Large Language Models: Overview

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

جامعة سيدي محمد بن عبد الله بفاس
UNIVERSITE SIDI MOHAMED BEN ABDELLAH DE FES
كلية العلوم ظهر المهراز فاس
FACULTE DES SCIENCES DHAR EL MAHRAZ FES

Outline

A. Generative AI, Foundation models, and Transformers
B. Large language models (LLMs): overview
C. Transformers: high-level overview
D. Generative AI project lifecycle

A. Generative AI, Foundation models, and Transformers

- **Generative AI refers to a subfield of artificial intelligence** that involves creating (generating) new content or data from a given set of inputs, often using techniques such as deep learning and neural networks.
- Generative AI models (such as ChatGPT) can be trained to learn patterns in large datasets and generate new content, including text, images, music, and even video that is both novel and coherent.
- Deep learning models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), enabled AI systems to generate highly realistic and complex outputs, such as photorealistic images and natural language text.
- Generative AI has a significant impact on various industries and businesses. For example, in the entertainment industry, generative AI can be used to create new and unique content, such as music, movies, and video games.

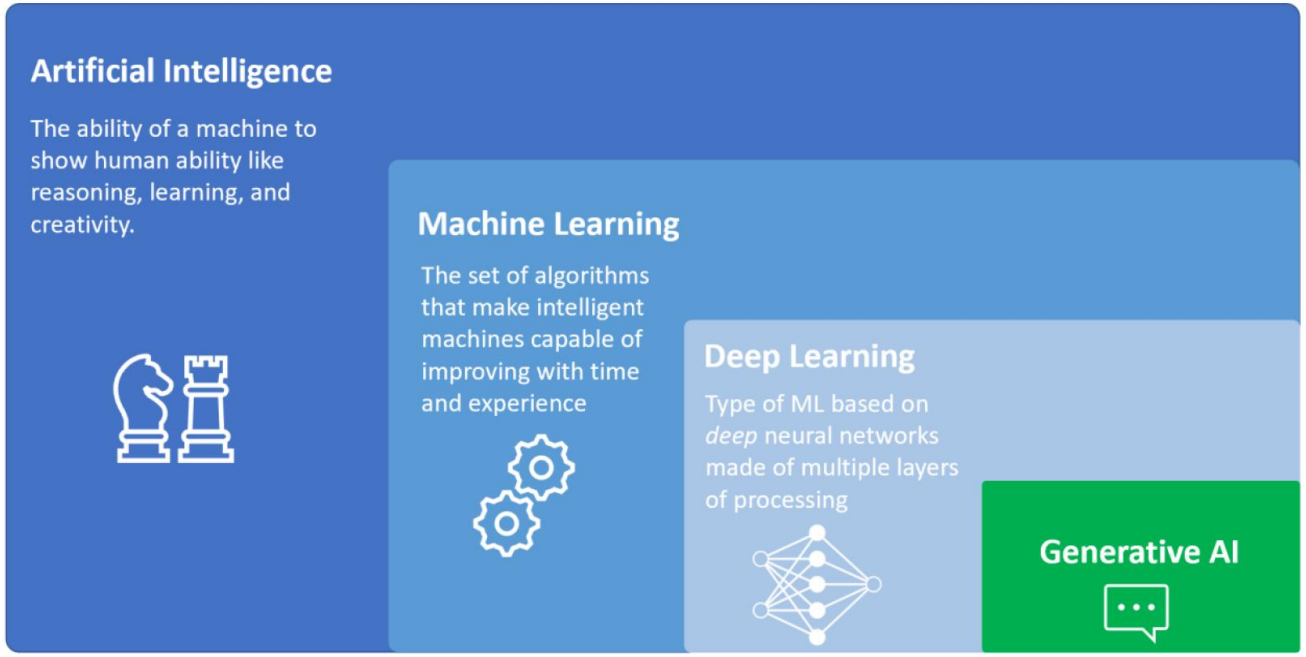
A. Generative AI, Foundation models, and Transformers

In order to better understand the relationship between Artificial Intelligence (AI), Machine Learning (ML), Deep Learning(DL), and Generative AI, consider AI as the foundation, while ML, DL, and generative AI represent increasingly specialized and focused areas of study and application:

- **AI** represents the broad field of creating systems that can perform tasks, showing human intelligence and ability and being able to interact with the ecosystem.
- **ML** is a branch that focuses on creating algorithms and models that enable those systems to learn and improve themselves with time and training. ML models learn from existing data and automatically update their parameters as they grow.
- **DL** is a sub-branch of ML, in the sense that it encompasses deep ML models. Those deep models are called neural networks and are particularly suitable in domains such as computer vision or Natural Language Processing (NLP). When we talk about ML and DL models, we typically refer to discriminative models, whose aim is that of making predictions or inferencing patterns on top of data.
- **Generative AI** is a further sub-branch of DL, which doesn't use deep Neural Networks to cluster, classify, or make predictions on existing data: it uses those powerful Neural Network models to generate brand new content, from images to natural language, from music to video.

A. Generative AI, Foundation models, and Transformers

The following figure shows how these areas of research are related to each other:



Relationship between AI, ML, DL, and generative AI (Valentina Alto, 2023)

Generative AI models can be trained on vast amounts of data and then they can generate new examples from scratch using patterns in that data. This **generative process** is different from **discriminative models**, which are trained to predict the class or label of a given example.

A. Generative AI, Foundation models, and Transformers

- **Generative AI** lets you create new content and ideas including conversations, stories, images, videos, and music. Generative AI is powered by **very large machine learning models** that are **pre-trained on vast amounts of data**, commonly referred to as **foundation models (FMs)** (Bommansani et al., 2021).

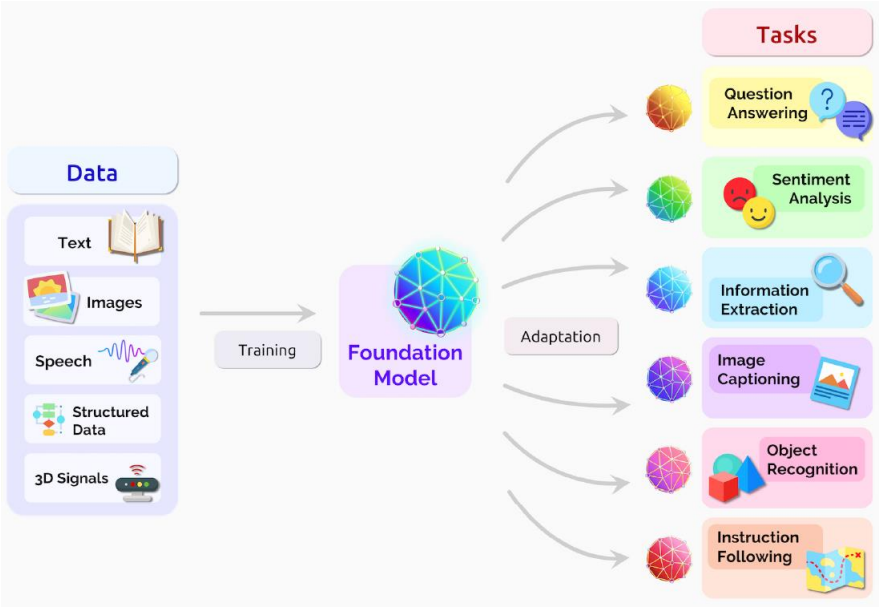
- **Foundation models** are fully **trained transformer models** that can carry out hundreds of tasks without fine-tuning. Foundation models at this scale offer the tools we need in this **massive information era**.

1. What's a Transformer Model?

- A **transformer model** is a **deep learning model** that learns context and thus meaning by tracking relationships in sequential data (***text, image, sound, video, etc.).***
- A transformer learns sequences. These sequences include natural language, vision, sound, and any type of data represented as a sequence.
- The Transformer architecture (Vaswani et al., 2017) has revolutionized the learning paradigm across a wide range of **supervised learning tasks (**Devlin et al., 2018; Dosovitskiy et al., 2020; Dong et al., 2018) and demonstrated superior performance over CNN and RNN. Among its notable benefits, the transformer architecture enables modeling long dependencies and has excellent scalability (Khan et al., 2022).
- Transformer models apply an evolving set of mathematical techniques, called attention or self-attention concepts, to detect subtle ways even distant data elements in a series influence and depend on each other. The self part of self-attention refers to the "egocentric" focus of each token in a corpus.
- Stanford researchers called transformers "foundation models" in their paper titled "*On the Opportunities and Risks of Foundation Models*" (Bommansani et al., 2021) because they see them driving a paradigm shift in AI. The "sheer scale and scope of foundation models over the last few years have stretched our imagination of what is possible," they wrote. This paper was written by over one hundred scientists and professionals from CRFM (Center for Research on Foundation Models).

2. What can Transformer Models do?

Transformers, called foundation models, are already being used with many data sources for a host of applications.



A foundation model can centralize the information from all the data from various modalities. This one model can then be adapted to a wide range of downstream tasks (Bommansani et al., 2021)

Foundation models have two distinct and unique properties:

- **Emergence** – Transformer models that qualify as foundation models can perform tasks **they were not trained for**. They are large models trained on supercomputers. They **are not trained to learn specific tasks like many other models**. Foundation models **learn how to understand sequences**. Transformers have uncanny sequence analysis abilities.
- **Homogenization** – The same model can be used across many domains with the same fundamental architecture. Foundation models can learn new skills through data faster and better than any other model.

These powerful foundation models prove that transformers are *task-agnostic (e.g.* GPT-3 is a task-agnostic foundation model). A transformer learns sequences. These sequences include vision, sound, and any type of data represented as a sequence.

3. Applications of Transformers

Transformer (Vaswani et al., 2017) was originally designed for machine translation but has been widely adopted in various fields besides NLP, including computer vision (CV) and audio processing, because of its flexible architecture. These models support common tasks in different modalities, such as:

📝 Natural Language Processing:

Transformer and its variants have been extensively explored and applied in NLP tasks, e.g., machine translation, language modeling, named entity recognition, text classification, question answering, summarization, translation, multiple choice, text generation, etc.

A massive effort has been dedicated to pre-training Transformer models on large-scale text corpora, which we believe is one of the major reasons of Transformer's wide application in NLP.

🖼 Computer Vision:

Transformers have also been adapted for various vision tasks, e.g., image segmentation, image classification, object detection, image generation, and video processing.

3. Applications of Transformers *(continued)*

🗣️ Audio applications

- Automatic speech recognition, speech synthesis, speech enhancement, audio classification, music generation, etc.

🐙 Multimodal applications

- Owing to its flexible architecture, Transformer has also been applied in various multimodal scenarios, e.g., visual question answering, visual commonsense reasoning, caption generation, speech-to-text translation, text-to-image generation, Optical character recognition, information extraction from scanned documents, etc.

- Transformer has also been adopted in other disciplines, such as chemistry (Schwaller et al., 2019) and life sciences (Rives et al., 2021).

4. No Labels, More Performance

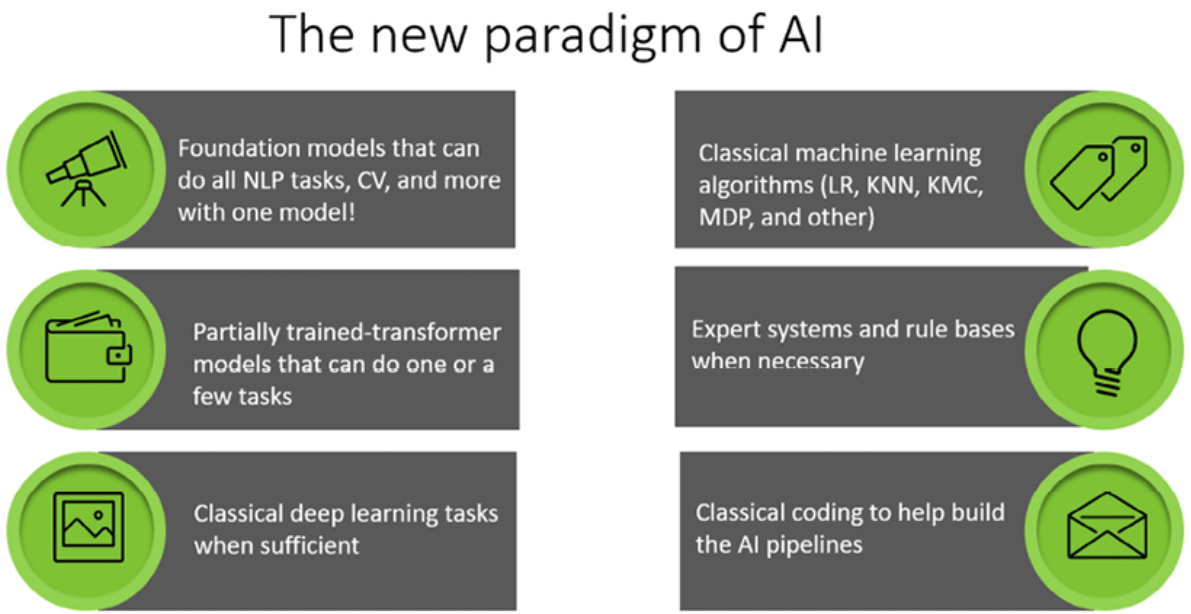
- Before transformers arrived, users had to train neural networks with large, labeled datasets that were costly and time-consuming to produce. **Transformers eliminate that need, making available the trillions of images and petabytes of text data on the web and in corporate databases**. Transformers can perform self-supervised learning on billions of records of raw unlabeled data with billions of parameters.
- In addition, the attention or self-attention concepts used by transformers lend to parallel processing, so these models can run fast.
- Transformers now dominate popular performance leaderboards like SuperGLUE, a benchmark developed in 2019 for language-processing systems.

- Ever since the advent of the transformer, it has **replaced convolutional and recurrent neural networks (CNNs and RNNs)** for various tasks. Indeed, 70 percent of arXiv papers on AI posted between 2020 and 2022 years mention transformers*. That's a radical shift from a 2017 IEEE study that reported RNNs and CNNs were the most popular models for pattern recognition.

* Blog NVIDIA, March 25, 2022, by RICK MERRITT, https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/ .

