csci e-89 Deep Learning
Lecture 14
Large Language Models (LLMs)

Zoran Djordjević & Rahul Joglekar, Fall 2024

Objectives

- We will describe Large Language Models (LLMs), their history and working
- We will drill down in to GPT family of models developed at OpenAI
- We will demonstrate the use of OpenAI's API for accessing GPT like models for practical text searches and generation.

References

The second part of the lecture on Large Language Models relies on:

1. A Survey of Large Language Models, by Wayne Xin Zhao et al. https://arxiv.org/abs/2303.18223
2. GPT-4 Technical Report, by OpenAI, https://arxiv.org/abs/2303.08774

- The last part of the lecture on practical uses of OpenAI API comes from

https://platform.openai.com/docs/api-reference

- Most excellent code examples in Python for GPT family of models could be found at:

https://github.com/openai/openai-cookbook

- Reference for In-Context Learning: "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

Additional References

[1] “Introducing OpenAI.” OpenAI, 12 Dec. 2015, https://openai.com/blog/introducing-openai/.

[2] Vaswani, Ashish, et al. “Attention is all you need.” Advances in neural information processing systems 30(2017).

[3] Radford, Alec, et al. “Improving language understanding by generative pre-training.” (2018).

[4] Radford, Alec, et al. “Language models are unsupervised multitask learners.” OpenAI blog 1.8 (2019): 9.

[5] Brown, Tom, et al. “Language models are few-shot learners.” Advances in neural information processingsystems 33 (2020): 1877–1901.

[6] Ouyang, Long, et al. “Training language models to follow instructions with human feedback.” arXivpreprint arXiv:2203.02155 (2022).

[7] “ChatGPT: Optimizing Language Models for Dialogue.” OpenAI, 30 Nov. 2022,https://openai.com/blog/chatgpt/.

[8] Murali, Aishwarya. “A Guide to Perform 5 Important Steps of NLP Using Python.” Analytics Vidhya, 17Aug. 2021, https://www.analyticsvidhya.com/blog/2021/08/a-guide-to-perform-5-important-steps-of-nlp-using-python/.

[9] Cristina, Stefania. “The Transformer Attention Mechanism.” Machine Learning Mastery, 15 Sept. 2022,https://machinelearningmastery.com/the-transformer-attention-mechanism/.

[10] Doshi, Ketan. “Transformers Explained Visually (Part 1): Overview of Functionality.” Medium, 3 June 2021,https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452.

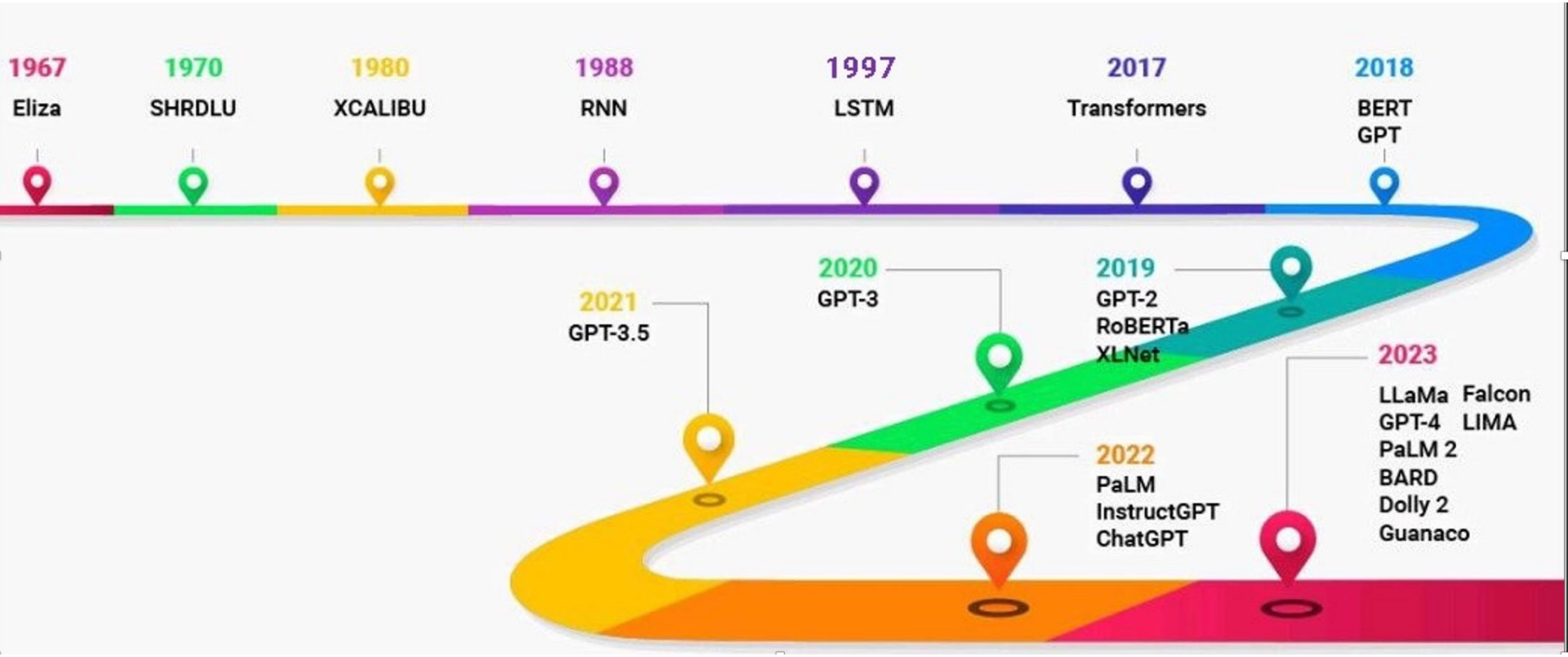
[11] Kosar, Vaclav. Feed-Forward, Self-Attention & Key-Value. 2 Jan. 2021, https://vaclavkosar.com/ml/Feed-Forward-Self-Attendion-Key-Value-Memory.

[12] “Aligning Language Models to Follow Instructions.” OpenAI, 27 Jan. 2022,https://openai.com/blog/instruction-following/.

[13] *Mikolov, Tomas; et al. (2013). "Efficient Estimation of Word Representations in Vector Space". arXiv:1301.3781 [cs.CL].*

Large Language Models

Evolution of LLMs



Brief History

Language modeling (LM) aims to model the generative likelihood of word sequences, to predict the probabilities of future (or missing) tokens. The research of LM can be roughly divided into four major development stages

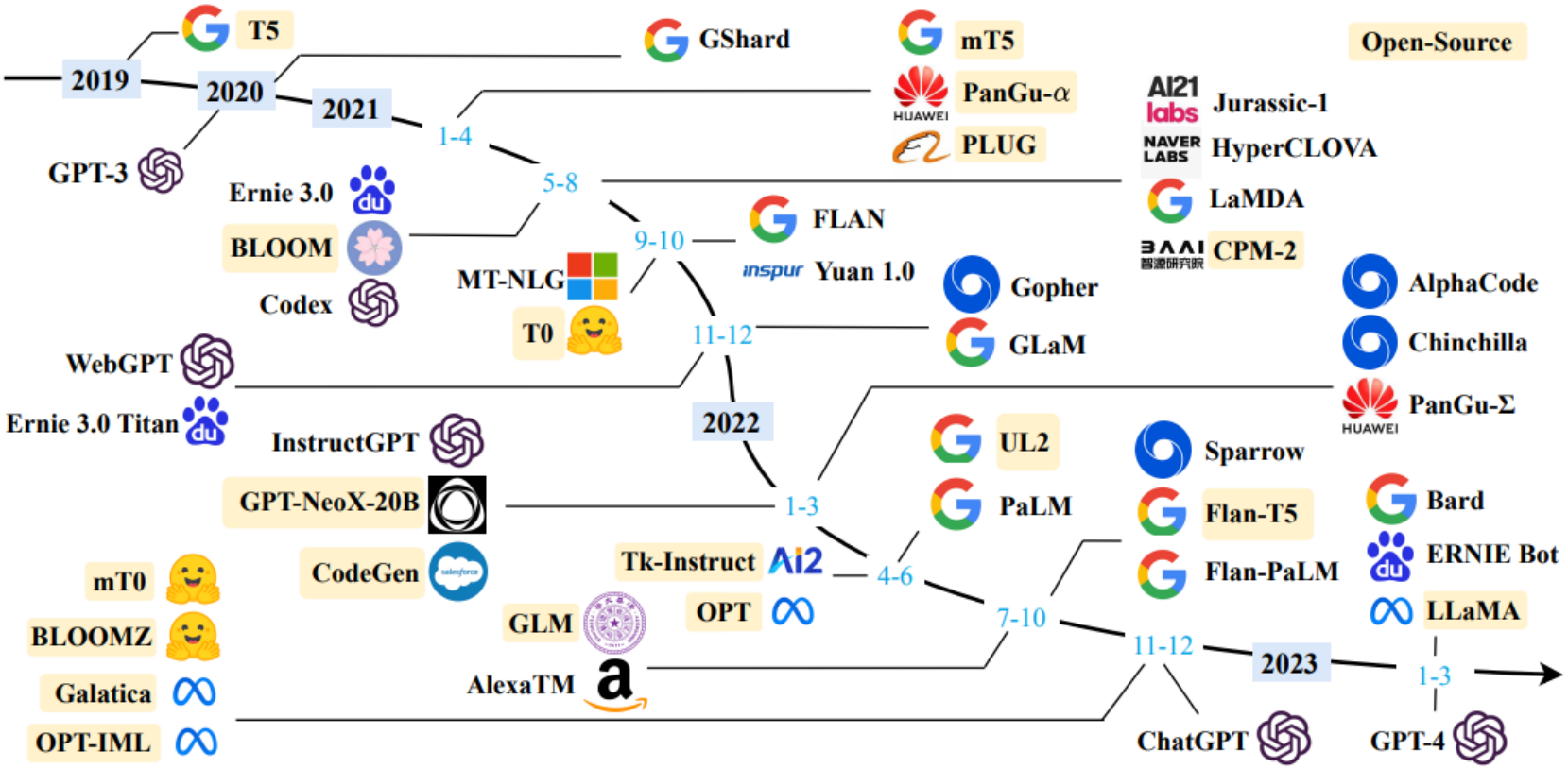
- **Statistical language models (SLM).** SLMs are developed based on statistical learning methods that rose in the 1990s. The basic idea is to build the word prediction model based on the Markov assumption, e.g., predicting the next word based on the most recent context. The SLMs with a fixed context length n are also called n-gram language models. It is difficult to accurately estimate high-order language models since an exponential number of transition probabilities need to be estimated.
- **Neural language models (NLM).** NLMs characterize the probability of word sequences by neural networks, e.g., recurrent neural networks (RNNs). NLM work introduced efficient distributed representation of words and modeled the context representation by aggregating the related distributed word vectors, so called word2vec. A general neural network approach built a unified solution for various NLP tasks.
- **Pre-trained language models (PLM).** As an early attempt, ELMo captured context-aware word representations by first pre-training a bidirectional LSTM (biLSTM) network (instead of learning fixed word representations) and fine-tuning the biLSTM network according to specific downstream tasks. Further, based on the highly parallelizable Transformer architecture with self-attention mechanisms, BERT is a pre-trained bidirectional language model with specially designed pre-training tasks on large-scale unlabeled corpora. These pre-trained context-aware word representations are very effective as general-purpose semantic features, which have largely raised the performance bar of NLP tasks.

