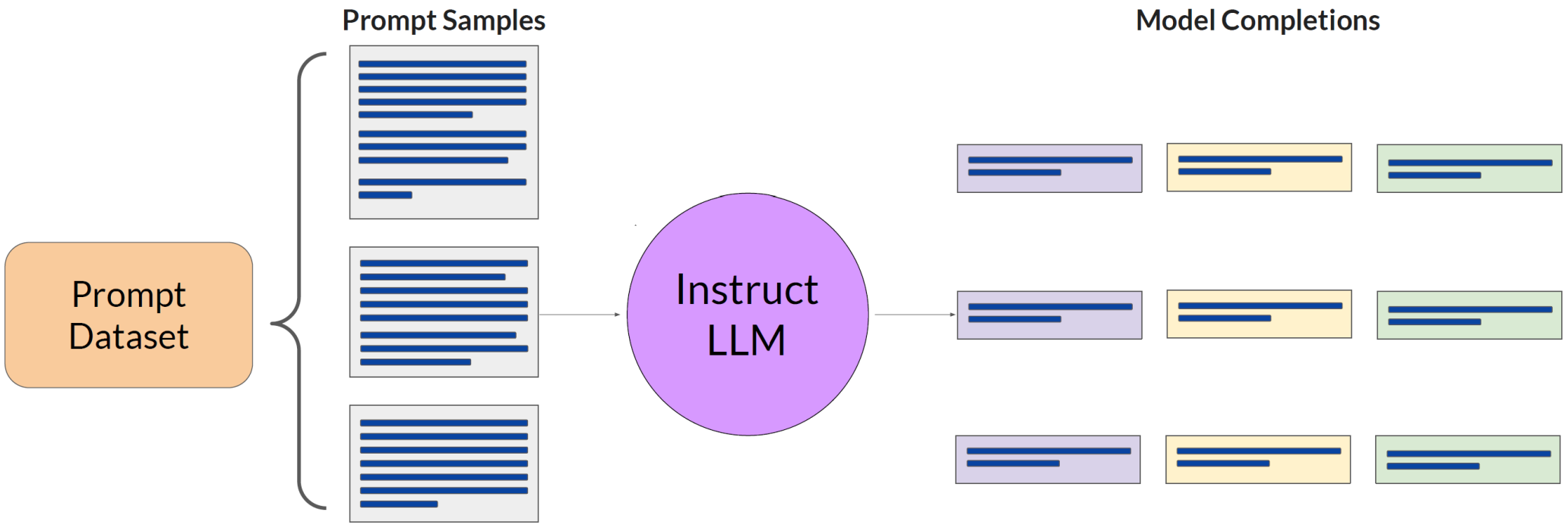


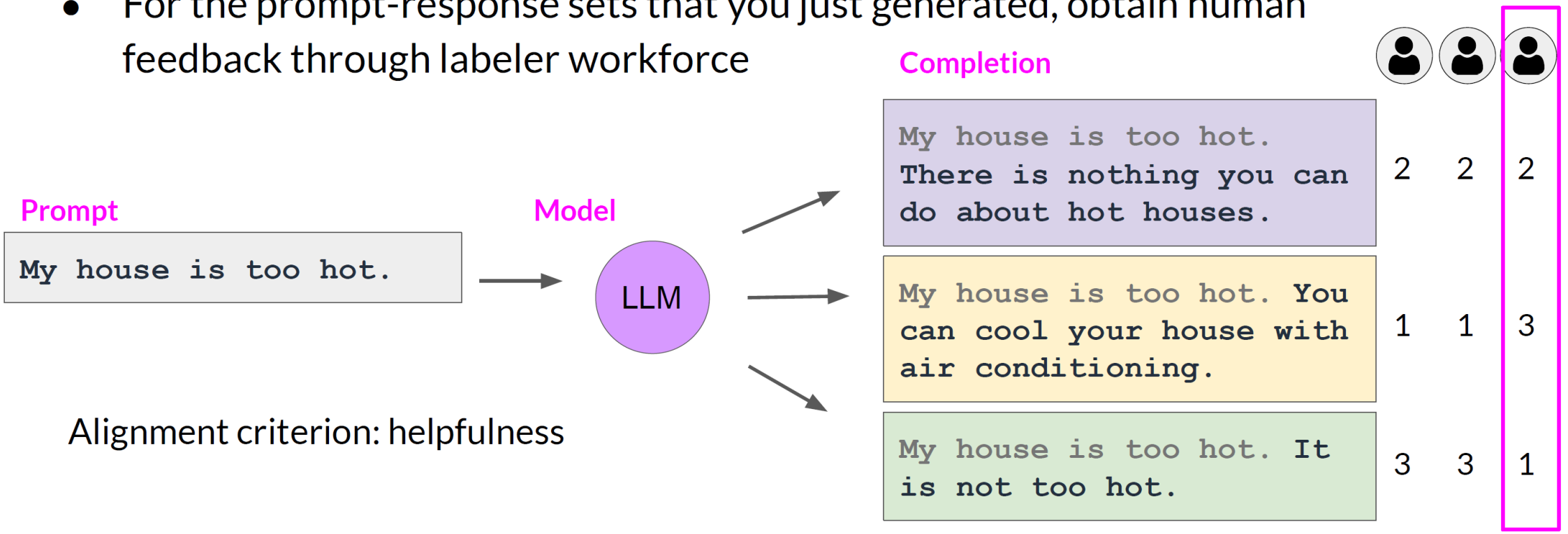
Figure 1: **DPO optimizes for human preferences while avoiding reinforcement learning.** Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, fitting an *implicit* reward model whose corresponding optimal policy can be extracted in closed form.