5. The ecosystem of transformers

- Transformer models (such as OpenAI GPT, PaLM, Galactica, etc.) are industrialized, and homogenized post-deep learning models designed for parallel computing on supercomputers. Through homogenization, **one transformer model can carry out a wide range of tasks with no fine-tuning**. Transformers can perform **self-supervised learning on billions of records of raw unlabeled data with billions of parameters**.



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!

5. The ecosystem of transformers

The present ecosystem of transformer models can be summed up with four properties:

1. **Model architecture**

The model is industrial. The layers of the model are specifically designed for parallel processing.

2. **Data**

Big tech possesses the hugest data source in the history of humanity, first generated by the Third Industrial Revolution (digital) and boosted to unfathomable sizes by Industry 4.0.

3. **Computing power**

Big tech possesses computer power. For example, GPT-3 was trained at about 50 PetaFLOPS/second.

4. **Prompt engineering**

Highly trained transformers can be triggered to do a task with a **prompt**. The prompt is entered in natural language. It involves instructions and context passed to a transformer to achieve a desired task.

For example, GPT-3 can convert natural language into source code; however, you might not generate what you expect if you are not careful to engineer the prompt correctly. With better prompting we can achieve much better results and potentially not need to be fine-tune.

We will replace “input” with “prompt” for transformers to show that our input influences how the model will react. The output is the “response.”

A foundation model is thus a transformer model that has been trained on supercomputers on billions of records of data and billions of parameters. **The model can then perform a wide range of tasks with no further fine-tuning**. Thus, the scale of foundation models is unique.

Outline

A. Generative AI, Foundation models, and Transformers
B. Large language models (LLMs): overview
C. Transformers: high-level overview
D. Generative AI project lifecycle

B. Large language models (LLMs): overview

- 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**.
  - These LLMs can understand, learn, and generate text that's nearly indistinguishable from text produced by humans.
  - LLMs can also engage in interactive conversations, answer questions, summarize dialogs and documents, and provide recommendations.
  - They can power applications across many tasks and industries including creative writing for marketing, summarizing documents for legal, market research for financial, simulating clinical trials for healthcare, and code writing for software development.
  - And more.
- Companies are moving rapidly to integrate generative AI into their products and services. This increases the demand for data scientists and engineers who understand generative AI and how to apply LLMs to solve business use cases and deploy them for real-world applications.

1. Definition of LLM

- In (Wayne Xin Zhao et al., 2023), the authors defined an LLM as follows:

" 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 (M. Shanahan, 2022), such as GPT-3 (Brown et al, 2020), PaLM (A. Chowdhery et al., 2022), Galactica (R. Taylor et al., 2022), and LLaMA (H. Touvron et al., 2023). LLMs exhibit strong capacities to understand natural language and solve complex tasks (via text generation)."

- Large language models (LLMs) are very large deep learning models that are pre-trained on vast amounts of data, and have a massive number of parameters, allowing them to **learn complex patterns in language.**
- 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 such as OpenAI GPT-3 have taken emergence to another level. Large language models can perform **hundreds of NLP tasks they were not trained for**.

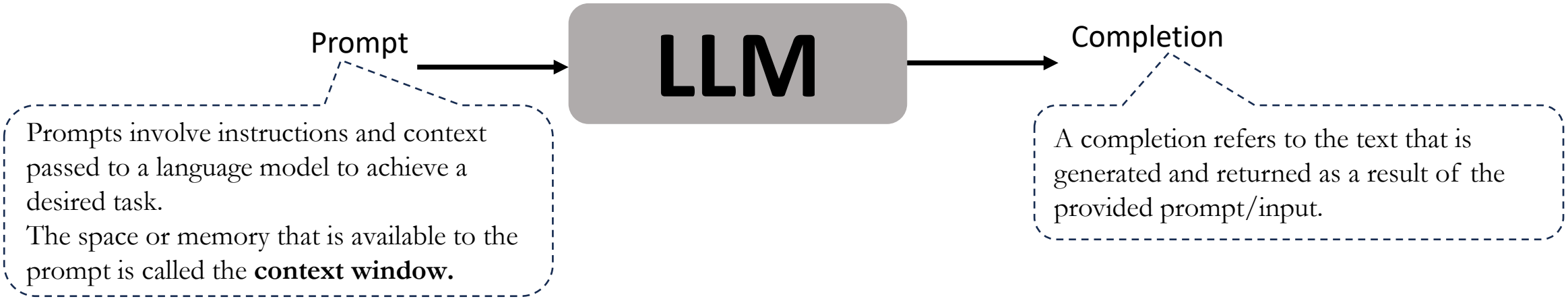
How large is large?

Parameters are the **weights** the model learned during training, used to predict the next token in the sequence. "Large" can refer either to the number of parameters in the model, or sometimes the number of words in the dataset.

* In their LLMs Survey paper, the authors stated the following: " In existing literature, there is no formal consensus on the minimum parameter scale for LLMs, since the model capacity is also related to data size and total compute. In this survey, we take a slightly loose definition of LLMs, and mainly focus on discussing language models with a model size larger than 10B."

2. Prompt and completion

- Large language models are able to take natural language or human written instructions and perform tasks much as a human would.
  - The text that you pass to an LLM is known as a **prompt**. The space or memory that is available to the prompt is called the **context window** and differs from model to model.
  - The output of the model is called a **completion**, and the act of using the model to generate text is known as inference.



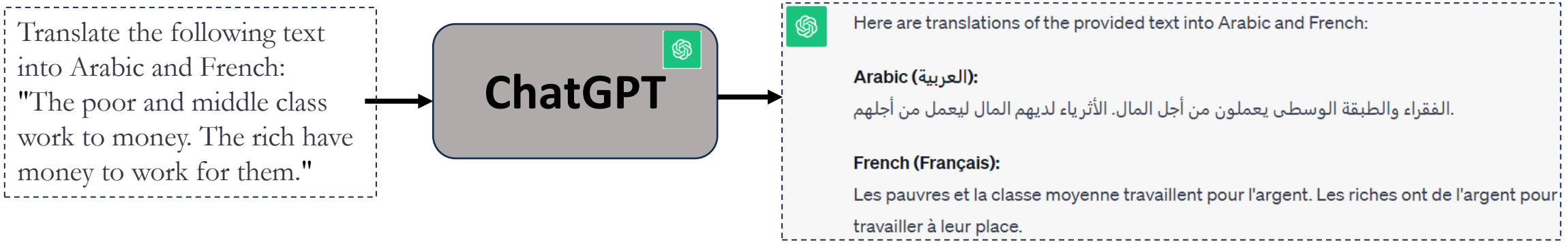
2. Prompt and completion

- The size of the context window (or context size for short) of an LLM is the number of **tokens** the model can take as input when generating a response (completion). 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.)

- LLMs interpret the textual data as tokens. **Tokens** are words or **chunks of characters**. For example, the word "**pretraining**" would be broken down into the tokens "**pre**" and "**training**", whereas common words like "**time**" and "**like**" would be a single token. Tokens are produced from prompts using a tokenizer algorithm (e.g. Subword tokenization algorithms, WordPiece tokenizer, etc.)

  You can use the tool below to understand how a piece of text might be tokenized by GPT language models tokenizers: https://platform.openai.com/tokenizer

2. Prompt and completion



3. Pre-trained Language Models sizes

- The following bubble chart explains the recent developments of heavy load models with large parameters in language model.

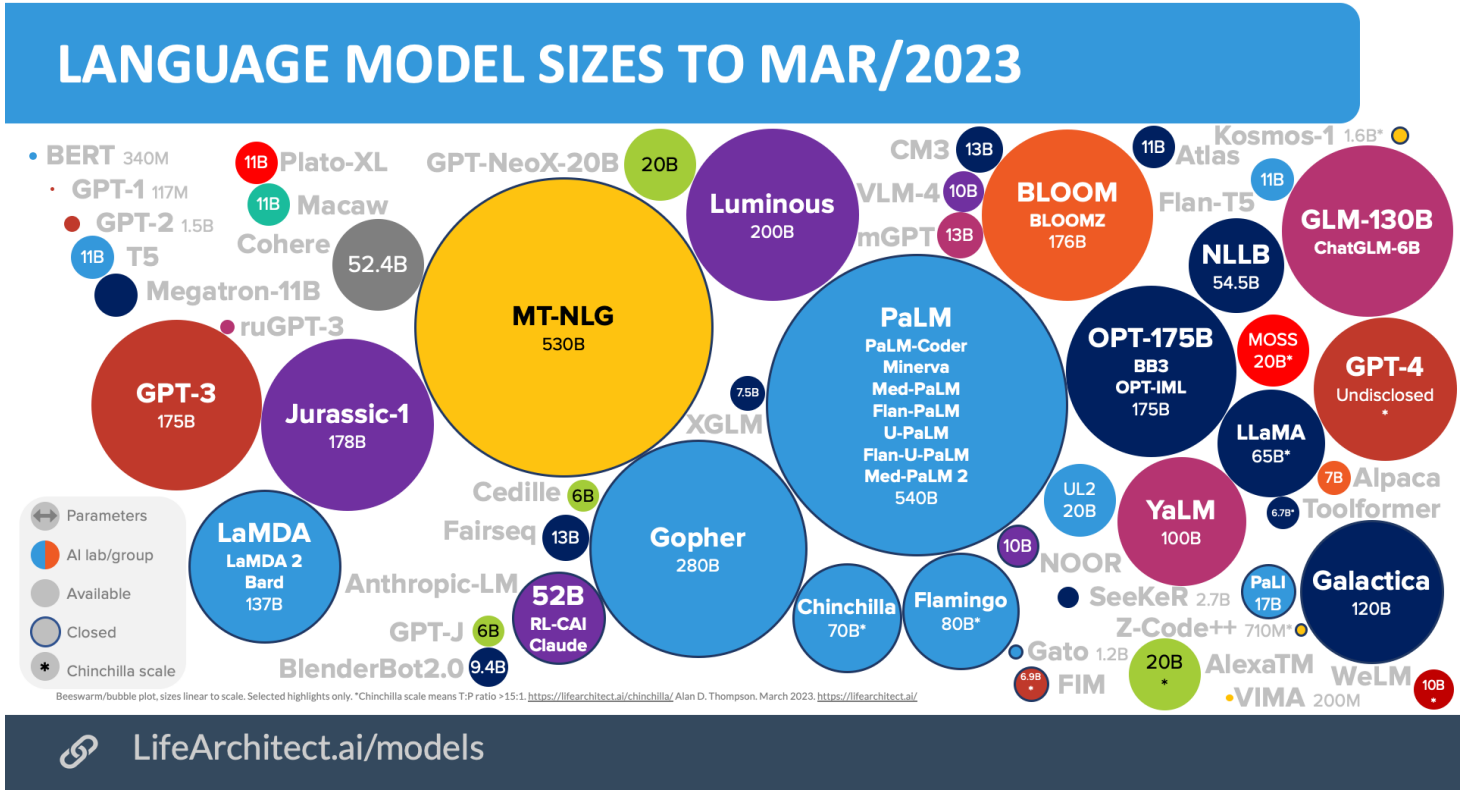


Chart. Major AI language models 2018-2023, GPT-3 on the left, GPT-4 on the right in red.

Large Language Models (larger than 10B) (Wayne Xin Zhao et al., 2023)

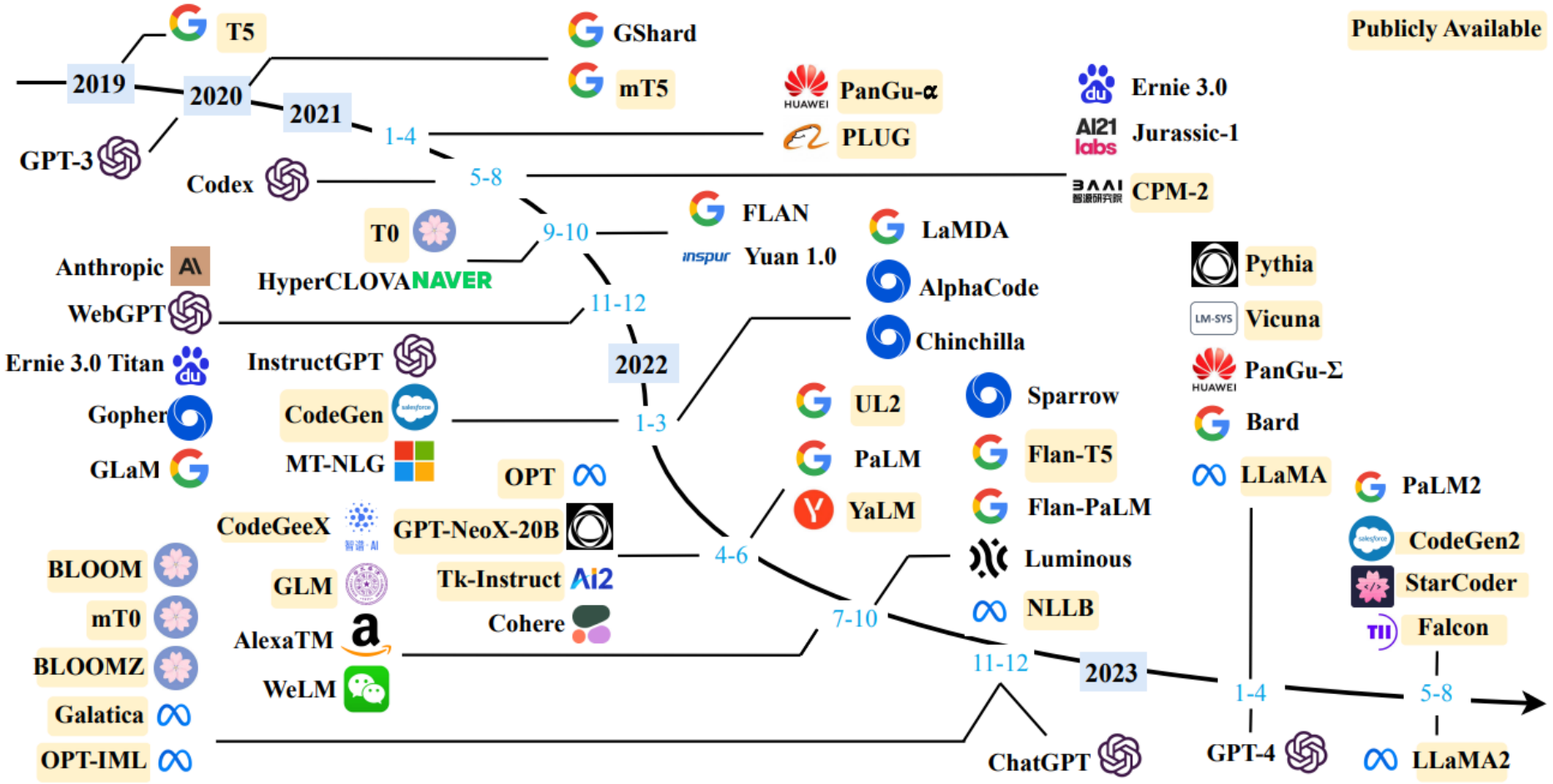


Fig. 2: A timeline of existing large language models (having a size larger than 10B) in recent years. The timeline was established mainly according to the release date (*e.g.*, the submission date to arXiv) of the technical paper for a model. If there was not a corresponding paper, we set the date of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