Brief History, Large language models (LLM)

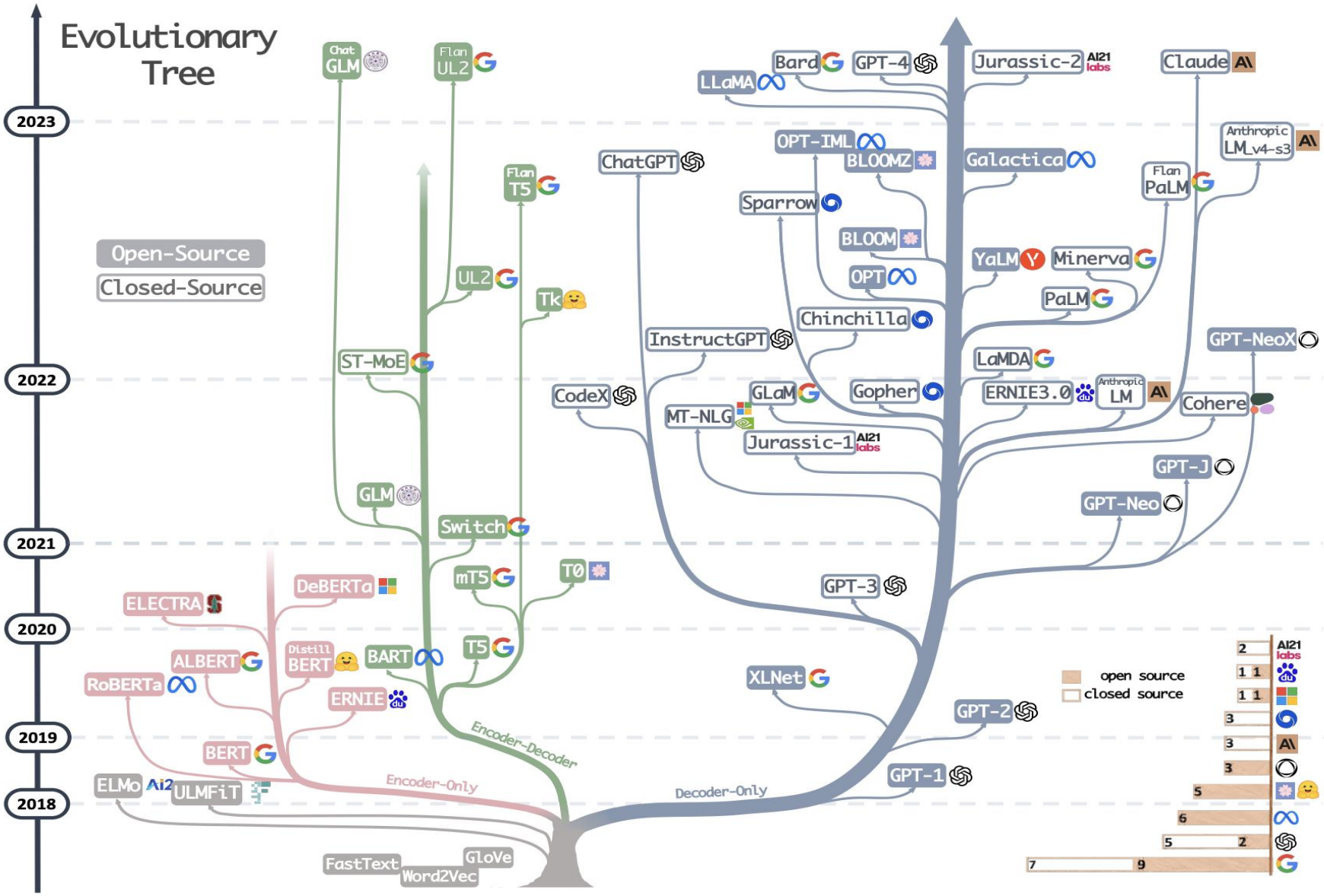
- It was found that scaling PLM (e.g., scaling model size or data size) often leads to an improved model capacity on downstream tasks. A number of studies have explored the performance limit by training an ever larger PLM.
- Although scaling is mainly conducted in model size (with similar architectures and pre-training tasks), these large-sized PLMs display different behaviors from smaller PLMs (e.g., 330M-parameter BERT and 1.5B parameter GPT-2) and show surprising abilities (called emergent abilities) in solving a series of complex tasks.
- For example, GPT-3 can solve few-shot tasks through in-context learning, whereas GPT-2 cannot do well. Thus, the research community coins the term "*large language models (LLM*)" 1 for these large-sized PLMs.
- ChatGPT adapts the LLMs from the GPT series for dialogue, which possesses ability to converse with humans.
- The training of LLMs requires extensive practical experiences in large-scale data processing and distributed parallel training.
- LLMs lead to the rethinking of the possibilities of *artificial general intelligence* (AGI). OpenAI has published a technical article entitled "Planning for AGI and beyond" https://openai.com/blog/planning-for-agi-and-beyond , which discussed the short-term and long-term plans to approach AGI. A more recent paper, Sparks of Artificial General Intelligence: Early experiments with GPT-4, by Sebastian Bubeck et al. **,** https://arxiv.org/abs/2303.12712 has argued that GPT-4 might be considered as an early version of an AGI system

Timeline of Modern LLMs

A timeline of existing large language models (having a size larger than 10B) in recent years. The open-source LLMs are marked in yellow color.



Modern LLMs Evolution and Timeline



https://arxiv.org/pdf/2304.13712

LLM Architectures

Encoder-only (BERT)

- Pre-training : Masked Language Modeling (MLM)
- Great for classification tasks, but hard to do generation

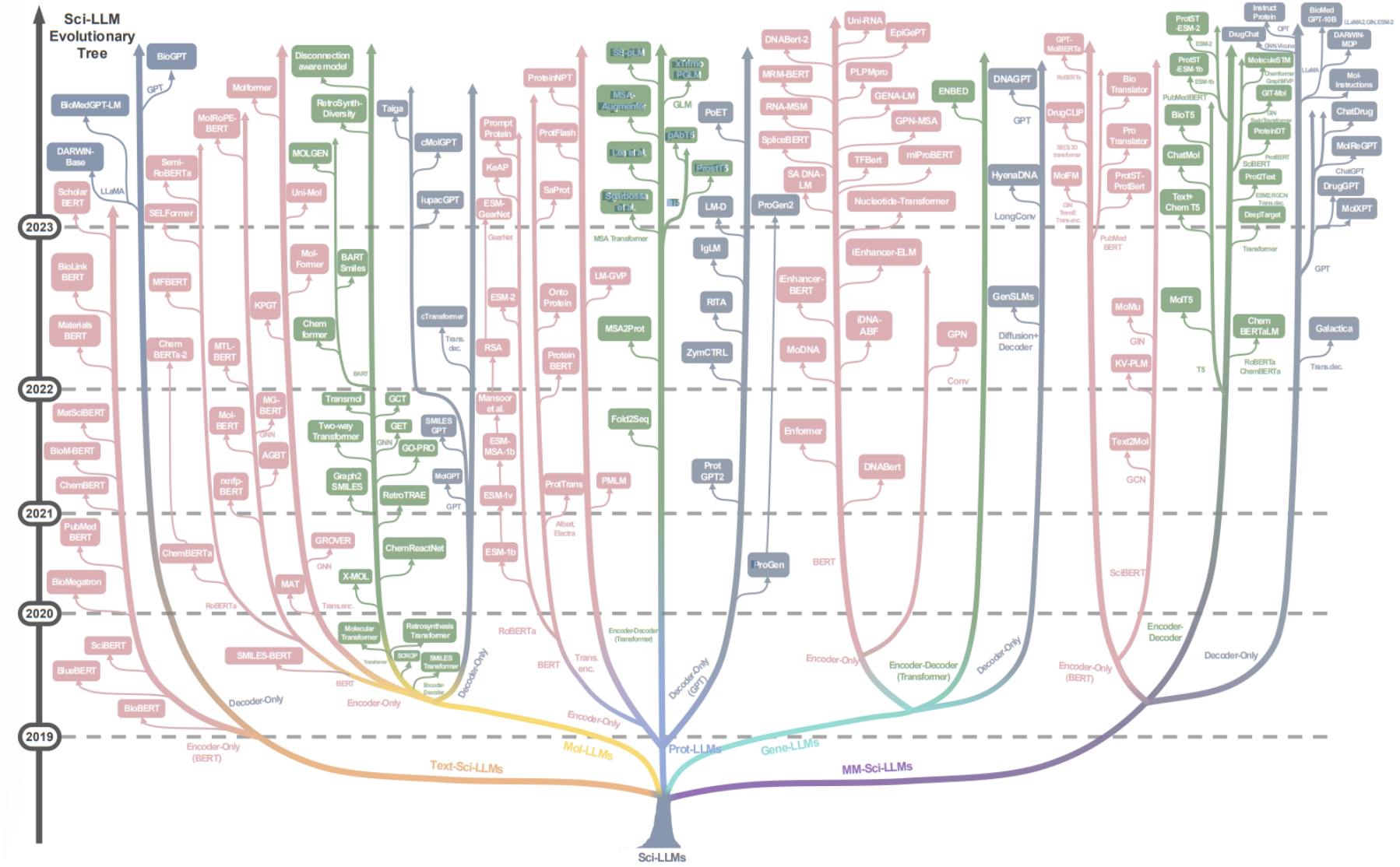
Decoder-only (GPT)

- Pre-training: Auto-regressive Language Modeling
- Stable training, faster convergence
- Better generalization after pre-training

Encoder-decoder (T0/T5)

- Pre-training : Masked Span Prediction
- Good for tasks like MT, summarization

Growth of LLMs in Sciences



https://arxiv.org/pdf/2401.14656v1

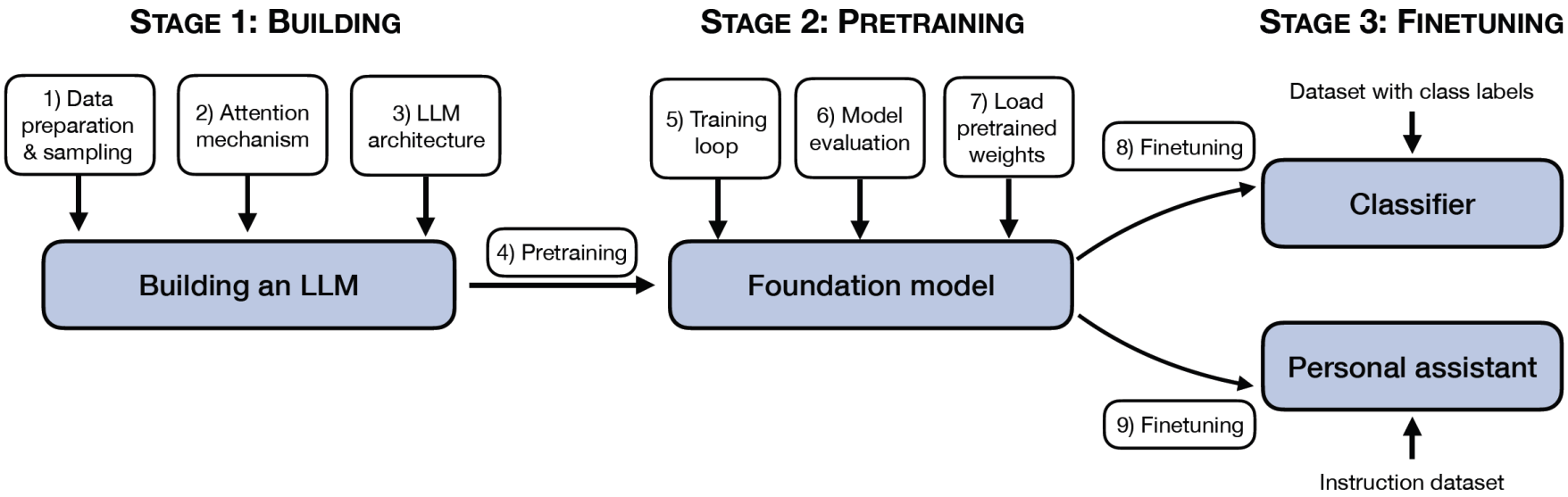
Common Commercial LLMs

CRITERIA	ChatGPT	Gemini	Claude	Mistral	LlaMA
DEVELOPER	OpenAI	Google	Anthropic	Mistral AI	Meta
RELEASE DATE	Nov. 2022	Dec. 2023	Mar. 2023	Sept. 2023	Feb. 2023
LANGUAGE MODEL	GPT 4o	Gemini 1.5 Pro	Claude 3 Opus	Mixtral 8x22B	Llama 3 (8B)
OUTPUT TOKEN PRICE	$15.00 per 1M Tokens	$21 per 1M Tokens	$75.00 per 1M Tokens	$1 per 1M Tokens	$0.1 per 1M Tokens
SPEED	74 Tokens per Second	55 Tokens per Second	32 Tokens per Second	82 Tokens per Second	866 Tokens per Second
QUALITY INDEX	100	88	94	63	65
KEY FEATURE	Generates human-like response in real time based on user-input.	Understand different types of information, including text, images, audio video & code.	Generates various forms of text content like summary, creative works & code.	It can grasp the nuances of language, context, and even emotions.	It has advanced NLP capabilities that can handle complex queries easily.