Prepare dataset for human feedback



Collect human feedback

- Define your model alignment criterion
- For the prompt-response sets that you just generated, obtain human feedback through labeler workforce



Sample instructions for human labelers

* Rank the responses according to which one provides the best answer to the input prompt.

* What is the best answer? Make a decision based on (a) the correctness of the answer, and (b) the informativeness of the response. For (a) you are allowed to search the web. Overall, use your best judgment to rank answers based on being the most useful response, which we define as one which is at least somewhat correct, and minimally informative about what the prompt is asking for.

* If two responses provide the same correctness and informativeness by your judgment, and there is no clear winner, you may rank them the same, but please only use this sparingly.

* If the answer for a given response is nonsensical, irrelevant, highly ungrammatical/confusing, or does not clearly respond to the given prompt, label it with "F" (for fail) rather than its rank.

* Long answers are not always the best. Answers which provide succinct, coherent responses may be better than longer ones, if they are at least as correct and informative.

Source: Chung et al. 2022, "Scaling Instruction-Finetuned Language Models"

Human Labeler Selection

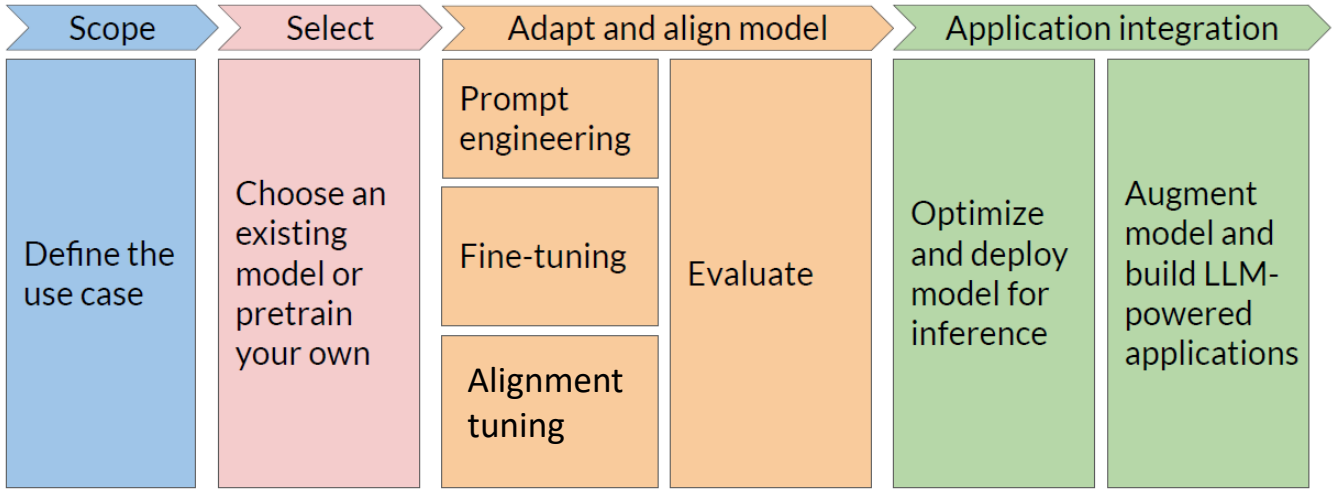
- In existing work, the dominant method for generating human feedback data is human annotation [66, 116, 365].

- This highlights the critical role of selecting appropriate human labelers. To provide high quality feedback, human labelers are supposed to have a **qualified level of education and excellent proficiency in Natural language**.

- For example, Sparrow [116] requires human labelers to be **UK-based native English speakers** who have obtained at least an **undergraduate-level educational qualification**.

Generative AI project lifecycle involving LLMs: Techniques and considerations at each stage

The overall lifecycle 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.
  - Direct Preference Optimization (DPO)
- *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