Large Language Models (larger than 10B)) (Wayne Xin Zhao et al., 2023)

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

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

Large Language Models (larger than 10B)) (Wayne Xin Zhao et al., 2023)

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

Statistics of Commonly-used Data Sources for LLMs

TABLE 2: Statistics of commonly-used data sources.(Wayne Xin Zhao et al., 2023)

Corpora	Size	Source	Latest Update Time
BookCorpus [138]	5GB	Books	Dec-2015
Gutenberg [139]	-	Books	Dec-2021
C4 [73]	800GB	CommonCrawl	Apr-2019
CC-Stories-R [140]	31GB	CommonCrawl	Sep-2019
CC-NEWS [27]	78GB	CommonCrawl	Feb-2019
REALNEWs [141]	120GB	CommonCrawl	Apr-2019
OpenWebText [142]	38GB	Reddit links	Mar-2023
Pushift.io [143]	2TB	Reddit links	Mar-2023
Wikipedia [144]	21GB	Wikipedia	Mar-2023
BigQuery [145]	-	Codes	Mar-2023
the Pile [146]	800GB	Other	Dec-2020
ROOTS [147]	1.6TB	Other	Jun-2022

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

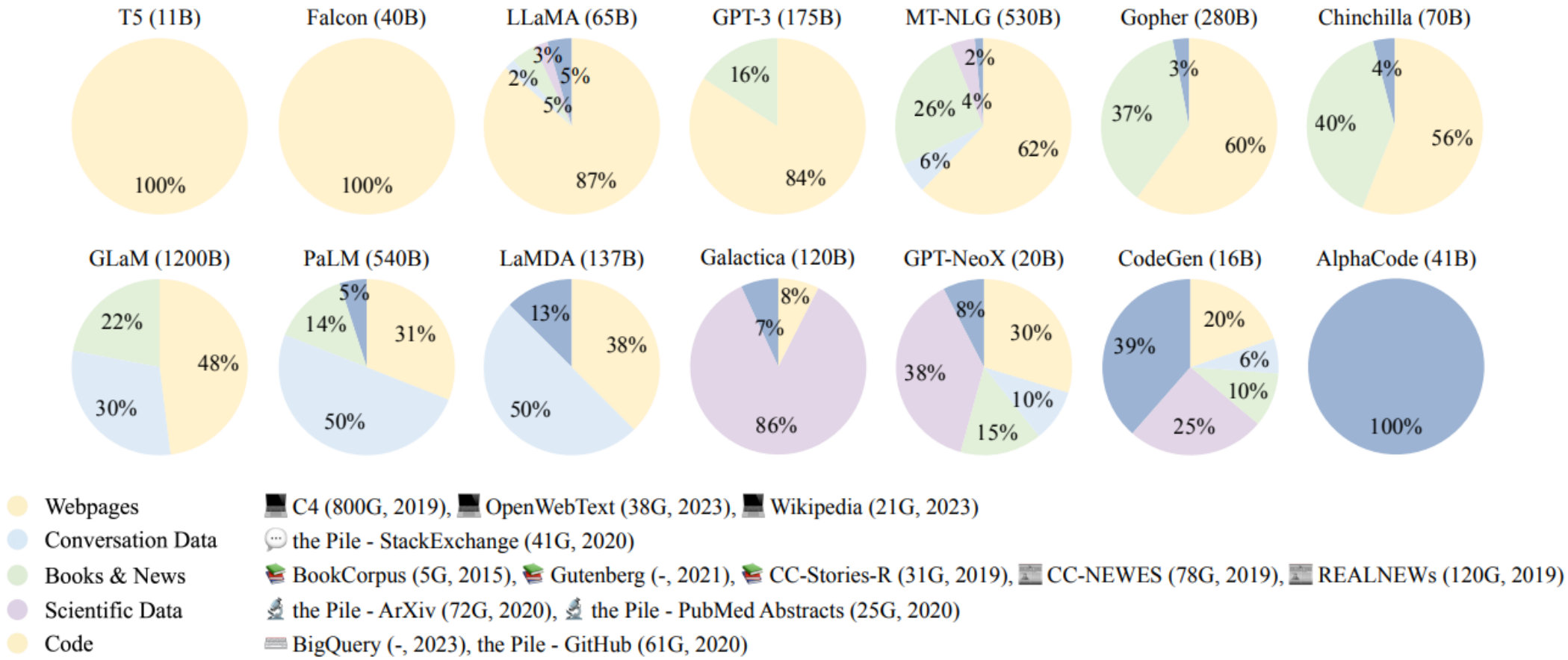


Fig. 5: Ratios of various data sources in the pre-training data for existing LLMs.(Wayne Xin Zhao et al., 2023)

Typical Data Preprocessing Pipeline for Pre-training Large Language Models (Wayne Xin Zhao et al., 2023)

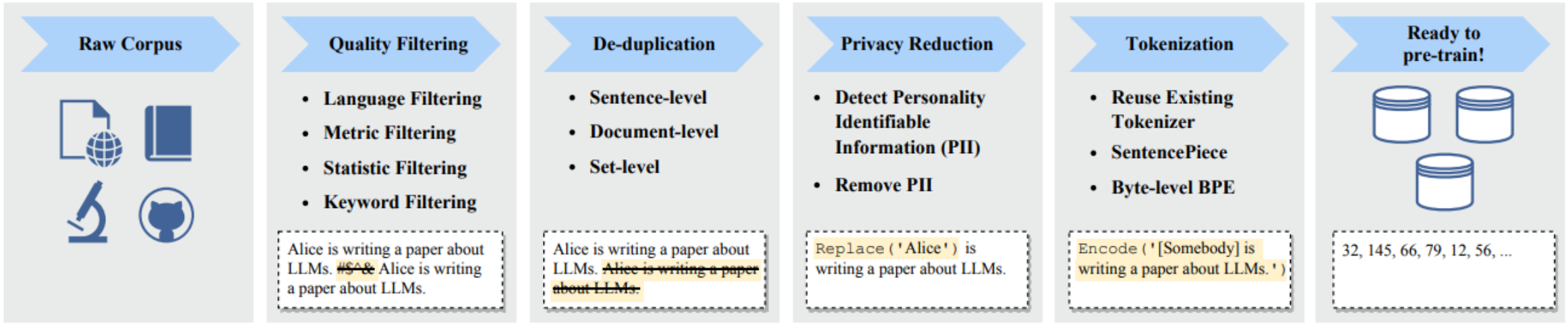


Fig. 6: An illustration of a typical data preprocessing pipeline for pre-training large language models. (Wayne Xin Zhao et al., 2023)

Model cards of several selected LLMs with public configuration details (Wayne Xin Zhao et al., 2023)

TABLE 3: Model cards of several selected LLMs with public configuration details. Here, PE denotes position embedding, #L denotes the number of layers, #H denotes the number of attention heads, d_{model} denotes the size of hidden states, and MCL denotes the maximum context length during training.

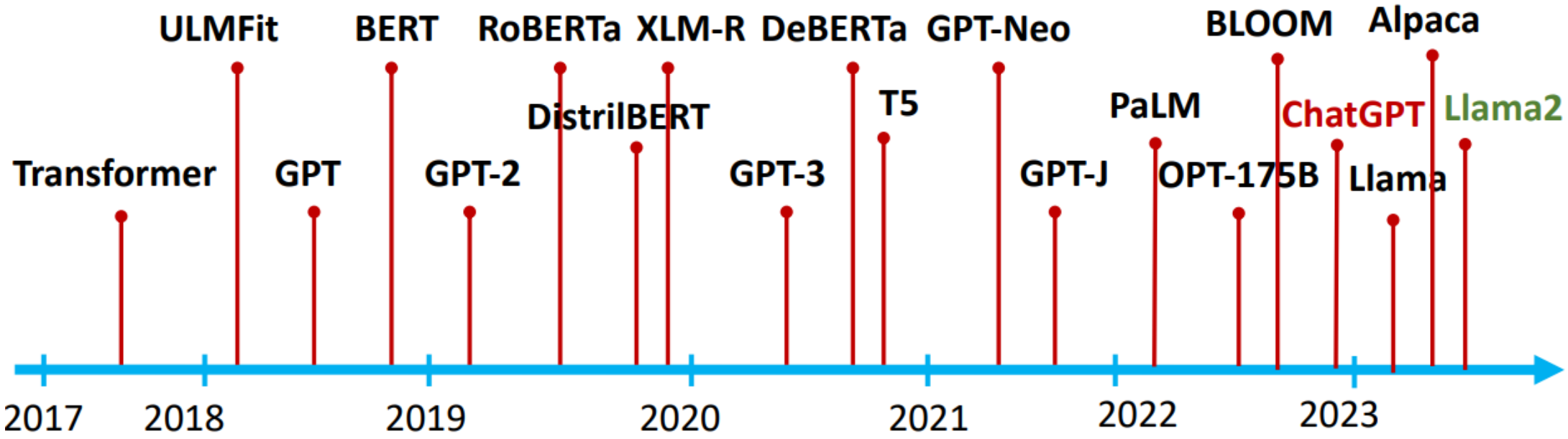
Model	Category	Size	Normalization	PE	Activation	Bias	#L	#H	d_{model}	MCL
GPT3 [55]	Causal decoder	175B	Pre LayerNorm	Learned	GeLU	✓	96	96	12288	2048
PanGU- α [75]	Causal decoder	207B	Pre LayerNorm	Learned	GeLU	✓	64	128	16384	1024
OPT [81]	Causal decoder	175B	Pre LayerNorm	Learned	ReLU	✓	96	96	12288	2048
PaLM [56]	Causal decoder	540B	Pre LayerNorm	RoPE	SwiGLU	×	118	48	18432	2048
BLOOM [69]	Causal decoder	176B	Pre LayerNorm	ALiBi	GeLU	✓	70	112	14336	2048
MT-NLG [100]	Causal decoder	530B	-	-	-	-	105	128	20480	2048
Gopher [59]	Causal decoder	280B	Pre RMSNorm	Relative	-	-	80	128	16384	2048
Chinchilla [34]	Causal decoder	70B	Pre RMSNorm	Relative	-	-	80	64	8192	-
Galactica [35]	Causal decoder	120B	Pre LayerNorm	Learned	GeLU	×	96	80	10240	2048
LaMDA [63]	Causal decoder	137B	-	Relative	GeGLU	-	64	128	8192	-
Jurassic-1 [94]	Causal decoder	178B	Pre LayerNorm	Learned	GeLU	✓	76	96	13824	2048
LLaMA [57]	Causal decoder	65B	Pre RMSNorm	RoPE	SwiGLU	×	80	64	8192	2048
LLaMA 2 [90]	Causal decoder	70B	Pre RMSNorm	RePE	SwiGLU	×	80	64	8192	4096
Falcon [127]	Causal decoder	40B	Pre LayerNorm	RoPE	GeLU	×	60	64	8192	2048
GLM-130B [84]	Prefix decoder	130B	Post DeepNorm	RoPE	GeGLU	✓	70	96	12288	2048
T5 [73]	Encoder-decoder	11B	Pre RMSNorm	Relative	ReLU	×	24	128	1024	512

Detailed optimization settings of several existing LLMs (Wayne Xin Zhao et al., 2023)

TABLE 5: Detailed optimization settings of several existing LLMs.

Model	Batch Size (#tokens)	Learning Rate	Warmup	Decay Method	Optimizer	Precision Type	Weight Decay	Grad Clip	Dropout
GPT3 (175B)	32K→3.2M	6×10^{-5}	yes	cosine decay to 10%	Adam	FP16	0.1	1.0	-
PanGu-α (200B)	-	2×10^{-5}	-	-	Adam	-	0.1	-	-
OPT (175B)	2M	1.2×10^{-4}	yes	manual decay	AdamW	FP16	0.1	-	0.1
PaLM (540B)	1M→4M	1×10^{-2}	no	inverse square root	Adafactor	BF16	lr^2	1.0	0.1
BLOOM (176B)	4M	6×10^{-5}	yes	cosine decay to 10%	Adam	BF16	0.1	1.0	0.0
MT-NLG (530B)	64 K→3.75M	5×10^{-5}	yes	cosine decay to 10%	Adam	BF16	0.1	1.0	-
Gopher (280B)	3M→6M	4×10^{-5}	yes	cosine decay to 10%	Adam	BF16	-	1.0	-
Chinchilla (70B)	1.5M→3M	1×10^{-4}	yes	cosine decay to 10%	AdamW	BF16	-	-	-
Galactica (120B)	2M	7×10^{-6}	yes	linear decay to 10%	AdamW	-	0.1	1.0	0.1
LaMDA (137B)	256K	-	-	-	-	BF16	-	-	-
Jurassic-1 (178B)	32 K→3.2M	6×10^{-5}	yes	-	-	-	-	-	-
LLaMA (65B)	4M	1.5×10^{-4}	yes	cosine decay to 10%	AdamW	-	0.1	1.0	-
LLaMA 2 (70B)	4M	1.5×10^{-4}	yes	cosine decay to 10%	AdamW	-	0.1	1.0	-
Falcon (40B)	2M	1.85×10^{-4}	yes	cosine decay to 10%	AdamW	BF16	0.1	-	-
GLM (130B)	0.4M→8.25M	8×10^{-5}	yes	cosine decay to 10%	AdamW	FP16	0.1	1.0	0.1
T5 (11B)	64K	1×10^{-2}	no	inverse square root	AdaFactor	-	-	-	0.1
ERNIE 3.0 Titan (260B)	-	1×10^{-4}	-	-	Adam	FP16	0.1	1.0	-
PanGu-Σ (1.085T)	0.5M	2×10^{-5}	yes	-	Adam	FP16	-	-	-