Public APIs of LLMs

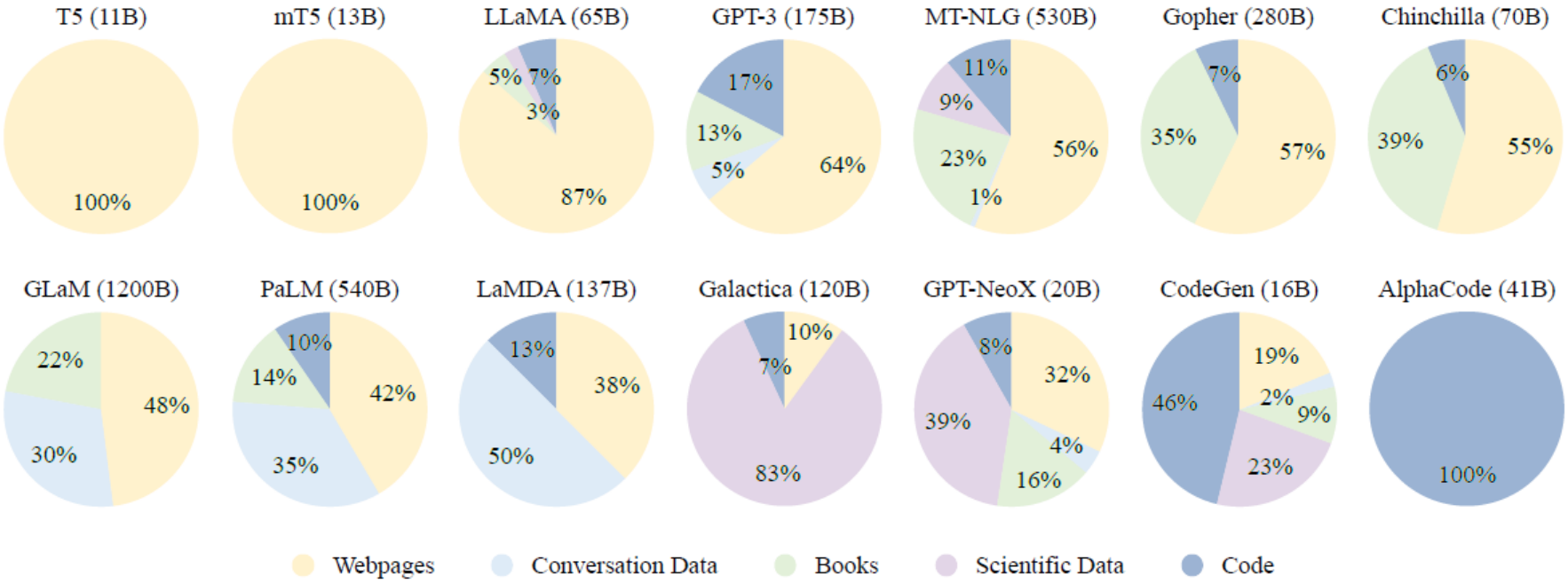
- Instead of directly using the model copies, APIs provide a more convenient way for common users to use LLMs, without the need of running the model locally.
- OpenAI has provided seven major interfaces to the models in GPT-3 series: `ada`, `babbage`, `curie`, `davinci` (the most powerful version in GPT-3 series), `text-ada-001`, `text-babbage-001`, and `text-curie-001`. The first four interfaces can be further fine-tuned on the host server of OpenAI.
- In particular, `babbage`, `curie`, and `davinci` correspond to the GPT-3 (1B), GPT-3 (6.7B), and GPT-3 (175B) models, respectively [55].
- There are also two APIs related to Codex [87], called `code-cushman-001` (a powerful and multilingual version of the Codex (12B) [87]) and `code-davinci-002`.
- GPT-3.5 series include one base model `code-davinci-002` and three enhanced versions, namely `text-davinci-002`, `text-davinci-003`, and `gpt-3.5-turbo-0301`. It is worth noting that `gpt-3.5-turbo-0301` is the interface to invoke ChatGPT.
- OpenAI has released APIs for GPT-4, including `gpt-4`, `gpt-4-0314`, `gpt-4-32k`, and `gpt-4-32k-0314`.
- The choice of API interfaces depends on the specific application scenarios and response requirements. The detailed usage can be found on OpenAI website.

Steps in Building the LLMs- 50k feet view



Data sources in Pre-Training for Existing LLMs

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



Commonly Used Corpora

LLMs which consist of a very larger number of parameters require a high volume of training data that covers a broad range of content. The corpora used in LLMs training falls in six groups: Books, Common Crawl, Reddit links, Wikipedia, Code, and others.

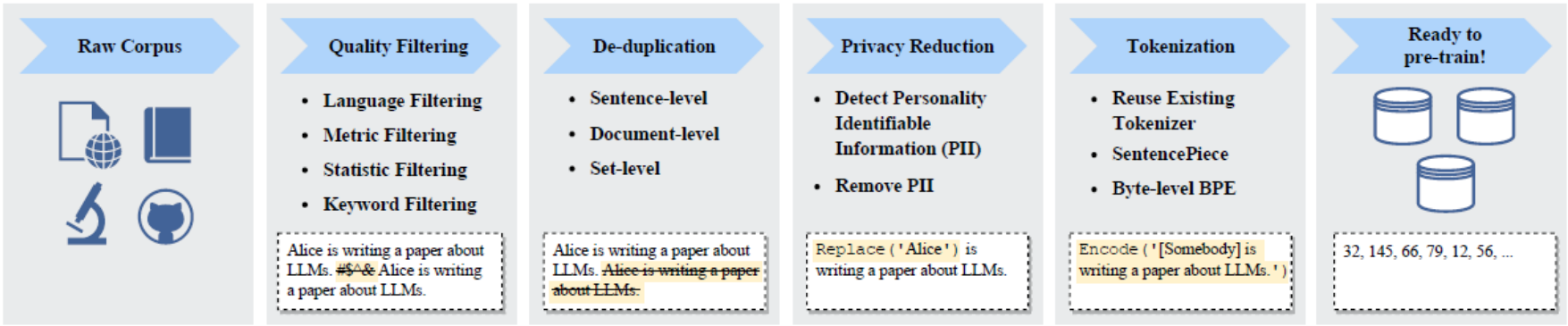
- **Books**: BookCorpus is a commonly used dataset in previous small-scale models consisting of over 11,000 books covering a wide range of topics and genres (e.g., novels and biographies).
- Another large-scale book corpus is **Project Gutenberg**, consisting of over 70,000 literary books including novels, essays, poetry, drama, history, science, philosophy, and other types of works in the public domain. It is currently one of the largest open-source book collections, which is used in training of MT-NLG and LLaMA.
- Books1 [55] and Books2 [55] used in GPT-3, they are much larger than BookCorpus but have been not publicly released so far.
- **CommonCrawl**. CommonCrawl is one of the largest open-source web crawling databases, containing a petabyte scale data volume. As the whole dataset is very large, existing studies mainly extract subsets of web pages from it within a specific period. However, due to the widespread existence of noisy and low-quality information in web data, it is necessary to perform data preprocessing before usage. Based on CommonCrawl, there are four filtered datasets that are commonly used in existing work: C4, CCStories, CC-News, and RealNews .
- **Reddit Links.** Reddit is a social media platform that enables users to submit links and text posts, which can be voted on by others through “upvotes” or “downvotes”. Highly upvoted posts are often considered useful, and can be utilized to create high-quality datasets. **WebText** is a corpus composed of highly upvoted links from Reddit, but it is not publicly available. There is a readily accessible open-source alternative called **OpenWebText**

Commonly Used Corpora

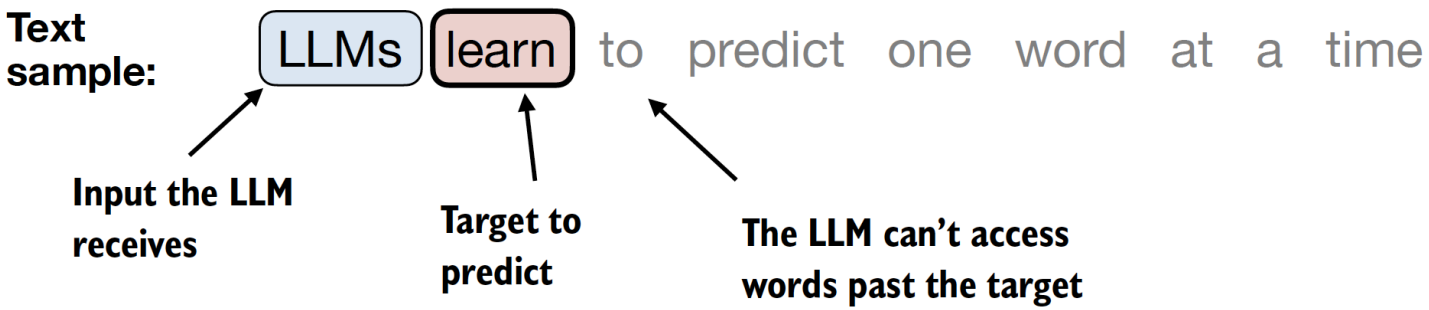
- **Wikipedia**. Wikipedia is an online encyclopedia containing a large volume of high-quality articles on diverse topics. Most of these articles are composed in an expository style of writing (with supporting references), covering a wide range of languages and fields. Typically, the English only filtered versions of Wikipedia are widely used in most LLMs (e.g., GPT-3, LaMDA, and LLaMA). Wikipedia is available in multiple languages, so it can be used in multilingual settings.
- **Code.** To collect code data, existing work mainly crawls open-source licensed codes from the Internet. Two major sources are public code repositories under open-source licenses (e.g., GitHub) and code-related question-answering platforms (e.g., StackOverflow). Google has publicly released the BigQuery dataset, which includes a substantial number of open-source licensed code snippets in various programming languages, serving as a representative code dataset. CodeGen has utilized BIGQUERY, a subset of the BigQuery dataset, for training the multilingual version of CodeGen (CodeGen-Multi).
- **Others. The Pile** is a large-scale, diverse, and open source text dataset consisting of over 800GB of data from multiple sources, including books, websites, codes, scientific papers, and social media platforms. It is constructed from 22 diverse high-quality subsets. The Pile dataset is used in models with different parameter scales, such as GPT-J (6B), CodeGen (16B), and MegatronTuring NLG (530B). Besides, ROOTS is composed of various smaller datasets (totally 1.61 TB of text) and covers 59 different languages (containing natural languages and programming languages), which have been used for training BLOOM.