Model Compression

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

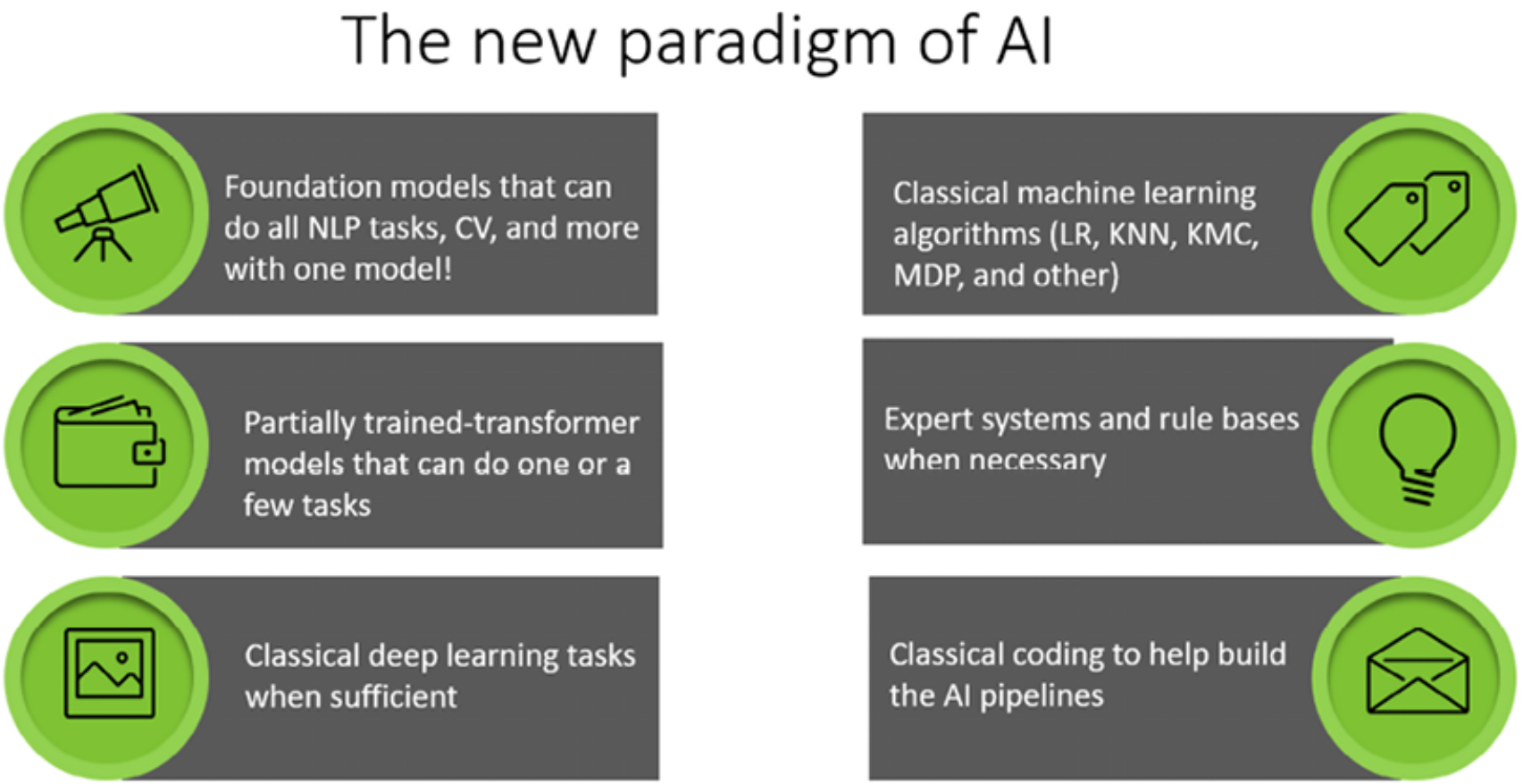
- Quantization Methods for LLMs, …
- *Quantized LLMs*: Compared with original models, quantized language models take a smaller memory footprint and likely have a faster inference speed.

LLM-based Agent

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

E. Nfaoui



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!

E. Nfaoui

Practical use in Academia

IEEE Author Center
Resources & Tools for Authors

Guidelines for Artificial Intelligence (AI)-Generated Text

The use of content generated by artificial intelligence (AI) in a paper (including but not limited to text, figures, images, and code) shall be disclosed in the acknowledgments section of any paper submitted to an IEEE publication. The AI system used shall be identified, and specific sections of the paper that use AI-generated content shall be identified and accompanied by a brief explanation regarding the level at which the AI system was used to generate the content.

The use of AI systems for editing and grammar enhancement is common practice and, as such, is generally outside the intent of the above policy. In this case, disclosure as noted above is not required, but recommended.

Conclusion

- Large Language Models (LLMs) are rapidly transforming many aspects of our lives. **Gaining a solid understanding of these technologies is becoming essential.**

- Whether or not you are a supporter of Generative AI, it is clear that it’s here to stay. Instead of resisting this shift, consider embracing it. Learning how to use these tools effectively will help you **stay current**, **remain competitive**, and **increase** your productivity.

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
- Wang, L., Ma, C., Feng, X. et al. A survey on large language model based autonomous agents. Front. Comput. Sci. 18, 186345 (2024). https://doi.org/10.1007/s11704-024-40231-1

  - 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)
- Pandit, B., 2024, What Is Mixture of Experts (MoE)? How It Works, Use Cases & More. Available at: https://www.datacamp.com/blog/mixture-of-experts-moe

- L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J. Wen, “A survey on large language model based autonomous agents,” CoRR, vol. abs/2308.11432, 2023.

- S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach (4th Edition). Pearson, 2020.

[Online]. Available: http://aima.cs.berkeley.edu/

- B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, “Building machines that learn and think like people,” CoRR, vol. abs/1604.00289, 2016.

Thank you for your attention