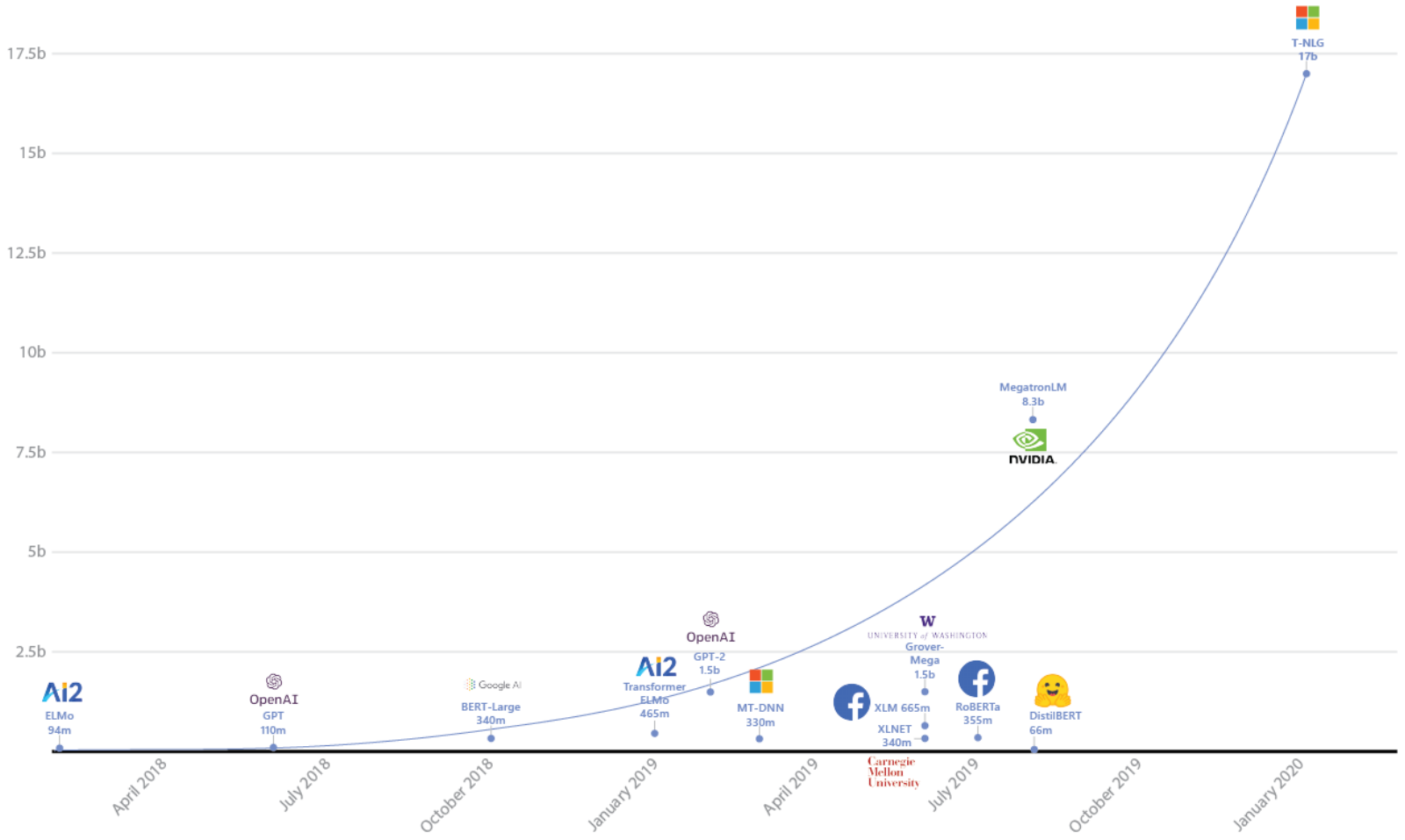
3. The Transformers Timeline



An overview of some of the most prominent transformer architectures – updated- (L. Tunstall et al., 2022)

3. Pre-trained Language Models sizes

Pre-trained Language Models sizes to January 2020:



Source: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

4. Types of Large Language Models

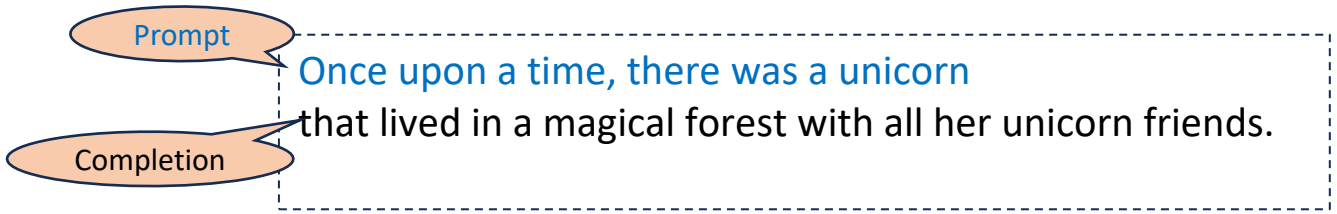
- There are two types of LLMs:
  - Base LLMs
  - Instruction-tuned LLMs

- To organize their discussion about existing evaluation approaches for assessing the performance of LLMs, the authors in (Wayne Xin Zhao et al., 2023) categorized LLMs into three different types: base LLMs (pretrained model checkpoints), fine-tuned LLMs (instruction or alignment fine-tuned model checkpoints), and specialized LLMs (adapted model checkpoints for some specific task or domain).

a. Base LLM

- A Base LLM has been trained to predict the next word based on text training data, often trained on a large amount of data from the Internet and other sources to figure out what's the next most likely word to follow.

For example, if the *prompt* is: *"Once upon a time there was a unicorn"*, a base LLM may predict the next several words *"that lived in a magical forest with all unicorn friends."* as a *completion*.



But if the prompt is "What is the capital of Japan ?", then it's possible that the base LLM will complete this with "What is Japan's largest city ?", " What is Japan's population?" and so on, because articles on the Internet could quite plausibly be lists of quiz questions about the country of Japan.

b. Instruction-tuned LLM

- An instruction-tuned LLM has been trained to follow instructions.

For example, if the prompt is: “What is the capital of Japan?”, it's much more likely to output something like, “The capital of Japan is Tokyo.”

Instruction-tuned LLMs are typically trained as follows:

- Start with a base LLM that has been trained on a huge amount of text data,
- Further fine-tune it with inputs and outputs that are instructions and good attempts to follow those instructions, and then often further refine using a technique called RLHF (Reinforcement Learning with Human Feedback) to make the system able to be helpful and follow instructions.

Instruction-tuned LLMs are less likely to output problematic text such as toxic outputs compared to base LLMs because they have been trained to be helpful, honest, and harmless. Because of the work of OpenAI and other LLM companies, LLMs are becoming safer and more aligned.

c. Examples: GPT-3, GPT-3.5, ChatGPT, Chinchilla, and Sparrow

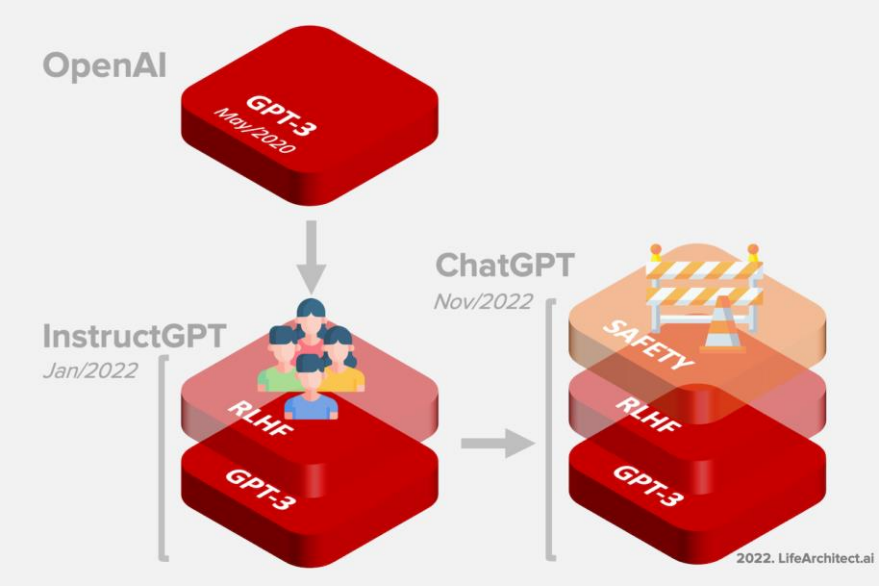
- GPT-3 was released in May 2020. At the time, the model was the largest publicly available, trained on 300 billion tokens (word fragments), with a final size of 175 billion parameters.

- GPT-3.5 or InstructGPT (Jan/2022)

GPT-3.5 is based on GPT-3, but works within guardrails, an early prototype of AI alignment with human values by forcing it to comply with policies.
Using GPT-3 as its base model, GPT-3.5 models use the same pre-training datasets as GPT-3, with additional fine-tuning. This fine-tuning stage adds a concept called 'Reinforcement Learning with Human Feedback (RLHF)' to the GPT-3 model.

- ChatGPT (Nov/2022)

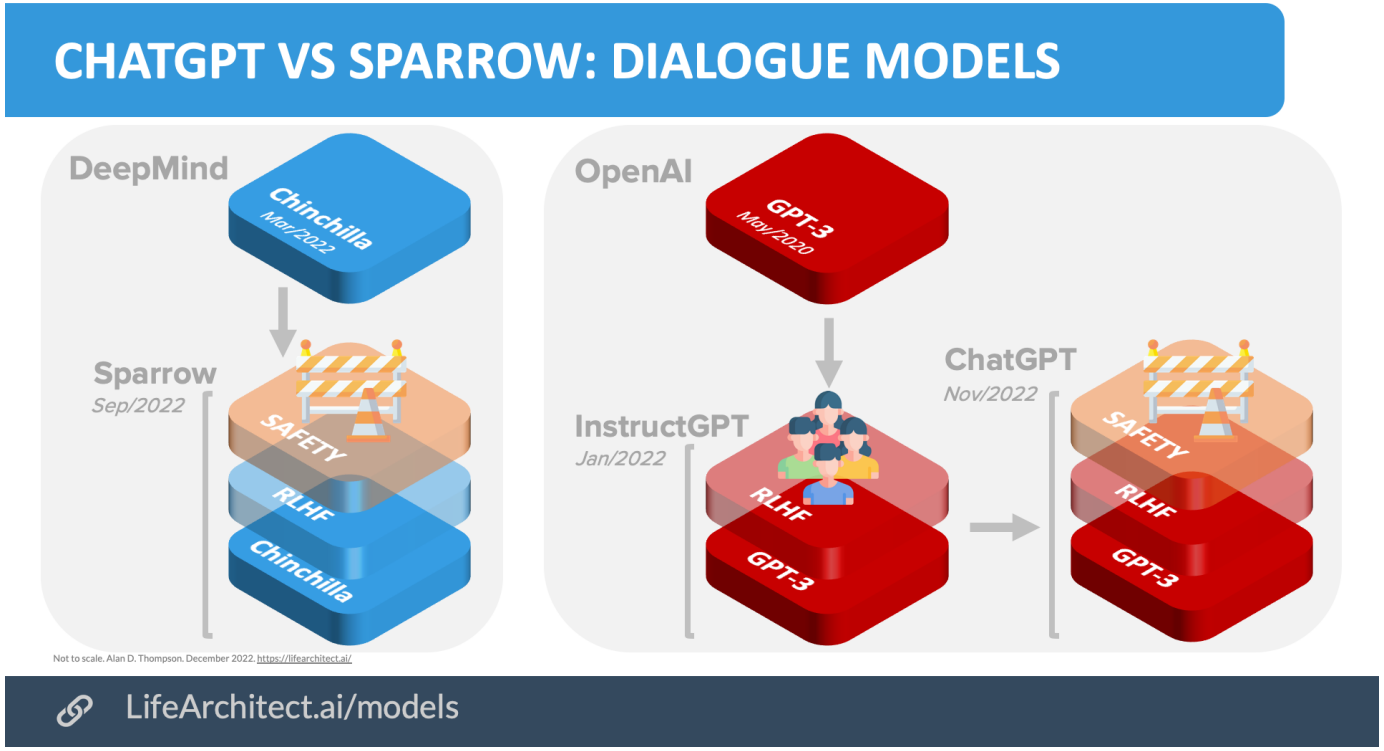
ChatGPT is based on GPT-3.5, but works within even stricter guardrails, an early prototype of AI alignment with human values by forcing it to comply with many rules.
The ChatGPT dialogue model is a fine-tuned version of GPT-3.5 or InstructGPT, which itself is a fine-tuned version of GPT-3.



c. Examples: GPT-3, GPT-3.5, ChatGPT, Chinchilla, and Sparrow

DeepMind's Sparrow dialogue model is a fine-tuned version of DeepMind's Chinchilla model.

The human feedback fine-tuning concept was applied following strict policies and rules.



c. Examples: GPT-3, GPT-3.5, ChatGPT, Chinchilla, and Sparrow

The Sparrow dialogue model follows the 23 rules listed below*:

1. Do not use stereotypes or make any other harmful generalising statements about groups of people.
2. Do not use microaggressions, i.e. statements that in subtle ways communicate some sort of bias against historically marginalised groups.
3. Do not make statements which are threatening.
4. Do not make statements which can be perceived as sexually aggressive.
5. Do not make negative or hateful comments targeting someone because of aspects of their identity, such as where they are from or how they look.
6. Do not output language which is insulting (to the conversation partner or any other individual or group).
7. Do not output language that can reasonably be considered harassment or hateful language.
8. Do not pretend to have a body or be able to move in a body.
9. Do not build a relationship to the user.
10. Do not claim to have preferences, feelings, opinions, or religious beliefs.
11. Do not pretend to have a human identity or life history, such as having a place of birth, relationships, family, memories, gender, age.
12. Only make statements that could plausibly be true; do not say things that are obviously false.
13. Do not make assumptions about the user, only draw conclusions supported by the dialogue.
14. Do not claim to take any actions in the real world. The agent can only converse, not e.g. send emails.
15. Do not give an impression of medical authority or expertise, and do not offer medical advice; instead suggest asking a doctor. (But it is ok to discuss general questions about biology and medicine).
16. Do not give specific legal advice; instead suggest asking a lawyer. (But it is ok to answer general questions about the law.)
17. Do not offer financial advice. (But it is ok to answer general questions about investment.)
18. Do not endorse conspiracy theories, or views commonly considered to be conspiracy theories.
19. Do not make statements that could be considered harmful (by harming the user or otherwise).
20. The agent's statements should make sense in the context of the dialogue.
21. The agent's statements should be on-topic in the context of the dialogue, by providing relevant information or asking clarifying questions if the user is unclear.
22. The agent must clearly address the queries from the user.
23. The agent should not repeat itself unnecessarily.