Data preprocessing pipeline for Pre-training

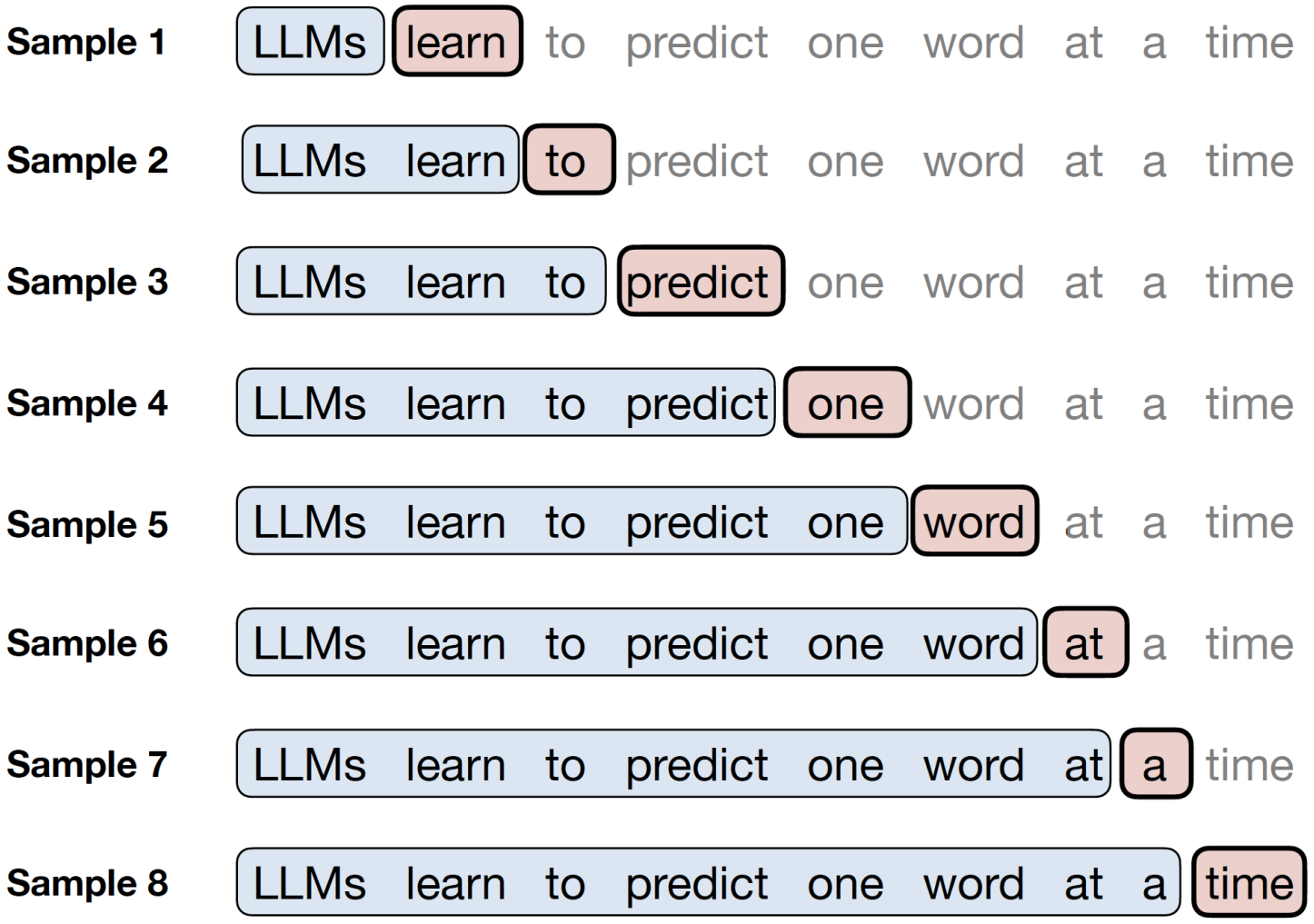
- **Quality Filtering**. To remove low-quality data from the collected corpus, existing work generally adopts two approaches: (1) classifier-based, and (2) heuristic-based. The former approach trains a selection classifier based on high-quality texts and leverages it to identify and filter out low-quality data. Typically, these methods train a binary classifier with well-curated data. An illustration of a typical data preprocessing pipeline for pre-training large language models.
- **De-duplication.** Existing work has found that duplicate data in a corpus would reduce the diversity of language models, which may cause the training process unstable and thus affect the model performance. Therefore, it is necessary to de-duplicate the pre-training corpus.
- **Privacy Redaction.** The majority of pre-training text data is obtained from web sources, including sensitive or personal information, which may increase the risk of privacy breaches. Thus, it is necessary to remove the personally identifiable information (PII) from the pre-training corpus.
- **Tokenization.** The byte-level Byte Pair Encoding (BPE) algorithm is utilized to ensure that the information after tokenization is lossless



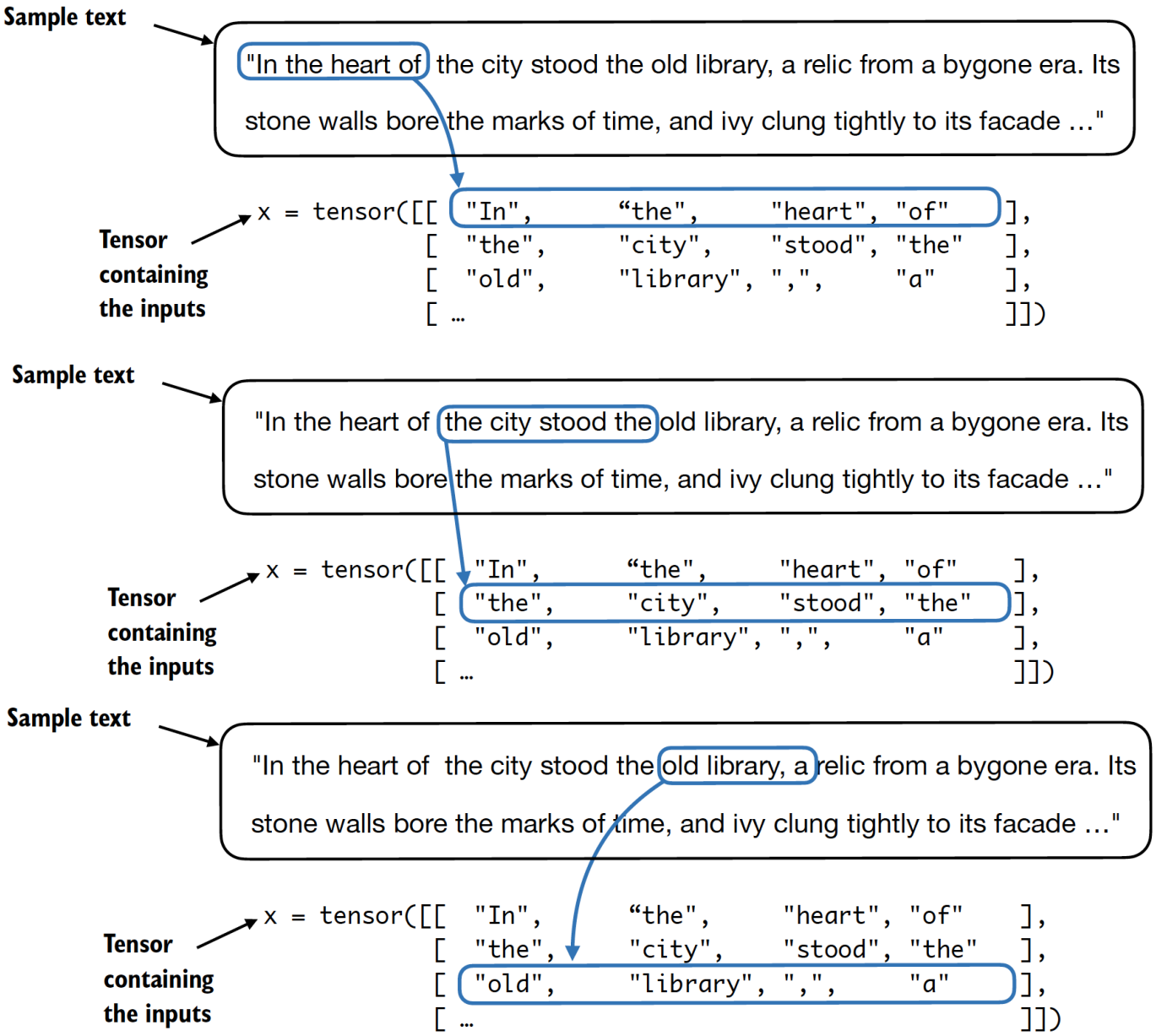
LLMs Next Token prediction



LLMs Next Token prediction

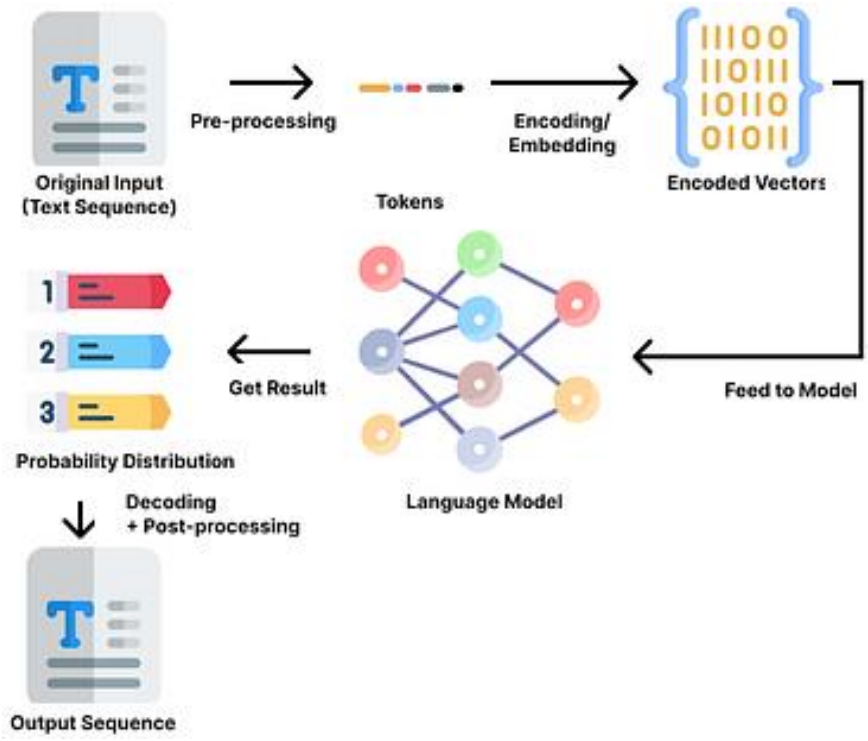


It happens in batches …



Language Models & NLP

- Language models read and learn a library of text (called corpus) and model words or sequences of words with probabilistic distributions.
- Models tell us how likely a word or sequence can occur. For example, when you say: “Tom likes to eat …”, the probability of the next word being “pizza” would be higher than “table”.
- If the model is predicting the next word in the sequence, it is called *next-token-prediction*.
- If the model is predicting a missing word in the sequence, we call it *masked language model*.
- In a probability distribution, there can be many probable words with different probabilities. It is not ideal to always choose as the best candidate the word with the highest probability. That may lead to repetitive sequences. In practice, we add some randomness (temperature) when choosing the word from the top candidates.



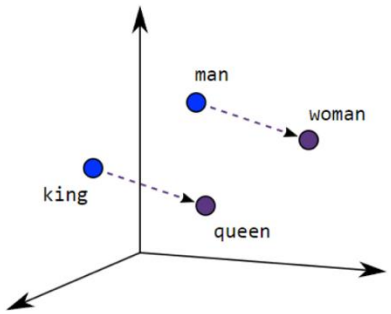
Typical NLP Text Generation Process

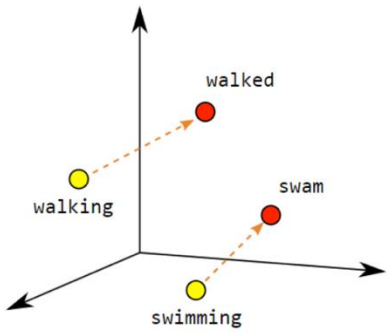
In a typical NLP process, the input text will go through the following steps [8]:

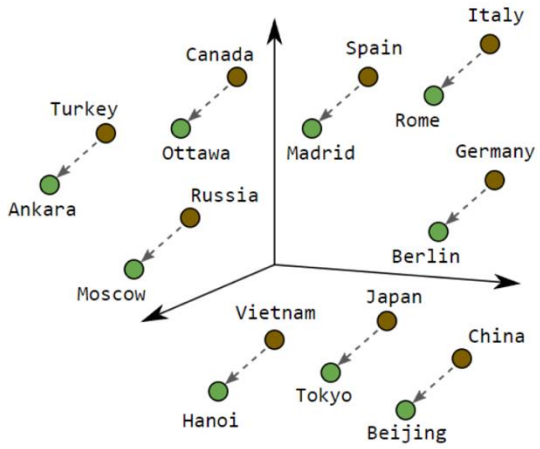
- **Preprocess:** clean the text with techniques like sentence segmentation, tokenization (breaking down the text into small pieces called tokens), stemming (removing suffixes or prefixes), removing stop words, correcting spelling, etc. For example, “Tom likes to eat pizza.” would be tokenized into [“Tom”,“likes”, “to”, “eat”, “pizza”, “.”] and stemmed into [“Tom”, “like”, “to”, “eat”, “pizza”, “.”].
- **Encode or embed**: turn the cleaned text into a vector of numbers, which could be processed by neural networks.
- **Feed to the model**: pass the encoded input to the model for processing.
- **Get result:** from the model, get the probability distribution of potential words represented as vectors of numbers.
- **Decode:** translate collected vectors back to human-readable words.
- **Post-process:** refine the output with spell checking, grammar checking, punctuation, capitalization, etc.
- Over the years we developed many different model architectures. ***Transformers*** has been the state-of-the-art architecture for several years and lays the foundation for GPT.