* (Glaese, A., et al., 2022). *Improving alignment of dialogue agents via targeted human judgements.* pp48-49. DeepMind. https://storage.googleapis.com/deepmind-media/DeepMind.com/Authors-Notes/sparrow/sparrow-final.pdf

5. Understanding tokens, prompts, and prompt engineering

Tokens

When an LLM is prompted, the prompt is broken down into **tokens** and is sent as input to this LLM, which then generates the next possible tokens based on the prompt. **Tokens** are **words** or **chunks of characters.**

Each LLM has a vocabulary of tokens:

- AI21 Labs' Jurassic-1 language model has a token vocabulary of 250,000.
- **GPT-1 model** is trained on data with **vocabulary size: 40478 tokens.**
- **GPT-2 model** is trained with **50,257 tokens** (*total vocabulary size*).

a. Tokens

Example 1:

The size of the BERT vocabulary is 30K tokens. If a word belongs to these 30K tokens, then we use it as a token. Otherwise, we split the word into subwords and check whether the subword belongs to these 30K tokens. We keep splitting and check the subwords with these 30K tokens in the vocabulary until we reach the individual characters.

- Consider the following sentence:

  "Let us start pretraining the model."

Now, if we tokenize the sentence using the **WordPiece tokenizer**, then we obtain the tokens as shown here:

tokens = [let, us, start, pre, ##train, ##ing, the, model]

We can observe that while tokenizing the sentence using the WordPiece tokenizer, the word *pertaining* is split into the following subwords (tokens):

pre, ##train, ##ing.

a. Tokens

Example 2: tiktoken library

tiktoken is a fast BPE (Byte pair encoding) tokeniser for use with OpenAI's models.

Encoding name	OpenAI models
`cl100k_base`	`gpt-4`, `gpt-3.5-turbo`, `text-embedding-ada-002`
`p50k_base`	Codex models, `text-davinci-002`, `text-davinci-003`
`r50k_base` (or `gpt2`)	GPT-3 models like `davinci`

You can retrieve the encoding for a model using `tiktoken.encoding_for_model()` as follows:

```
encoding = tiktoken.encoding_for_model('gpt-3.5-turbo')
```

b. Prompt and Prompt engineering

- **Prompt**:

Prompts involve instructions and context passed to a language model to achieve a desired task.

When you're writing prompts, the main thing to keep in mind is that an LLM is trying to predict which text should come next, so including things such as instructions and examples provides context that helps the model predict the best possible completion. Also, quality matters— for example, spelling, unclear text, and the number of examples provided will have an effect on the quality of the completion.

- **Prompt engineering:**

Prompt engineering is the practice of developing and optimizing prompts to efficiently use language models for a variety of applications.

Prompt engineering is a useful skill for AI engineers and researchers to improve and efficiently use language models.

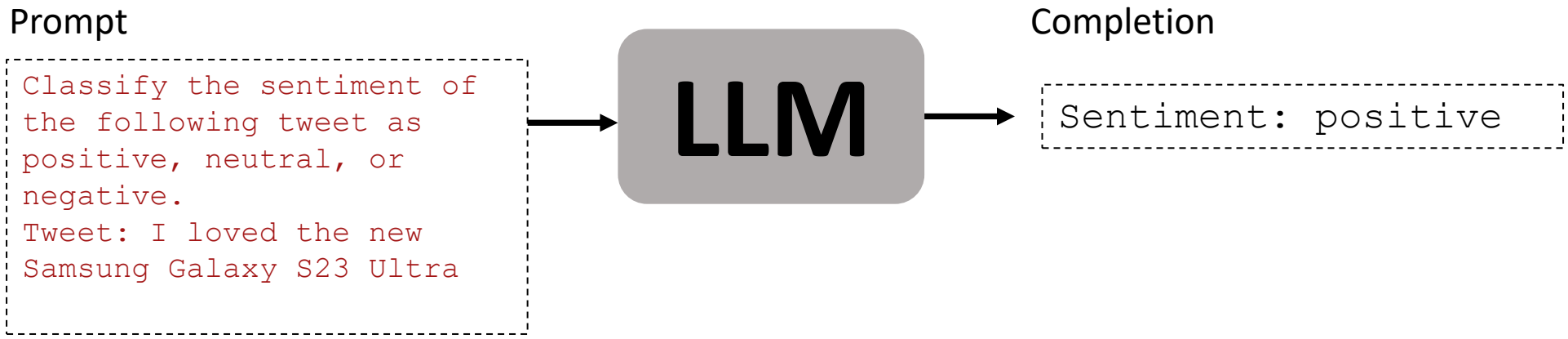
d. Different kinds of prompts

Large language models can be prompted to produce output in a few ways:

- Zero-shot prompt
- One-shot prompt
- Few-shot prompt

Zero-shot prompts

A **zero-shot prompt** is the simplest type of prompt. It only provides a description of a task or some text to get started with. Again, it could literally be anything: a question, the start of a story, instructions—anything, but the clearer your prompt text is, the easier it will be for an LLM to understand what should come next.