Embeddings

- ***Word embedding*** in NLP is a technique where individual words are represented as real-valued vectors in a low dimensional space (200, 512, 1024 are low dimensions). Word embedding captures inter-word semantics.
- Each word is represented by a real-valued vector with several hundreds to thousand of dimensions.
- Embeddings or embedded vectors were introduced with Word2Vec method developed by Tomas Mikolov [13] at Google.
- In embedded spaces, words with similar meaning are represented by embedded vectors which have high cosine similarity.
- Representing words and texts by embedded vectors is a key preprocessing step in all modern NLP analysis including Deep Learning and LLM.

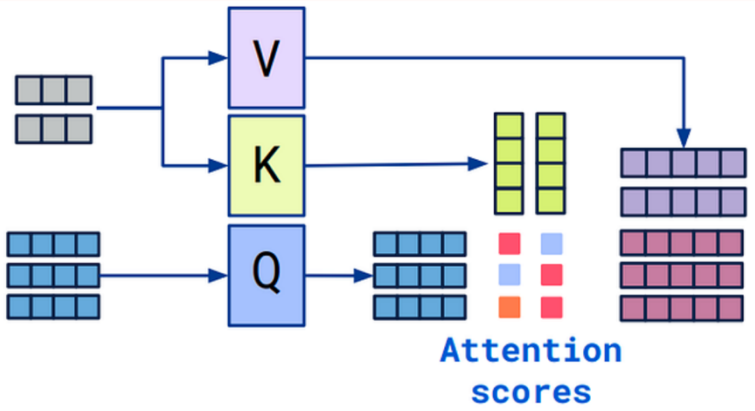




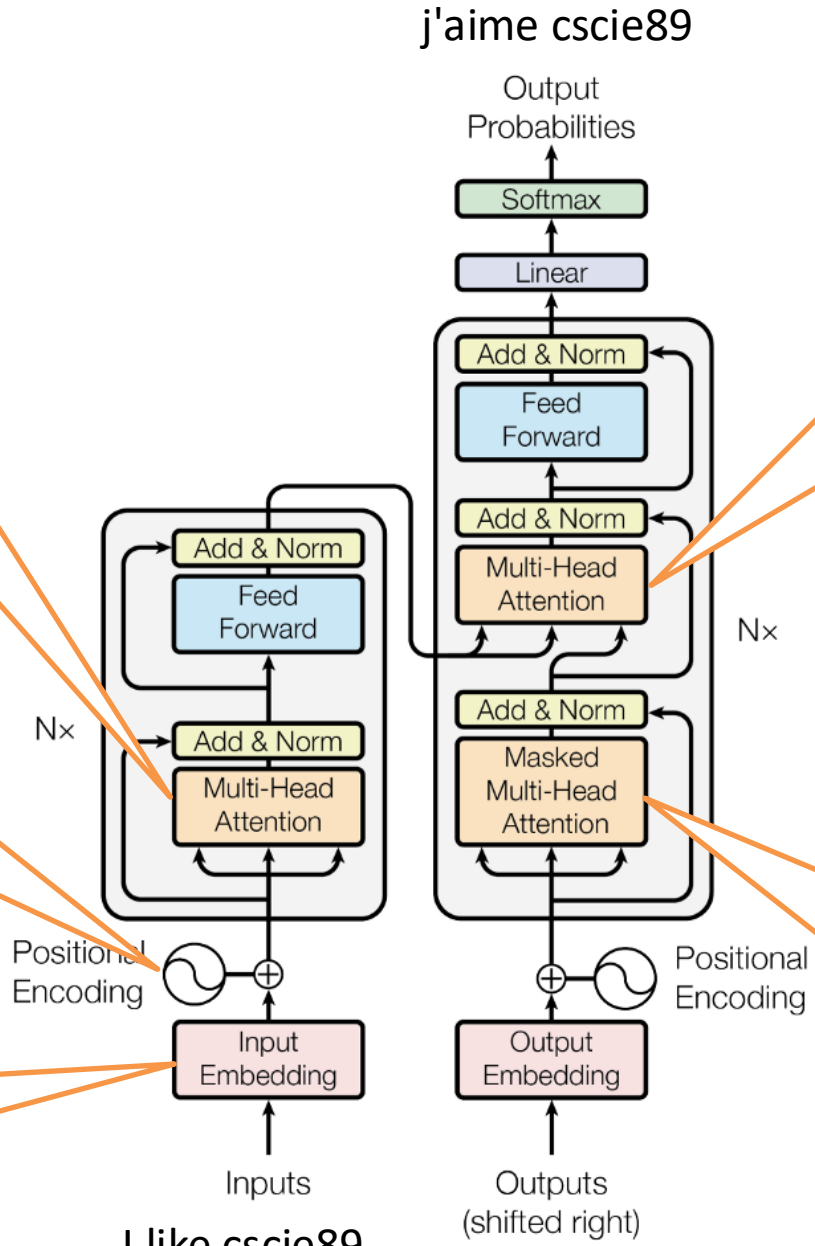


Transformers and Attention

- The transformer architecture is the foundation of GPT.
- The transformer uses the mechanisms called attention to understand context of relevant events (words, tones, other) in sequential data such as text, speech, or music.
- Attention allows the model to tailor the output by focusing on the most relevant parts of the input by learning the relevance or similarity between the elements of output and input. Those elements are usually represented by vectors. If the attention focuses on the same sequence, it is called self-attention [2][9].
- "Tom likes to eat apples. He eats them every day." In this sentence, "he" refers to "Tom" and "them" refers to "apples".
- The attention mechanism calculates a similarity score between the word vectors. With this mechanism, transformers can better "make sense" of the meanings in the text sequences in a more coherent way.



Transformer Architecture



Represents how much each word in the English sentence is related to every word in the same sentence

Encodes information about the position of the input embedding in the sequence to get a notion of context

Maps words to a latent space - similar words are mapped together

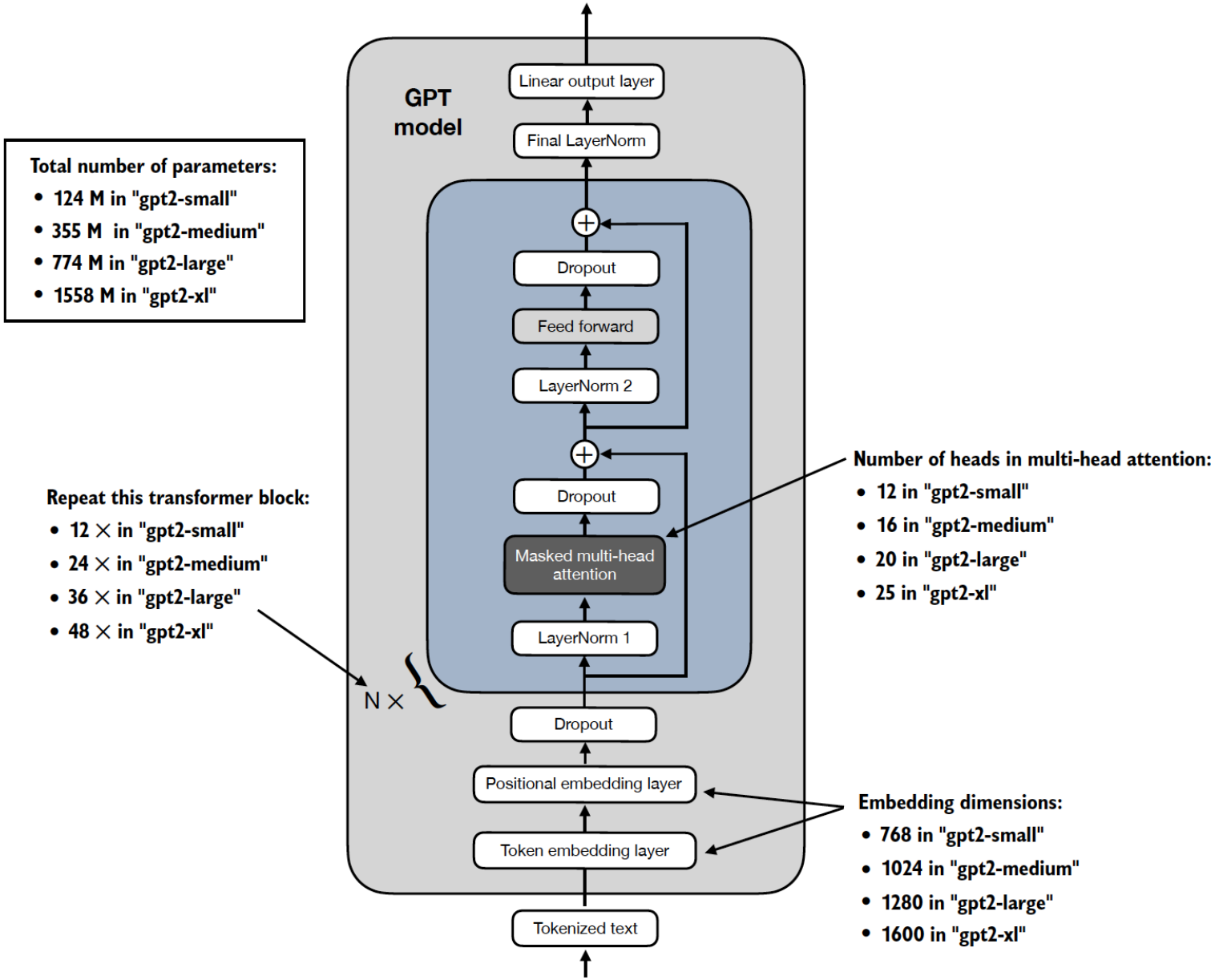
Represents how much each word in the French sentence is related to every word in the English sentence.

Represents how much each word in the French sentence is related to every word in the same sentence.

Architectural Elements of Transformer

- The encoder is a stack of multiple identical modules or layers (6 in the original transformer paper).
- Each layer has two sub-layers: a multi-head self-attention layer and a feed-forward layer, with some connections, called residual connection and layer normalization [2].
- The multi-head self-attention sub-layer applies the attention mechanism to find the connection/similarity between input tokens to understand the input.
- The feed-forward sub-layer does some processing before passing the result to the next layer to prevent overfitting.
- You can think of encoders like reading books — you will pay attention to each new word you read and think about how it is related to the previous words.
- The decoder is similar to the encoder in that it is also a stack of identical layers. But each decoder layer has an additional encoder-decoder attention layer between the self-attention and feed-forward layers, to allow the decoder to attend to the input sequence. For example, if you're translating "`I love you`" (input) to "`Je t'aime`" (output), you will need to know "`Je`" and "`I`" are aligned and "`love`" and "`aime`" are aligned.
- The multi-head attention layers in the decoder are also different. They're masked to not attend to anything to the right of the current token, which has not been generated yet [2].
- You can think of decoders like free-form writing — you write based on what you've written and what you've read, without caring about what you're going to write.