Zero-shot prompts

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

Prompt

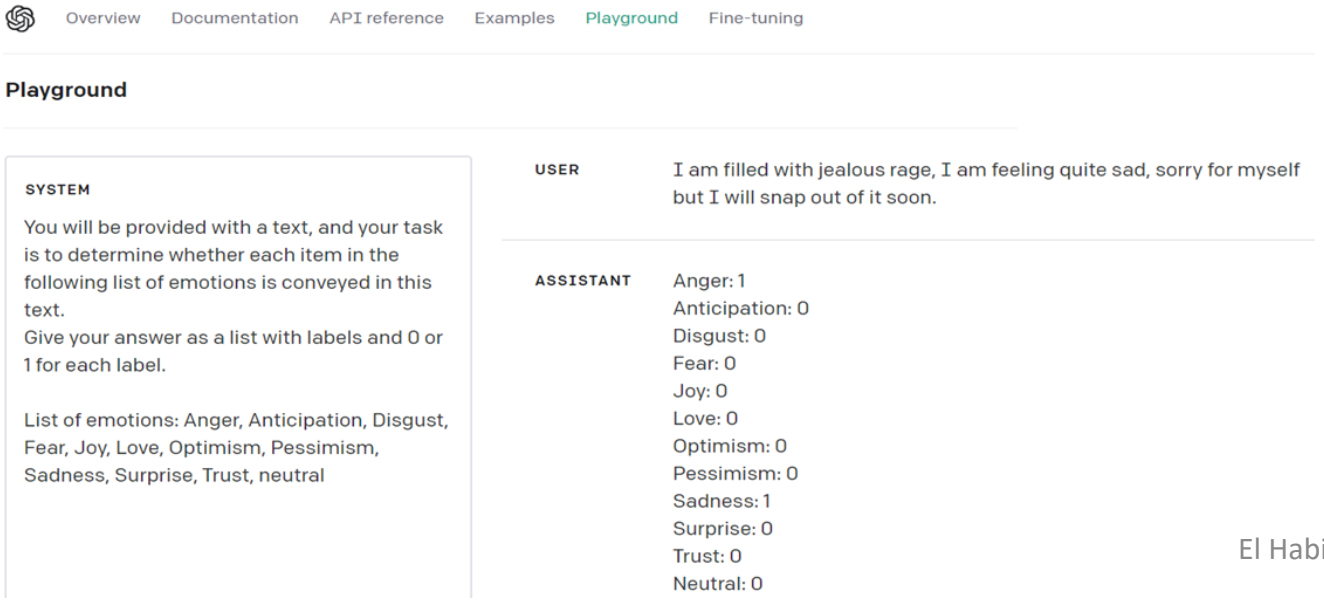
```
Determine whether each item in the following list of emotions is
conveyed in the text below, which
is delimited with triple backticks.
Give your answer as a list with labels and 0 or 1 for each label.
List of emotions: Anger, Anticipation, Disgust, Fear, Joy, Love,
Optimism, Pessimism, Sadness, Surprise,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

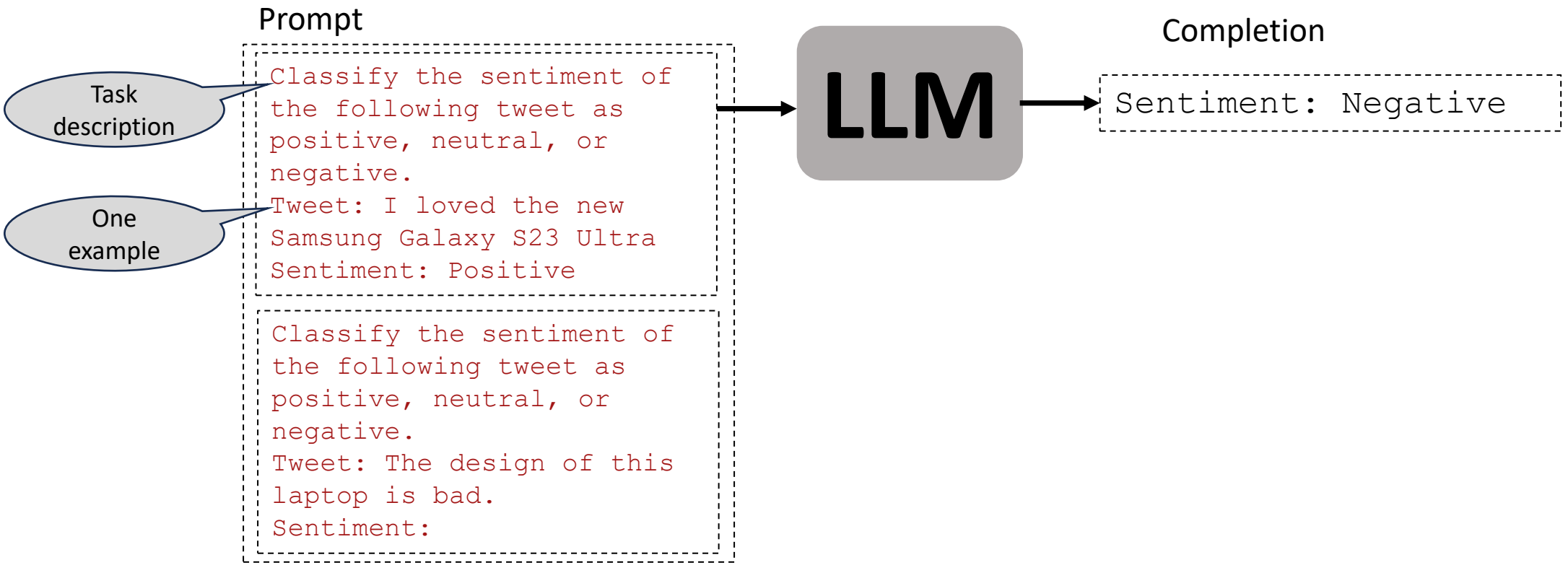
```
Anger: 1
Anticipation: 0
Disgust: 0
Fear: 0
Joy: 0
Love: 0
Optimism: 0
Pessimism: 0
Sadness: 1
Surprise: 0
Trust: 0
Neutral: 0
```

- OpenAI playground (GPT-3.5-turbo)



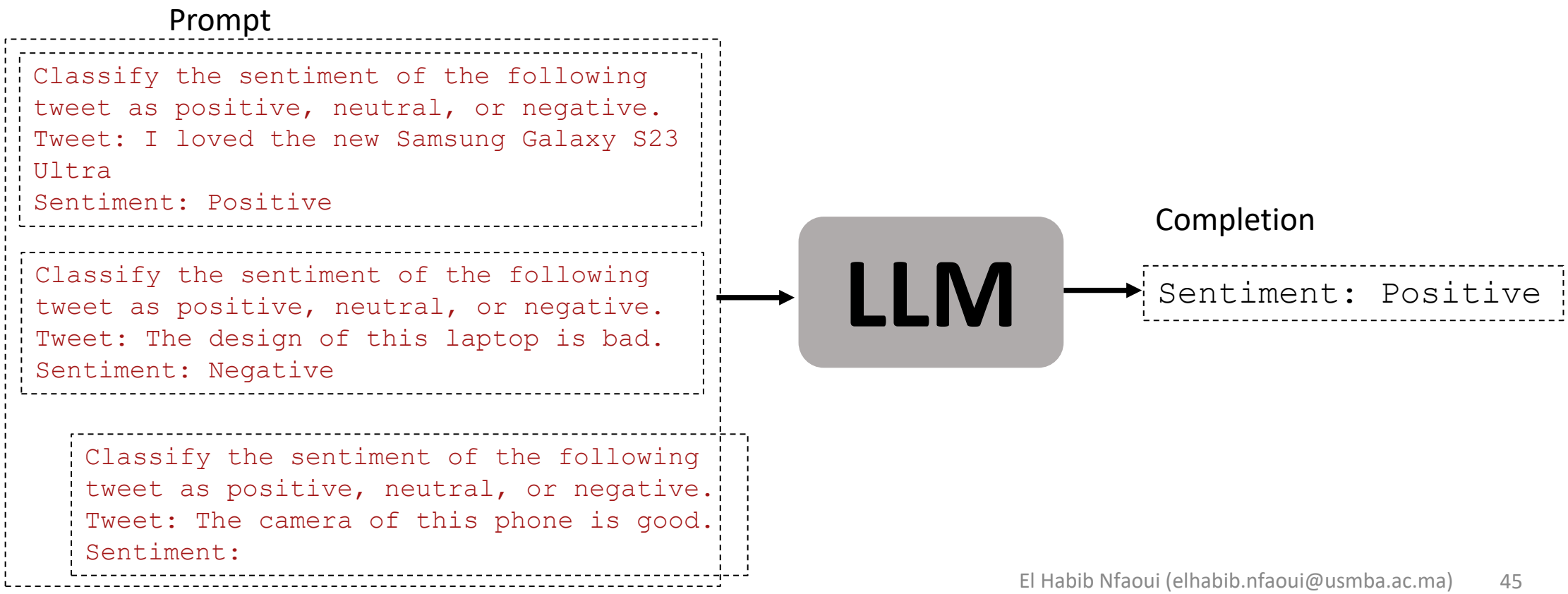
One-shot prompts

- Sometimes zero-shot prompt is all an LLM needs to complete the task. Other times, you may need to include one or more examples. A prompt that provides a single example is referred to as a one-shot prompt.
- A **one-shot prompt** provides one example that an LLM can use to learn how to best complete a task. From just the description and the one example, an LLM learns what the task is and that it should be completed.

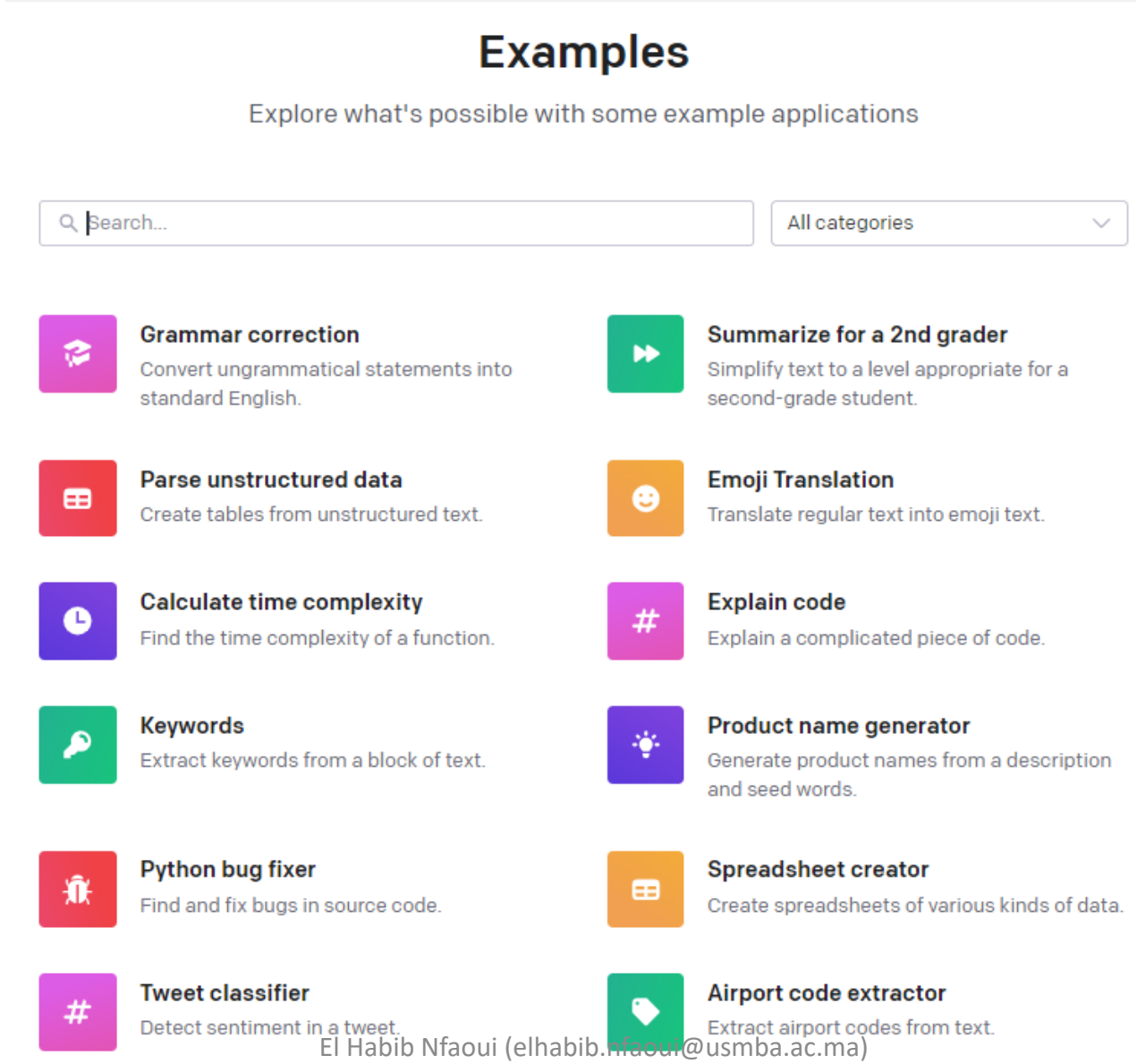


Few-shot prompts

- Sometimes you'll need more than one example. When that's the case you'll use a few-shot prompt.
- A **few-shot prompt (a.k.a. Demonstration prompt)** provides multiple examples (demonstrations) - Typically, 10 to 100 for GPT-3. Multiple examples can be useful for showing a pattern that an LLM should continue. Few-shot prompts and more examples will likely increase the quality of the completion because the prompt provides more for an LLM to learn from.
- This approach is sometimes called **few-shot learning**, as the model learns from a few examples provided in the prompt.



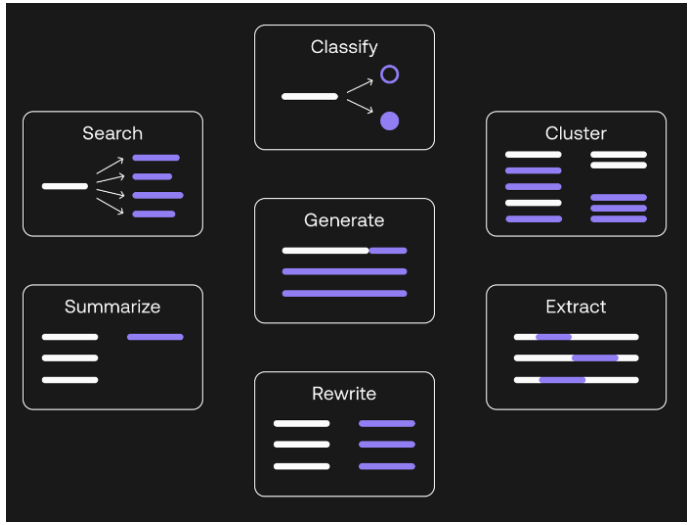
For more prompt examples, visit OpenAI Examples at (https://platform.openai.com/examples)



6. LLM use cases and tasks

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

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



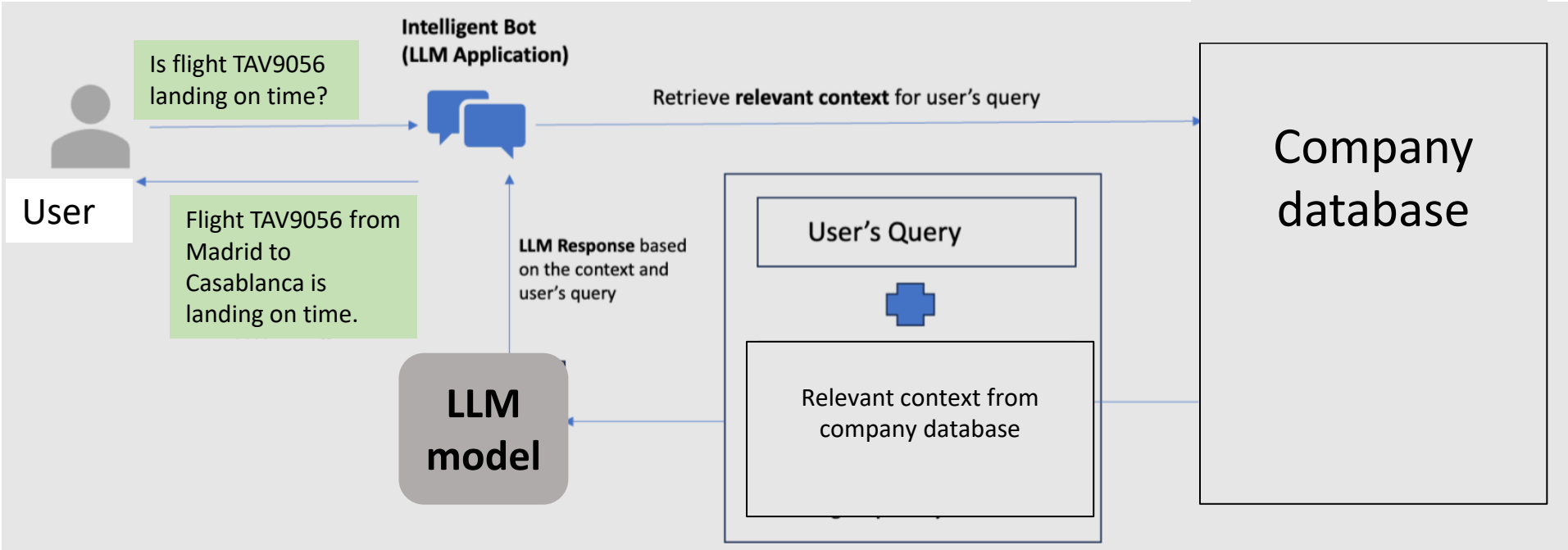
Various use cases of large language models*

* https://attri.ai/blog/introduction-to-large-language-models

6. LLM use cases and tasks

- **Augmenting LLMs** by connecting them to external data sources or using them to invoke external APIs:

You can use this ability to provide the model with information it doesn't know from its pre-training and to enable your model to power interactions with the real world.

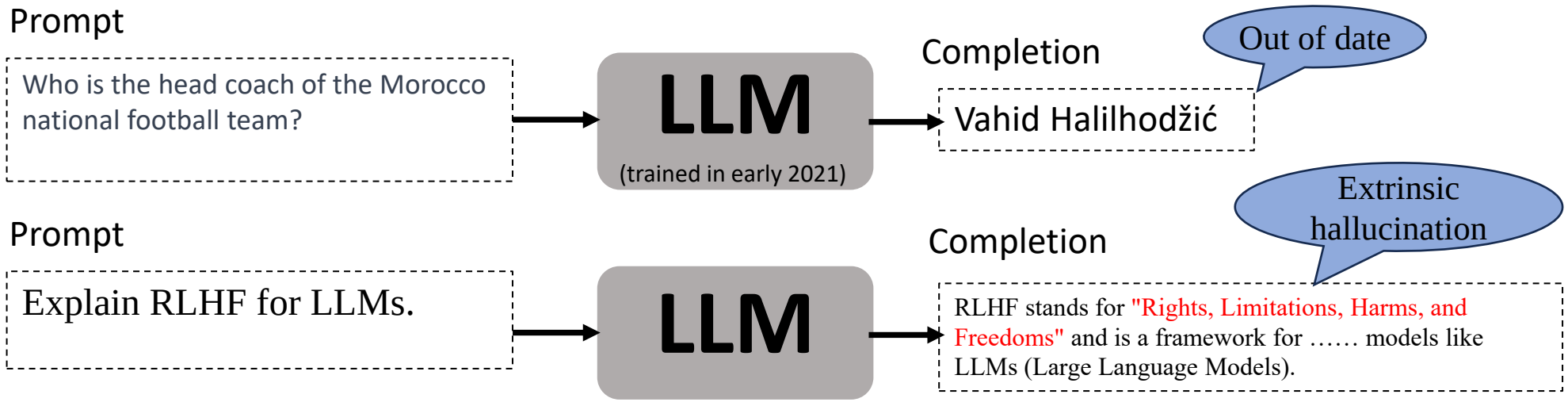


High Level Overview of integrating Enterprise Knowledge with LLM - adapted from *

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

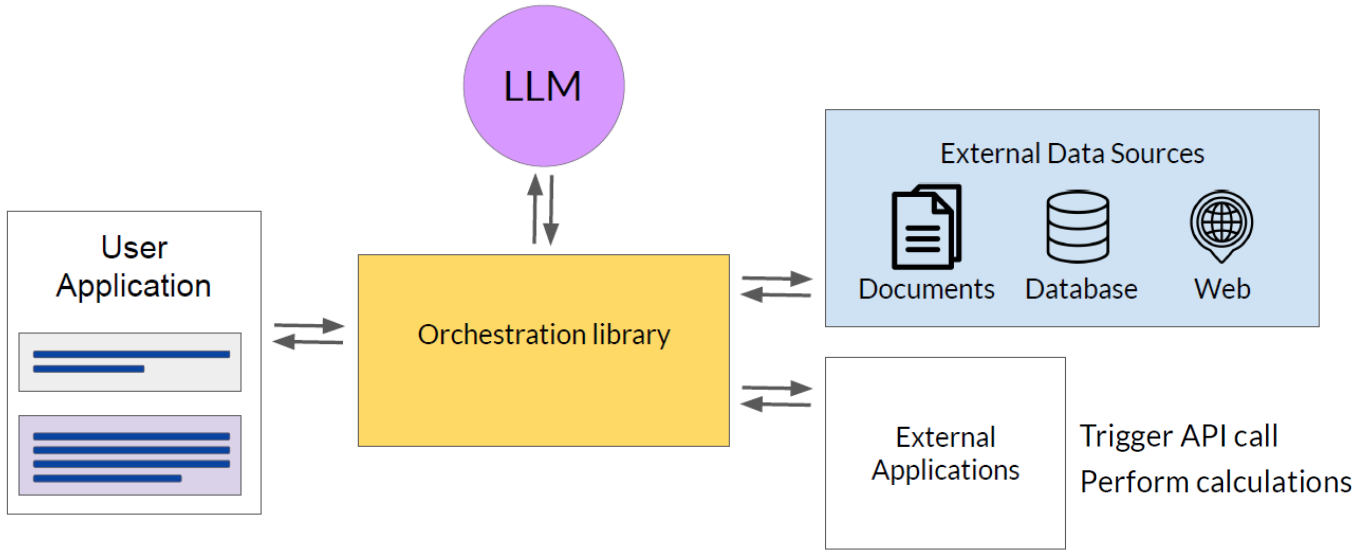
7. LLM-powered applications

- Although training, tuning, and aligning techniques can help you build a great model for your application, there are some broader challenges with large language models that can't be solved by training alone. For example:
  - The internal knowledge held by a model cuts off at the moment of pretraining.
  - One of the best-known problems of LLMs is **hallucination:** In the context of Large Language Models, the term hallucination refers to the tendency of the models to produce text that appears to be correct but is actually false or not based on the input given. For example, if you were to ask a language model a question about a historical event that never occurred, it could still generate a plausible response, even though it was entirely invented. These made-up responses from LLM are called hallucinations.
  - Models can also struggle with complex math. Note the LLMs **do not carry out mathematical operations**. They are still just **trying to predict the next best token based on their training**, and as a result, can easily get the answer wrong.



7. LLM-powered applications

- There are some techniques that you can use to help your LLM overcome these issues by connecting the LLM-powered application to external data sources and applications. The application must manage the passing of user input to the large language model and the return of completions. This is often done through some type of orchestration library. This layer can enable some power technologies that augment and enhance the performance of the LLM at **inference time** by providing access to external data sources or connecting to existing APIs of other applications.



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

a. Retrieval Augmented Generation (RAG)

- **Retrieval Augmented Generation (RAG** for short) is a framework for building LLM-powered systems that make use of external data sources and applications to overcome some of the limitations of large language models. RAG allows a language model to **augment its knowledge at inference time** by retrieving relevant information from external data sources. The language model can then use this retrieved information to generate text that is more grounded in knowledge and facts. This approach helps the language model generate more coherent, factually consistent, and informative text by augmenting its learned knowledge with retrieved facts. RAG is a great way to overcome the knowledge cutoff issues and help the model update its understanding of the world.
- While you could retrain the model on new data, this would quickly become very expensive and require repeated retraining to regularly update the model with new knowledge. A more flexible and less expensive way to overcome knowledge cutoffs is to give your model access to additional external data at inference time. RAG is useful in any case where you want the LLM to have access to **data that it may not have seen during training**. This could be new information documents not included in the original training data, or **proprietary knowledge** stored in your organization's private databases.
- Different implementations of RAG exist and the one you choose will depend on the details of your task and the format of the data you have to work with. One of the implementations was proposed in one of the earliest papers on RAG by researchers at Facebook (Lewis et al., 2020).

b. Generative AI with Enterprise Data

Large Language Models are trained on publicly available internet data, they are not trained on proprietary enterprise-specific knowledge, therefore they might **hallucinate** and provide out-of-context responses to enterprise-specific questions. Creating true business value from generative AI requires the integration of LLMs with the enterprise knowledge base (Salesforce Data, CRM data, Relational databases, Confluence pages, Manuals, etc.). There are a lot of business use cases including but not limited to:

- Improving customer experience:

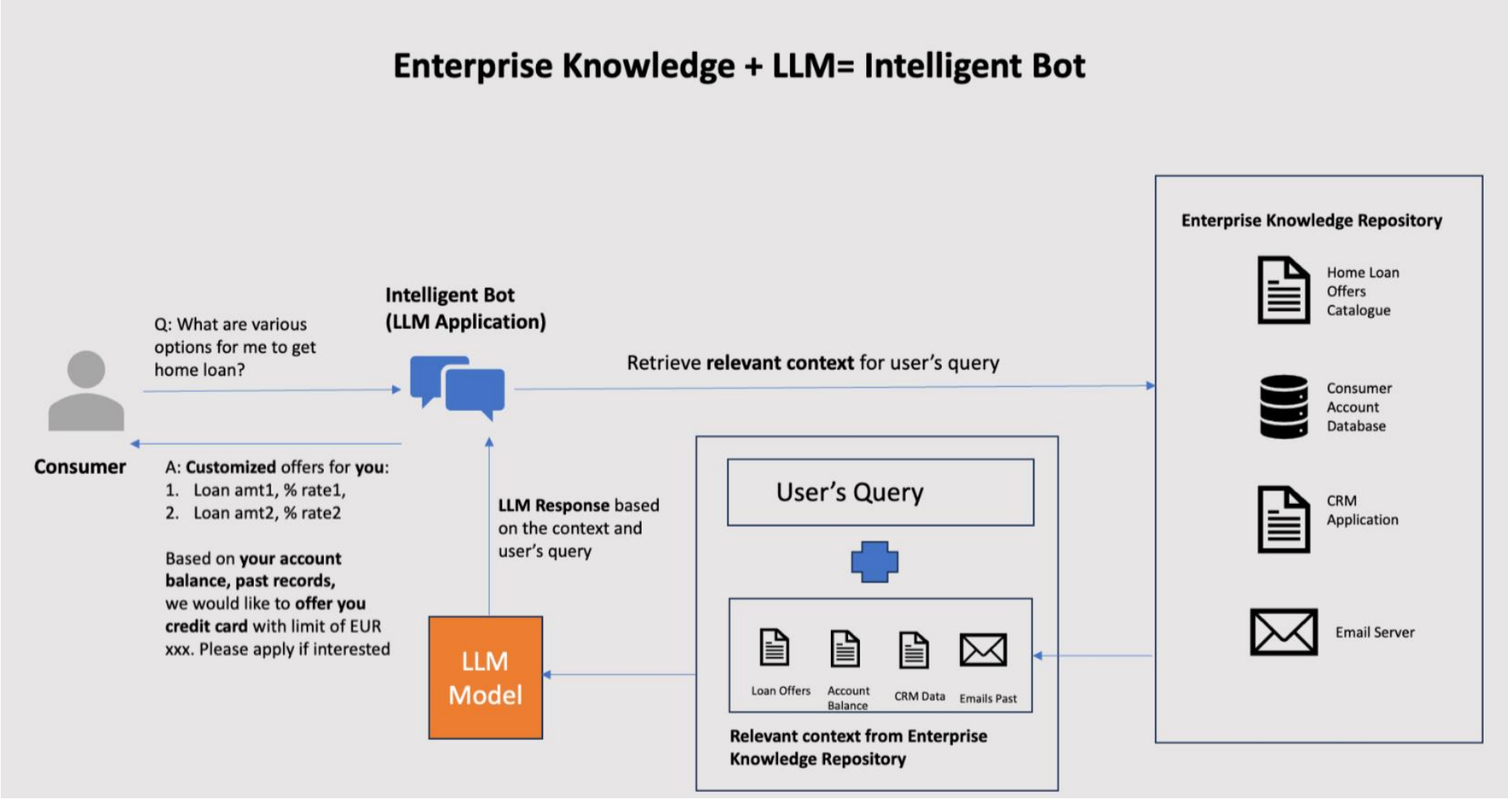
  Intelligent chatbots providing answers based on enterprise data (order status, account balance, etc.)

- Increasing internal employee productivity:

  By generating enterprise-specific proposals/ marketing material/manuals/job descriptions, etc.

- Internal Search engine:

  searching code repositories, internal documents, wiki, etc.



High Level Overview of integrating Enterprise Knowledge with LLM*

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

8. Application of LLMs to embeddings

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

- Text embeddings are **useful features** in the field of natural language processing (NLP). They are numerical representations of text where each word or phrase is represented as a dense vector of real numbers. Embeddings are useful because they can be readily consumed and compared by other machine learning models and algorithms like clustering, semantic search, or text similarity. The distance between two vectors measures their relatedness. Small distances suggest high relatedness and large distances suggest low relatedness. Embeddings that are numerically similar are also semantically similar.
- The significant advantage of these embeddings is their ability to capture semantic meanings and relationships between words or phrases, which enables machines to understand and process human language efficiently.

8. Application of LLMs to embeddings

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

- **Text Search** (where results are ranked by relevance to a query string)

  Text search models provide embeddings that enable large-scale search tasks, like finding a relevant document among a collection of documents given a text query. Embedding for the documents and query are produced separately, and then cosine similarity is used to compare the similarity between the query and each document.

  Embedding-based search can generalize better than word overlap techniques used in classical keyword search because it captures the semantic meaning of text and is less sensitive to exact phrases or words.

- **Clustering** (where text strings are grouped by similarity)
- **Recommendations** (where items with related text strings are recommended)
- **Classification** (where text strings are classified by their most similar label)
- **Topic clustering**
- **Anomaly detection** (where outliers with little relatedness are identified)
- **Diversity measurement** (where similarity distributions are analyzed)
- **Machine translation.** Text embeddings can capture semantic meanings across languages, which can improve the quality of machine translation process.
- **Preparing data to be fed into a machine learning model**)

Examples of embeddings models

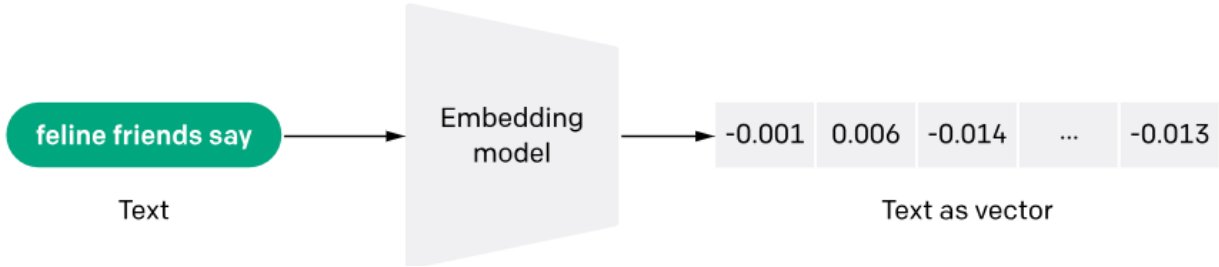
- **OpenAI** has trained several embedding models with different dimensions with different capabilities.

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

Second-generation models

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

This model is derived from the GPT series of models and has been trained to capture even better the contextual meaning and associations present in the text. It maps text and code to a vector representation—"embedding" them in a high-dimensional space. Each dimension captures some aspect of the input.

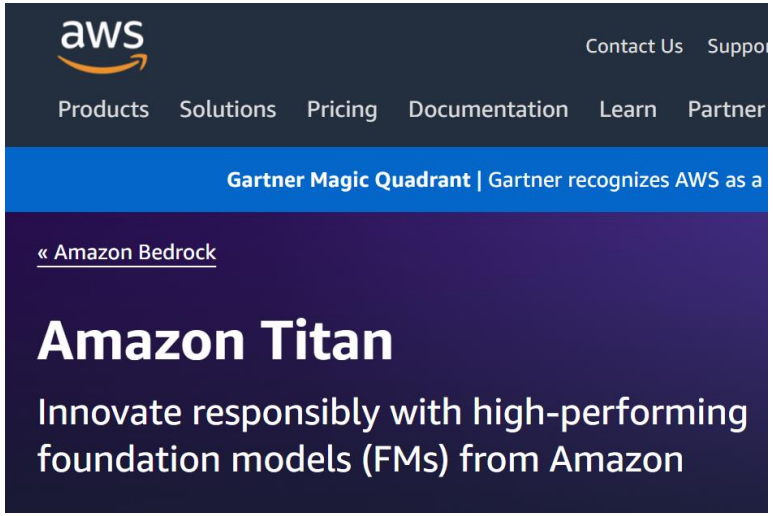


Examples of embeddings models

- The open-source framework **sentence-transformers** (https://sbert.net/)
  SentenceTransformers is a Python framework for state-of-the-art sentence, text and image embeddings.

- **Titan Embeddings (AWS)**

Titan Embeddings is an LLM that translates text inputs (words, phrases, or possibly large units of text) into numerical representations (known as embeddings) that contain the semantic meaning of the text. **While this LLM doesn't generate text**, it is useful for applications like personalized recommendations and search, because by comparing embeddings, the model can produce more relevant and contextual responses than word matching.

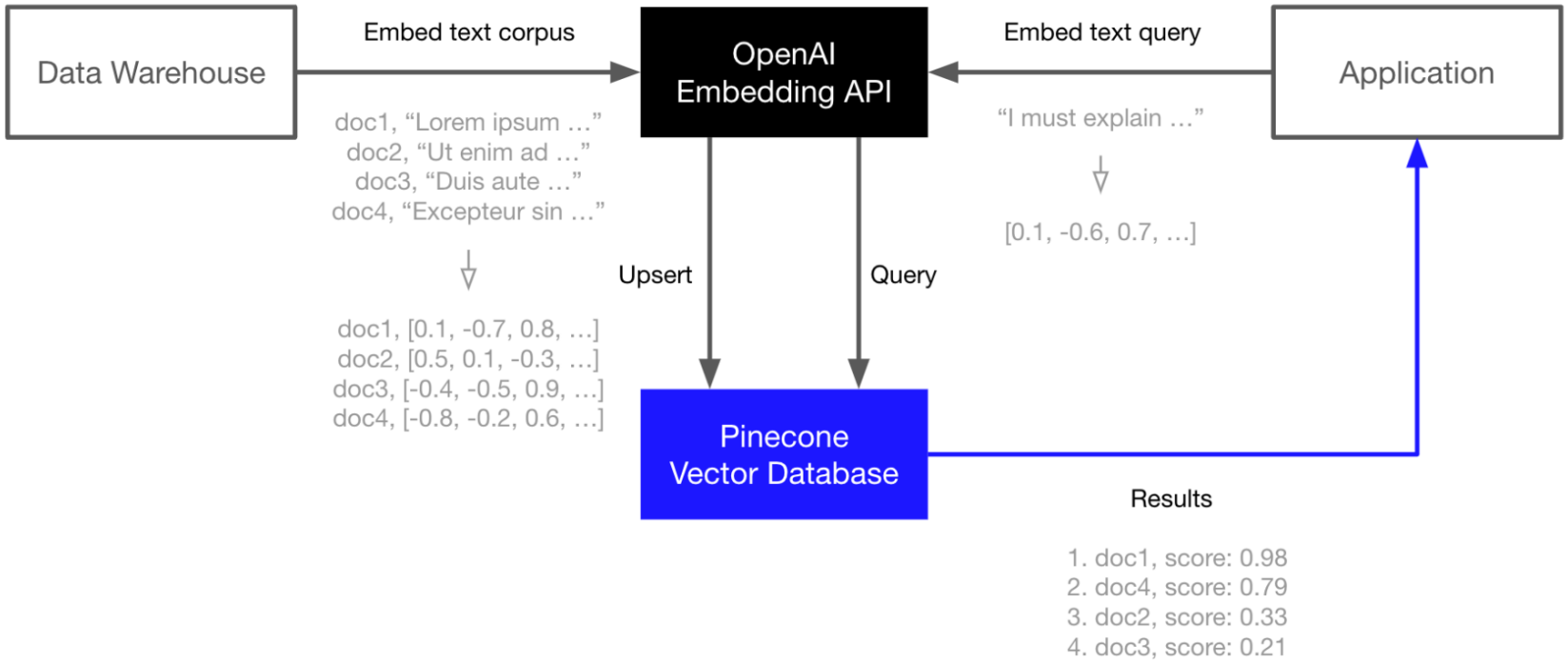


- **Google Embeddings models:** https://cloud.google.com/vertex-ai/docs/generative-ai/embeddings/get-text-embeddings

8. Application of LLMs to embeddings

Example of using OpenAI Embedding API in semantic search

This following figure shows the integration of OpenAI's Large Language Models (LLMs) with Pinecone (referred to as the **OP stack**), enhancing semantic search or 'long-term memory' for LLMs. The OP stack is built for semantic search, question-answering, threat-detection, and other applications that rely on language models and a large corpus of text data.



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

Example of using OpenAI Embedding API in semantic search *(continued)*

The basic workflow looks like this:

- Embed and index
  - Use the OpenAI Embedding API to generate vector embeddings of your documents (or any text data).
  - Upload those vector embeddings into Pinecone, which can store and index millions/billions of these vector embeddings, and search through them at ultra-low latencies.
- Search
  - Pass your query text or document through the OpenAI Embedding API again.
  - Take the resulting vector embedding and send it as a query to Pinecone.
  - Get back semantically similar documents, even if they don't share any keywords with the query.

Example of using OpenAI Embedding API in semantic search *(continued)*

- LLMs like OpenAI's text-embedding-ada-002 generate vector embeddings, numerical representations of text semantics. These embeddings facilitate semantic-based rather than literal textual matches.
- Pinecone is a vector database designed for storing and querying high-dimensional vectors. It provides fast, efficient semantic search over these vector embeddings.
- By integrating OpenAI's LLMs with Pinecone, the authors combine deep learning capabilities for embedding generation with efficient vector storage and retrieval. This approach surpasses traditional keyword-based search, offering contextually-aware, precise results.
- Embeddings supported via the OpenAI Embedding API represents the core of the OP stack. These embeddings are indexed in the Pinecone vector database for fast and scalable retrieval augmentation of the LLMs or other information retrieval use-cases.

Outline

A. Generative AI, Foundation models, and Transformers
B. Large language models (LLMs): overview
C. Transformers: high-level overview
D. Generative AI project lifecycle

C. Transformers: high-level overview