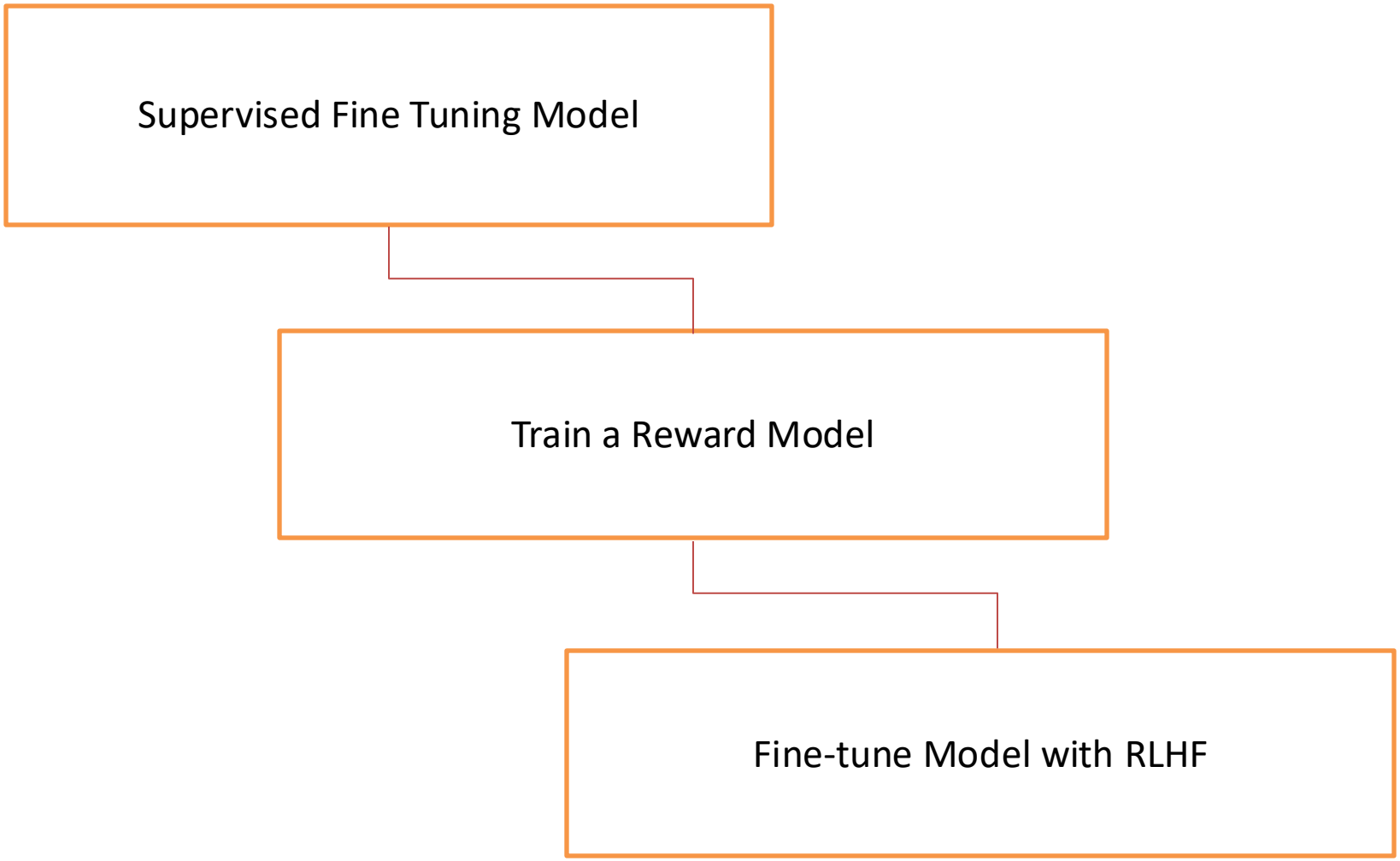
Transformer within GPT2 Architecture



RLHF System

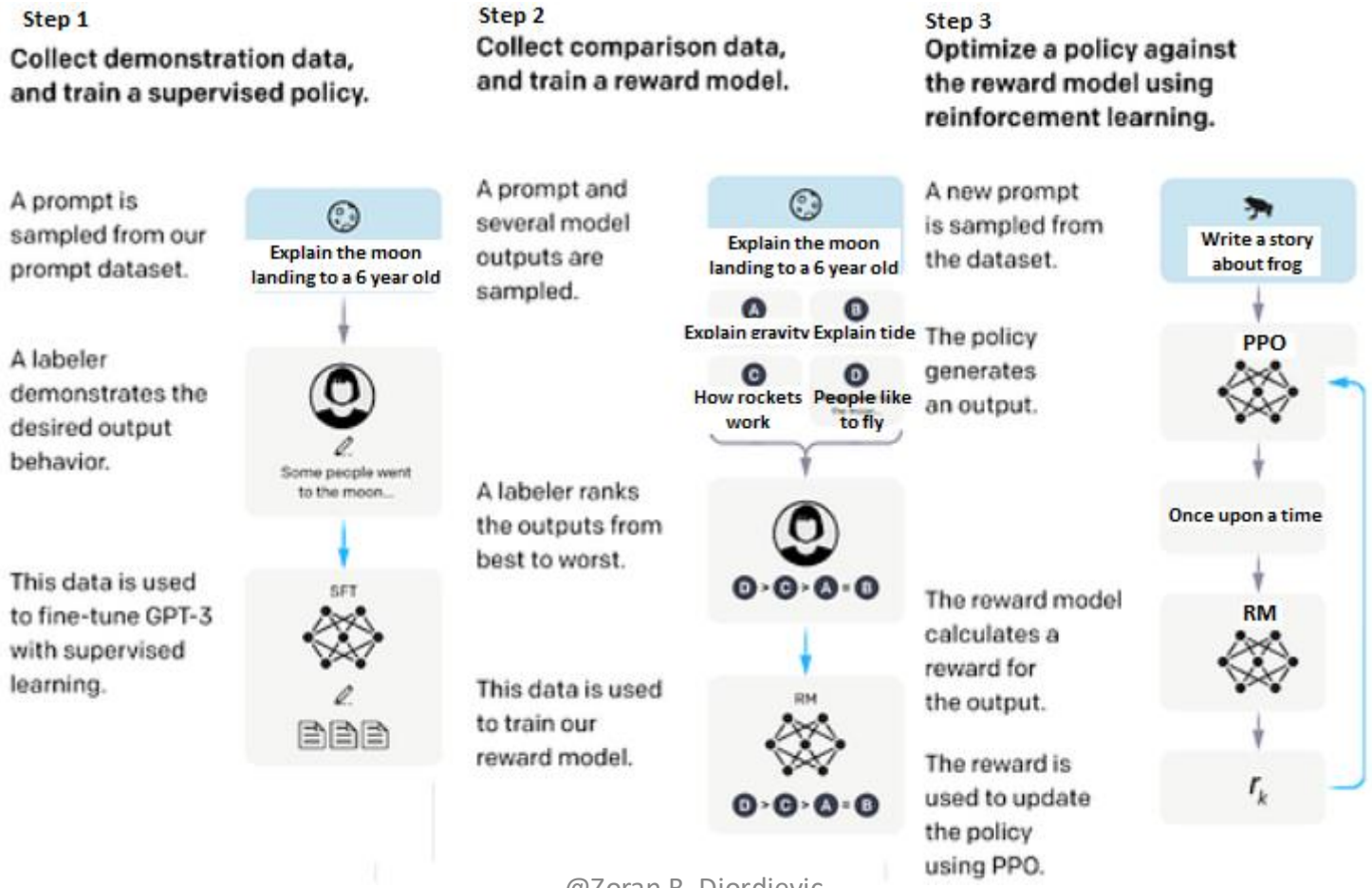
- The RLHF system mainly comprises three key components:
  - a pre-trained LM to be aligned,
  - reward model learning from human feedback, and
  - a RL algorithm training the LM.
- Specifically, the pre-trained LM is typically a generative model that is initialized with existing pre-trained LM parameters. For example, OpenAI uses 175B GPT-3 for its first popular RLHF model, InstructGPT , and DeepMind uses the 280 billion parameter model Gopher for its GopherCite model.
- The reward model (RM) provides (learned) guidance signals that reflect human preferences for the text generated by the LM, usually in the form of a scalar value. The reward model can take on two forms: a fine-tuned LM or a LM trained de novo using human preference data.
- Existing work typically employs reward models having a parameter scale different from that of the aligned LM. For example, OpenAI uses 6B GPT-3 and DeepMind uses 7B Gopher as the reward model, respectively.
- Finally, to optimize the pre-trained LM using the signal from the reward model, a specific RL algorithm is designed for large-scale model tuning. Specifically, Proximal Policy Optimization (PPO) is a widely used RL algorithm for alignment in existing work.

Instruction Fine Tuning



Teaching GPT to interact with humans

- At this stage, the GPT-3 model is more like general-purpose language model. Open AI wanted to explore how GPT can follow human instructions and have conversations with humans.
- They attempted to fine-tune GPT-3 with supervised learning and reinforcement learning from human feedback (RLHF) [6][12]. With these training steps came the two fine-tuned models — InstructGPT and ChatGPT. Proximal Policy Optimization, or PPO, is a policy gradient method.



Step 1

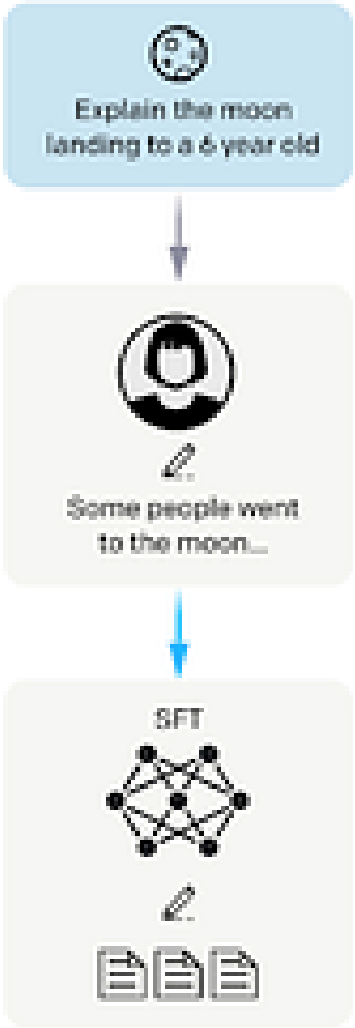
Collect demonstration data and train a supervised policy

- The first step is supervised learning from human examples.
- Researchers first provided the pre-trained GPT with a curated, labeled dataset of prompt and response pairs written by human labelers.
- This dataset is used to let the model learn the desired behavior from those examples.
- From this step, they get a **supervised fine-tuned** (SFT) model.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



Step 2

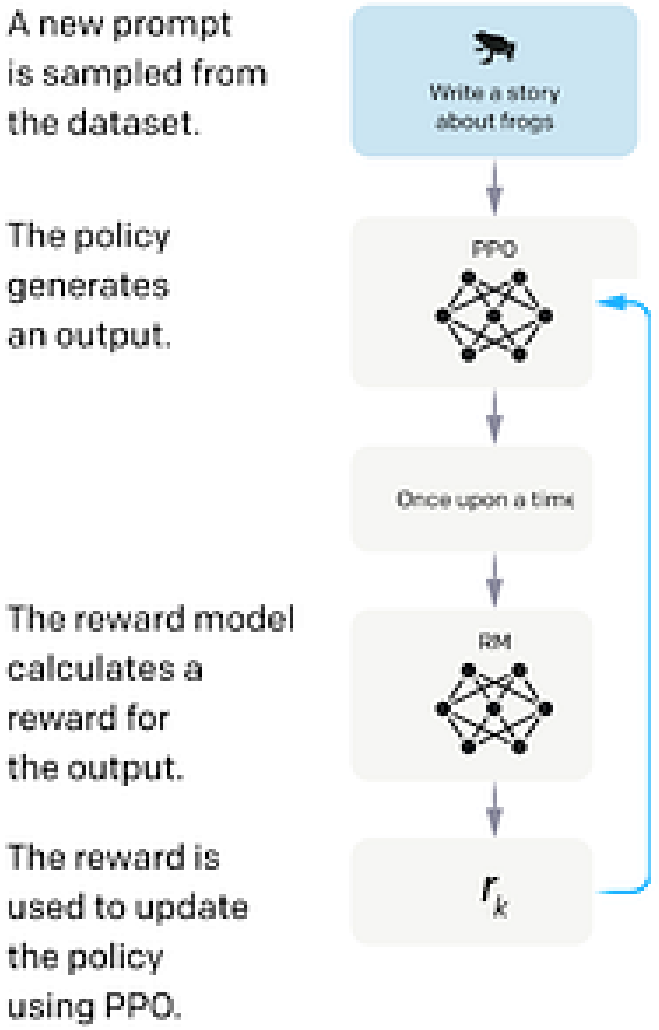
Collect comparison Data and train a reward model

- The second step is training a **reward model** (RM) to rate the responses from the generative model.
- Researchers used the SFT model to generate multiple responses from each prompt and asked human labelers to rank the responses from best to worse by quality, engagement, informativeness, safety, coherence, and relevance.
- The prompts, responses, and rankings are fed to a reward model to learn human preferences of the responses through supervised learning.
- The reward model can predict a scalar reward value based on how well the response matches human preferences.