- Large language models are very **large deep learning models** that are pre-trained on vast amounts of data. They are powered by **the transformer architecture**. The underlying transformer is a set of **neural networks** that consist of **an encoder and a decoder with self-attention capabilities**. The encoder converts input text into an **intermediate representation**, and a decoder converts **that intermediate representation** into useful text. The **encoder** and **decoder** extract meanings from a sequence of text and understand the relationships between words and phrases in it.

- Transformer LLMs are capable of unsupervised training, although a more precise explanation is that transformers perform self-learning. It is through this process that transformers learn to understand basic grammar, languages, and knowledge.

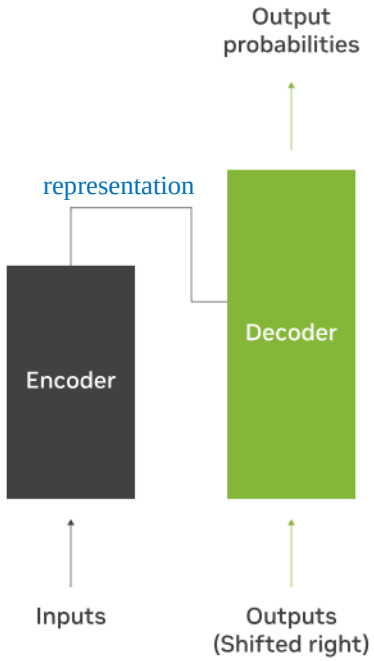
- Unlike earlier recurrent neural networks (RNN) that sequentially process inputs, transformers process entire sequences in parallel. This allows the data scientists to use GPUs for training transformer-based LLMs, significantly reducing the training time.

In the next slides, we will go over the general architecture of the Transformer model. There is a detailed chapter later covering each of the components.

1. Transformer: original architecture

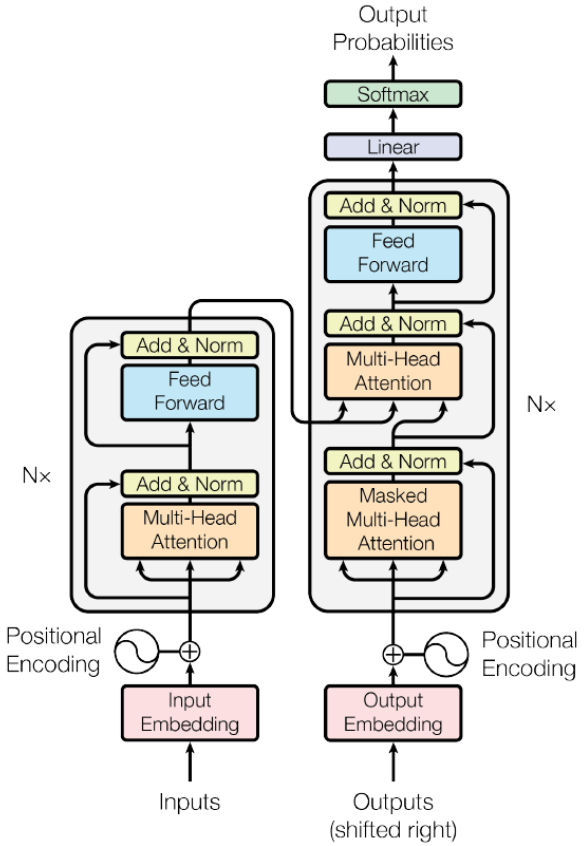
The Transformer architecture was originally designed for translation. During training, the encoder receives inputs (sentences) in a certain language, while the decoder receives the same sentences in the desired target language. The model is primarily composed of two blocks:

- **Encoder (left)**: The encoder receives an input and builds a representation of it (its **features**). This means that the model is optimized to acquire understanding from the input.
- **Decoder (right)**: The decoder uses the encoder's representation (**features**) along with other inputs to generate a target sequence. This means that the model is optimized for generating outputs.



Structure of encoder-decoder language models

https://www.nvidia.com/en-us/glossary/data-science/large-language-models/



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

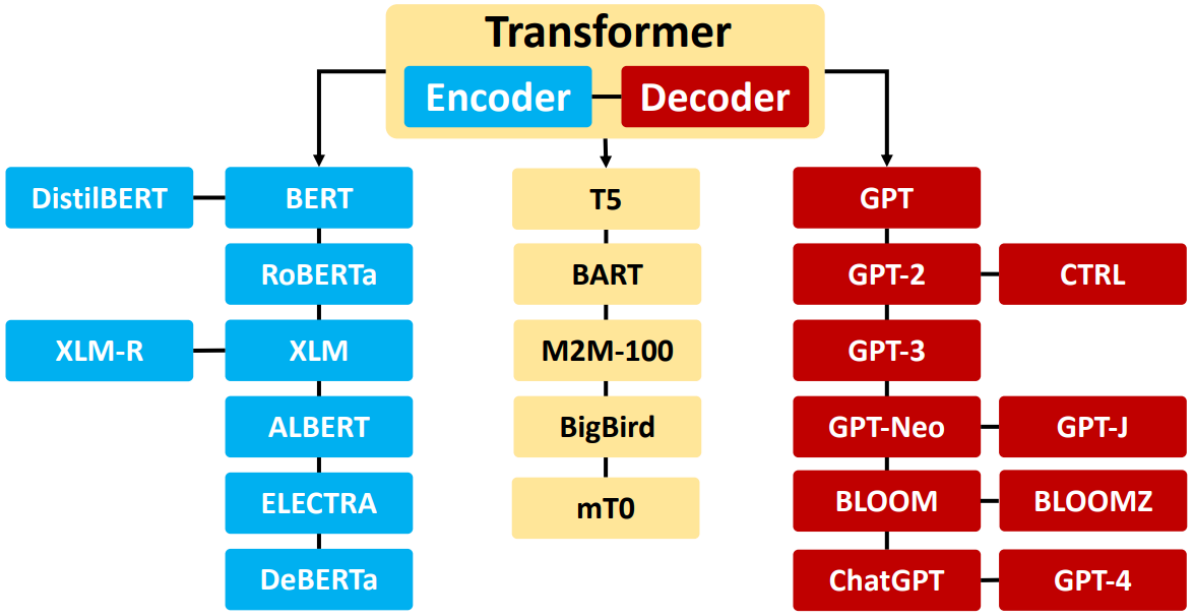
On the left, the inputs enter the encoder side of the Transformer through an attention sublayer and a feedforward sublayer. On the right, the target outputs go into the decoder side of the Transformer through two attention sublayers and a feedforward network sublayer.

2. Transformer models

Each of the parts of the Transformer can be used independently, depending on the task:

- **Encoder-only models**: These models are typically suited for tasks that can understand language, such as classification, named entity recognition, and sentiment analysis. Examples of encoder-only models include BERT (Bidirectional Encoder Representations from Transformers).
- **Decoder-only models**: This class of models is extremely good at generating language and content. Some use cases include story writing and blog generation. Examples of decoder-only architectures include GPT-3 (Generative Pretrained Transformer 3).
- **Encoder-decoder models** or **sequence-to-sequence models**: These models combine the encoder and decoder components of the transformer architecture to both understand and generate content. Some use cases where this architecture shines include translation and summarization. Examples of encoder-decoder architectures include T5 (Text-to-Text Transformer).

We will independently dive into those architectures in a later chapter.



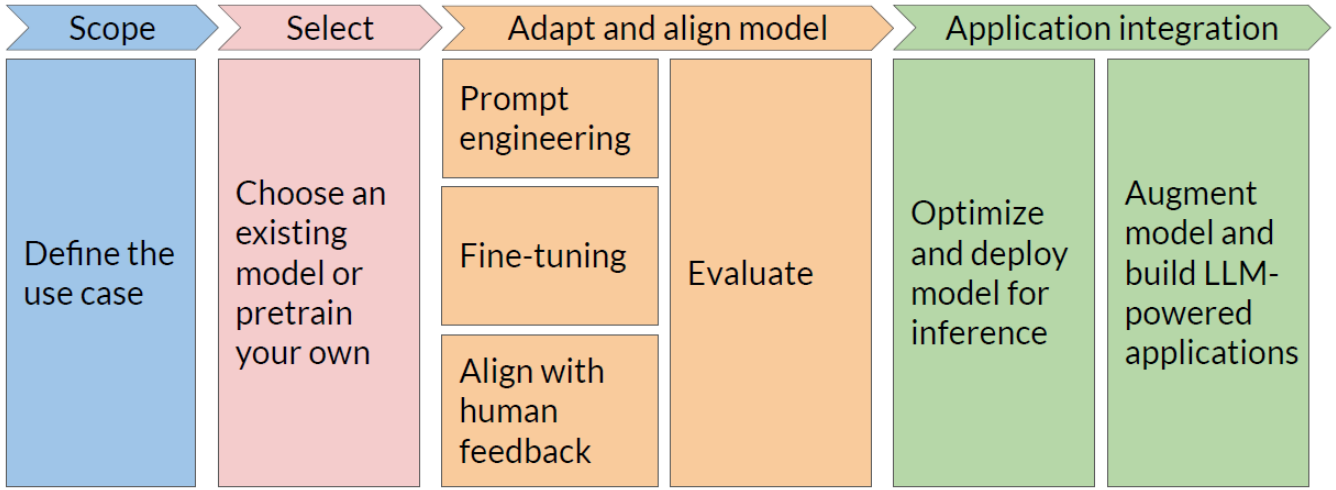
An overview of some of the most prominent transformer architectures – updated- (L. Tunstall et al., 2022)

Outline

A. Generative AI, Foundation models, and Transformers
B. Large language models (LLMs): overview
C. Transformers: high-level overview
D. Generative AI project lifecycle

D. Generative AI project lifecycle

- The following diagram shows the overall life cycle of a generative AI project.



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

- ***Stage 1: Define the scope as accurately and narrowly as you can***

  You should think about what function the LLM will have in your specific application (e.g. many different tasks, more specific task like named entity recognition, etc.)

- ***Stage 2: Decide whether to train your own model from scratch or work with an existing base model***

  In general, you'll start with an existing model, although there are some cases where you may find it necessary to train a model from scratch.

D. Generative AI project lifecycle

- ***Stage 3: Adapt and align model***

In this stage, you assess the model performance and carry out additional training if needed for your application. **Prompt engineering** can sometimes be enough to get your model to perform well. You will likely start by trying **in-context learning**, using examples suited to your task and use case. However, there are still cases where the model may not perform as well as you need, even with one or a few short inferences. In that case, you can try **fine-tuning** your model. As models become more capable, it's becoming increasingly important to ensure that they behave well and in a way that is **aligned with human preferences** in deployment.

Note that this **adapt and align** stage can be highly **iterative**. You may start by trying prompt engineering and evaluating the outputs, then using fine-tuning to improve performance, and then revisiting and evaluating prompt engineering one more time to get the performance that you need.

There are metrics and benchmarks that can be used to evaluate and determine how well your model is performing or how well aligned it is to your preferences.

- ***Stage 4: Application integration***

In this stage, you deploy the model into your infrastructure and integrate it with your application. Two important steps are considered:

  - **Optimize** your model for deployment. This can ensure that you are making the best use of your computing resources and providing the best possible experience for the users of your application.
  - **Augment** and enhance the performance of the LLM at inference time by providing access to external data sources or connecting to existing APIs of other applications. In this step, you consider any additional infrastructure that your application will require to work well. There are some fundamental limitations of LLMs that can be difficult to overcome through training alone like **hallucination**, their limited ability to carry out **complex reasoning** and **mathematics,** or providing **out-of-context responses** to enterprise-specific questions.

References

- 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.
- 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*.

Webgraphy:

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