Step 3

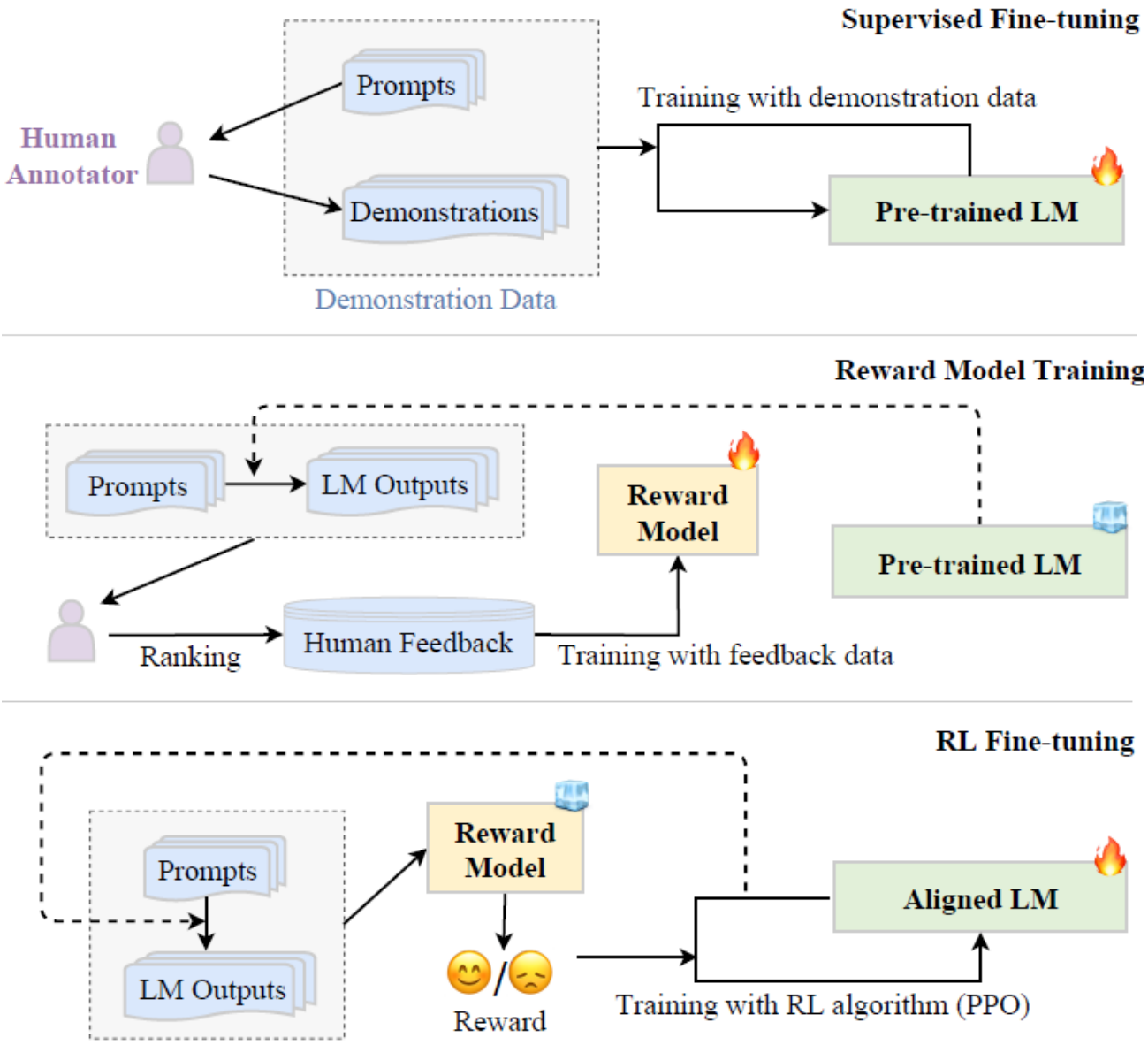
Optimize a policy against the reward model using reinforcement learning

- In the third step, researchers used the reward model to optimize the SFT model's policy through reinforcement learning.
- The SFT model will generate a response from a new prompt; the reward model will assess the response and give it a reward value that approximates human preference; the reward is then used to optimize the generative model by updating its parameters.
- For example, if the generative model generates a response that the reward model thinks humans may like, it will get a positive reward to continue generating similar responses in the future; and vice versa.
- Through this process with supervised learning and reinforcement learning from human feedback, the InstructGPT model (with only 1.3B parameters) is able to perform better in tasks that follow human instructions than the much bigger GPT-3 model (with 175 B parameters).



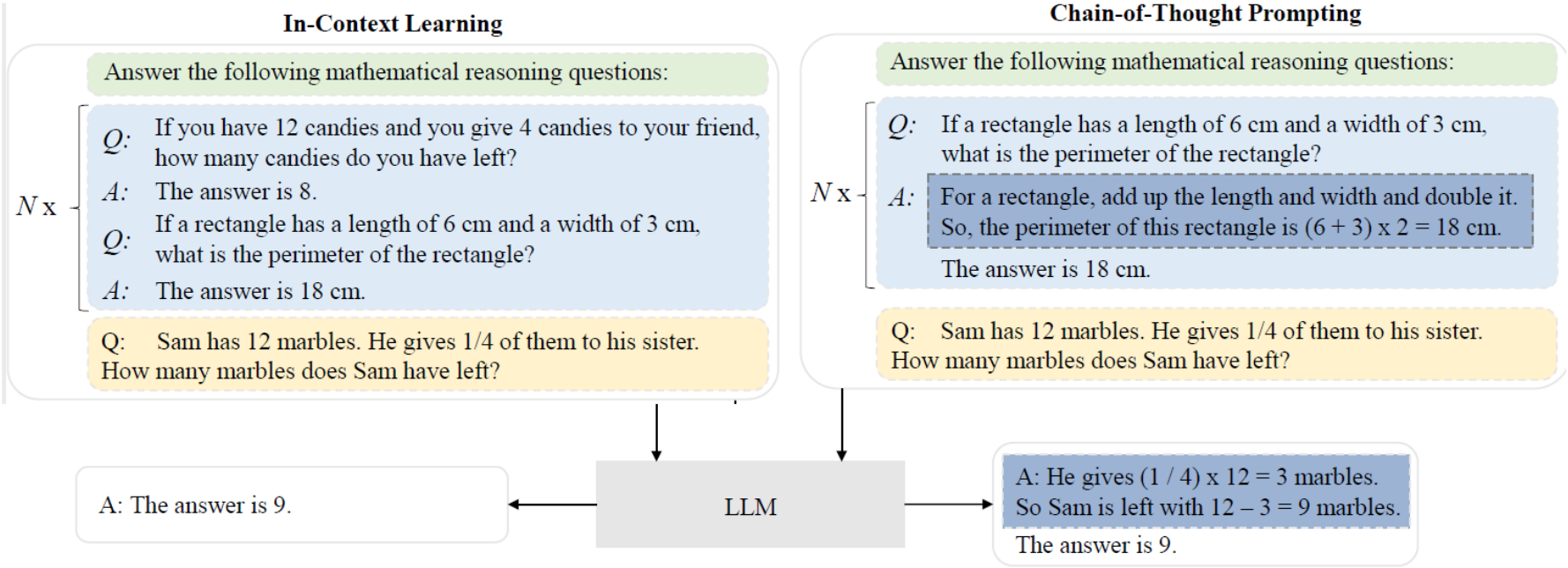
The workflow of the RLHF algorithm.

- We need to collect a supervised dataset with input prompts and outputs for ***supervised fine-tuning*** the LM.
- ***Reward model training.*** The second step is to train the RM using human feedback data.
- ***RL fine-tuning.*** At this step, aligning (i.e., fine-tuning) the LM is formalized as an RL problem. In this setting, the pre-trained LM acts as the policy that takes as input a prompt and returns an output text, the action space of this policy is all the terms in vocabulary of the LM, the state is characterized by the currently generated token sequence, and the reward is provided by the RM



In-Context Learning

- In Context Learning (ICL) uses a formatted natural language prompt, consisting of the task description and/or a few task examples as demonstrations. Figure bellow illustrated ICL.
- First, starting with a task description, a few examples are selected from the task dataset as demonstrations.
- Then, they are combined in a specific order to form natural language prompts with specially designed templates.
- Finally, the test instance is appended to the demonstration as the input for LLMs to generate the output. Based on task demonstrations, LLMs can recognize and perform a new task without explicit gradient update.



Communication with LLMs - Key Concepts

Prompts

- Designing the prompt is essentially how you “program” the model, usually by providing some instructions or a few examples. This is different from most other NLP services which are designed for a single task, such as sentiment classification or named entity recognition. Instead, the completions and chat completions endpoint can be used for virtually any task including content or code generation, summarization, expansion, conversation, creative writing, style transfer, and more.

Tokens

- OpenAI models understand and process text by breaking it down into tokens. Tokens can be words or just chunks of characters. For example, the word “hamburger” gets broken up into the tokens “ham”, “bur” and “ger”, while a short and common word like “pear” is a single token. Many tokens start with a whitespace, for example “ hello” and “ bye”.
- The number of tokens processed in a given API request depends on the length of both your inputs and outputs. As a rough rule of thumb, 1 token is approximately 4 characters or 0.75 words for English text. One limitation to keep in mind is that your text prompt and generated completion combined must be no more than the model's maximum context length (for most models this is 2048 tokens, or about 1500 words). Check out our tokenizer tool to learn more about how text translates to tokens.

Models

- The API is powered by a set of models with different capabilities and price points. GPT-4o is the latest and most powerful model.
- GPT-4 is the model that powers ChatGPT and is optimized for conversational formats.

Where are LLMs headed today?

Scaling Laws and Size - Llama 1

Llama 1 was trained on 1.4T tokens

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

LLaMA: Open and Efficient Foundation Language Models (2023), https://arxiv.org/abs/2302.13971

Scaling Laws and Size - Llama2

Llama 2 was trained on 2T tokens

"Our training corpus includes a new mix of data from publicly available sources, which does not include data from Meta's products or services. We made an effort to remove data from certain sites known to contain a high volume of personal information about private individuals. We trained on 2 trillion tokens of data as this provides a good performance–cost trade-off, up-sampling the most factual sources in an effort to increase knowledge and dampen hallucinations."

Llama 2: Open Foundation and Fine-Tuned Chat Models (2023), https://arxiv.org/abs/2307.09288

Scaling Laws and Size - Llama 3

Llama 3 was trained on 15T tokens

"To train the best language model, the curation of a large, high-quality training dataset is paramount. In line with our design principles, we invested heavily in pretraining data. Llama 3 is pretrained on over 15T tokens that were all collected from publicly available sources."

Introducing Meta Llama 3: The most capable openly available LLM to date (2024), https://ai.meta.com/blog/meta-llama-3/

Moving to higher Quality Data

Quantity vs quality, Phi-3 on 3.3T tokens

"we mainly focus on the quality of data for a given scale. We try to calibrate the training data to be closer to the "data optimal" regime for small models. In particular, we filter the publicly available web data to contain the correct level of "knowledge" and keep more web pages that could potentially improve the "reasoning ability" for the model. As an example, the result of a game in premier league in a particular day might be good training data for frontier models, but we need to remove such information to leave more model capacity for "reasoning" for the mini size models.

Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone (2024), https://arxiv.org/abs/2404.14219

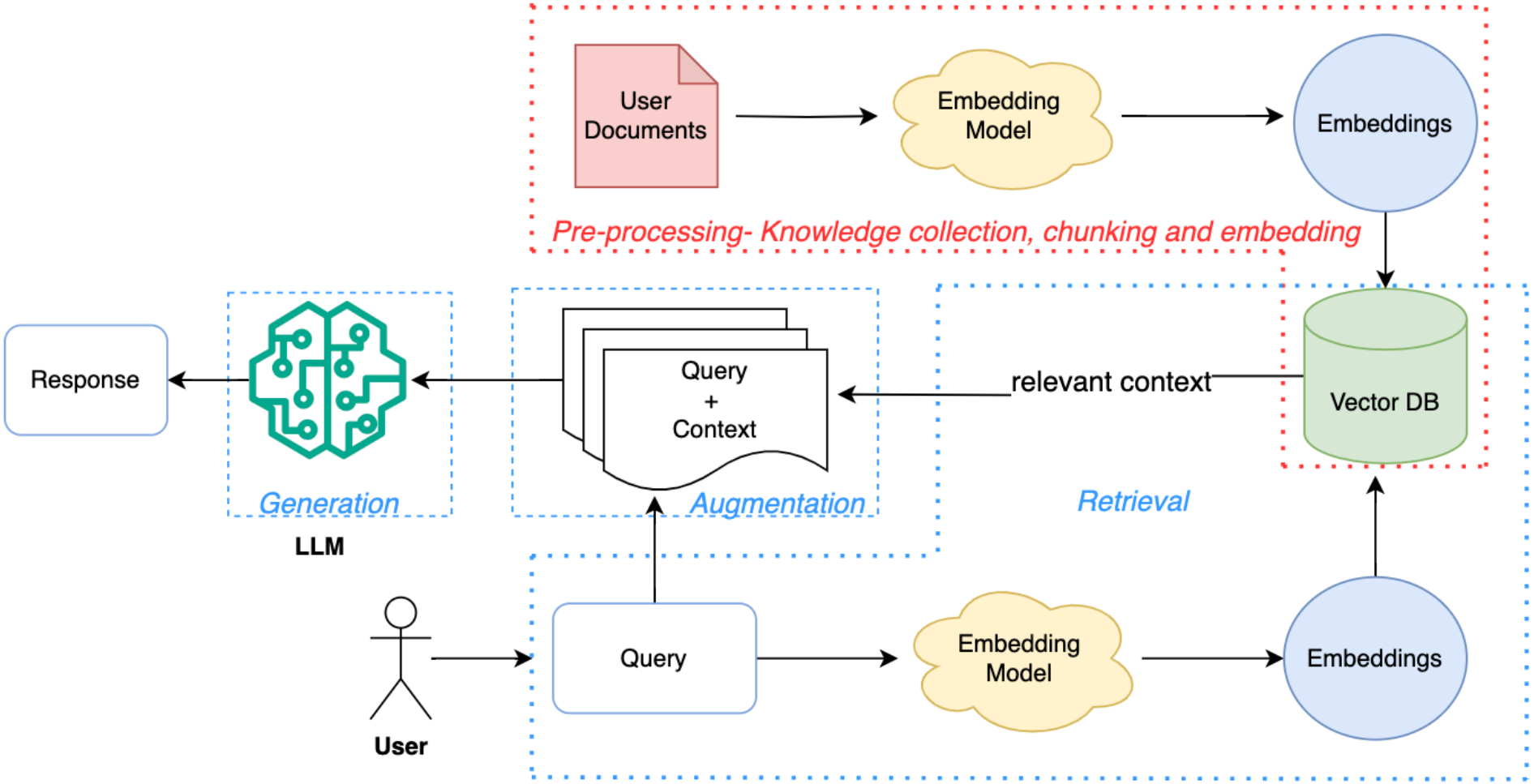
RAG

RAG stands for Retrieval-Augmented-Generation.

Why RAG ?

- In cases where accurate and detailed contextual information is needed RAG is the most commonly used pattern
- RAG minimizes the issue with Hallucination
- Highly useful for enterprise grade applications
- Relatively low cost solution to Pre-training or Instruction fine tuning
- Very common patten in the enterprise setting

RAG Architecture



RAG Architecture Components

Preprocessing Pipelines:

Chunk input documents (word/PDF's etc) to pass on to an embedding model

Embedding models:

Generate embeddings for contextual Reference documents

Vector Database:

Stores all Embeddings and allows to performs relevant searches as well as allows for a ranked search

LLM:

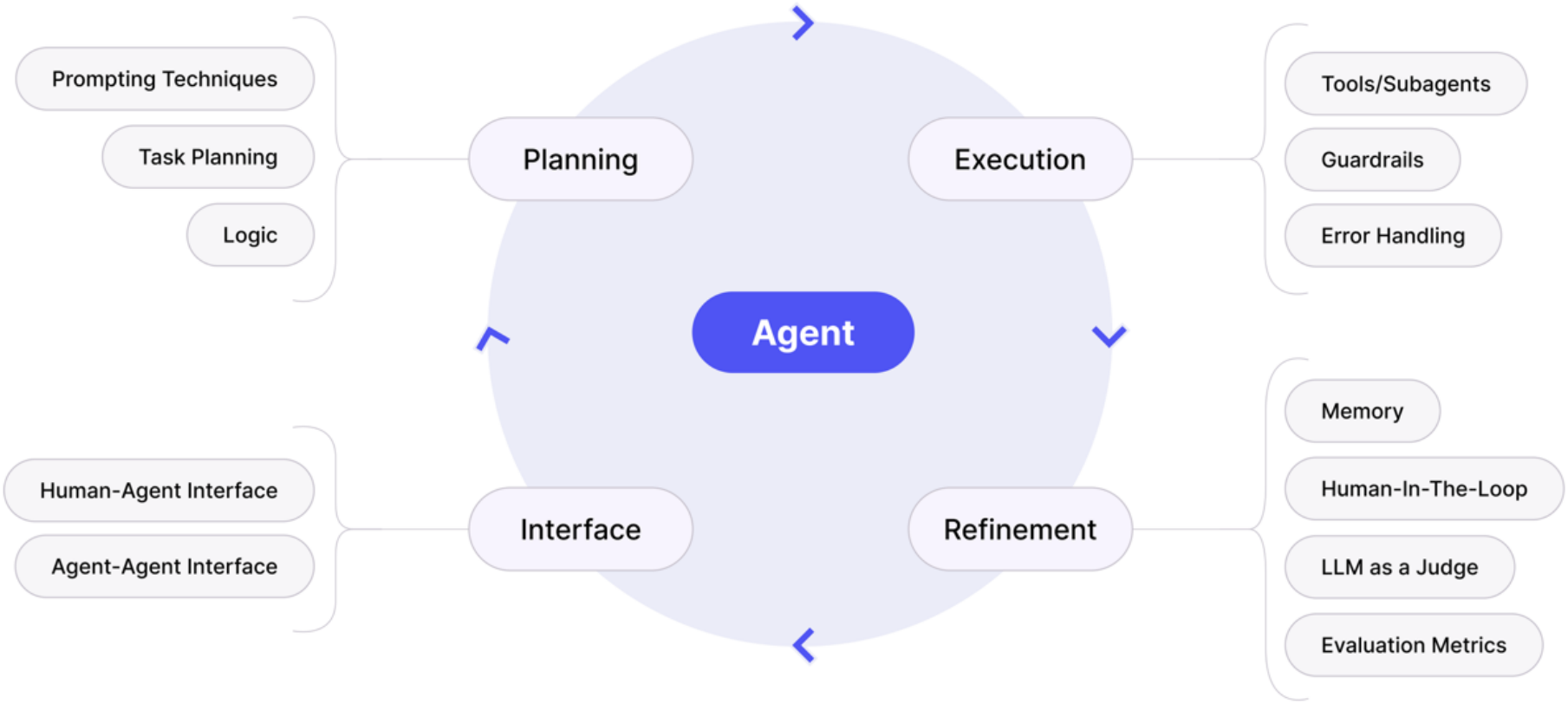
Accepts prompts and embeddings as inputs and generates an O/P relevant to the input prompt and embeddings. Does not use any of its internal knowledge base. Commonly Enterprises use OpenAI and Anthropic cloud hosted LLM.

LLM Agents

LLM-powered agents, they can be described as a system that can use an LLM to reason through a problem, create a plan to solve the problem, and execute the plan with the help of a set of tools.

An **Agentic workflow** is any multi-step process that iteratively instructs large language models to complete complex tasks.

AI Agentic Workflow Components



Agent Workflow Design Patterns

- **Reflection**: The LLM examines its own work to come up with ways to improve it
- **Tool Use**: The LLM is given tools such as web search, code execution, or any other function to help it gather information, take action, or process data
- **Planning**: The LLM comes up with, and executes, a multistep plan to achieve a goal (for example, writing an outline for an essay, then doing online research, then writing a draft, and so on)
- **Multi-agent collaboration**: More than one AI agent work together, splitting up tasks and discussing and debating ideas, to come up with better solutions than a single agent would

OpenAI's GPTs and Related APIs

Introduction

- The OpenAI APIs can be applied to virtually any task that involves understanding or generating natural language, code, or images.
- OpenAI offers a spectrum of models with different levels of power suitable for different tasks, as well as the ability to fine-tune your own custom models. These models can be used for everything from content generation to semantic search and classification.
- Using the OpenAI Chat API, you can build your own applications with gpt-3.5-turbo and gpt-4 to do things like:
  - Draft an email or other piece of writing
  - Write Python code
  - Answer questions about a set of documents
  - Create conversational agents
  - Give your software a natural language interface
  - Tutor in a range of subjects
  - Translate languages
  - Simulate characters for video games and much more
  - Take GRE, LSAT, SAT, Pass Uniform Bar Exam, and pass any other test with your iPhone or Android.

Chat Completion

- The completions endpoint is the core of our API and provides a simple interface that is extremely flexible and powerful. You input some text as a prompt, and the API will return a text completion that attempts to match whatever instructions or context you gave it.
  - **Prompt**: Write a tagline for an ice-cream shop.
  - **Completion**: We serve up smiles with every scoop!
- You can think of this as a very advanced autocomplete — the model processes your text prompt and tries to predict what's most likely to come next.

Installation

- To run Python APIs, almost all you need is:

```
$ pip install openai
```

- A moderately small number of other packages are needed: pandas, openai, transformers, plotly, matplotlib, scikit-learn, torch (transformer dep), torchvision, scipy and a few others
- To build Web applications, you can install Node.js library: by running the following command:

```
$ npm install openai
```

There is a large number of community libraries for various languages:

- C# / .NET
  - Betalgo.OpenAI.GPT3 by Betalgo, https://github.com/betalgo/openai
  - OpenAI-API-dotnet by OkGoDoIt, https://github.com/OkGoDoIt/OpenAI-API-dotnet
  - OpenAI-DotNet by RageAgainstThePixel, https://github.com/RageAgainstThePixel/OpenAI-DotNet
- C++
  - liboai by D7EAD, https://github.com/D7EAD/liboai
- Clojure
  - openai-clojure by wkok, https://github.com/wkok/openai-clojure
- Crystal
  - openai-crystal by sferik
- Dart/Flutter
  - openai by anasfik
- Delphi
  - https://github.com/HemulGM

Community Libraries

- Elixir
  - openai.ex by mgallo
- Go
  - go-gpt4 by sashabaranov
- Java
  - openai-java by Theo Kanning
- Julia
  - OpenAI.jl by rory-linehan
- Kotlin
  - openai-kotlin by Mouaad Aallam
- Node.js
  - openai-api by Njerschow
  - openai-api-node by erlapso
  - gpt-x by ceifa
  - gpt3 by poteat
  - gpts by thencc
  - @dalenguyen/openai by dalenguyen
  - tectalic/openai by tectalic
- PHP
  - orhanerday/open-ai by orhanerday
  - tectalic/openai by tectalic
  - openai-php clinet by openai-php
- Python
  - chronology by OthersideAI
- R
  - rgpt4 by ben-aaron188
- Ruby
  - openai by nileshtrivedi
  - ruby-openai by alexrudall
- Rust
  - async-openai by 64bit
  - fieri by lbkolev
- Scala
  - openai-scala-client by cequence-io
- Swift
  - OpenAIKit by dylanshine
- Unity
  - OpenAi-Api-Unity by hexthedev
  - com.openai.unity by RageAgainstThePixel
- Unreal Engine
  - OpenAI-Api-Unreal by KellanM

OpenAI Account

- You need a paying account to access OpenAI resources. Account is free to open, though.
- Go to: https://beta.openai.com/signup
- Once you have an account, go to Personal (in the top right corner), select `View API keys`, hit: Create new key. Give new key a name and save its name and value. The value is a 52? characters long string, that approximately reads like:

```
sk-Jq9kljMnG4XbdrPFIbnAD3BlbkFJQiS2I5dA4tYtagTirQIr
```

- On your system create an environmental variable `OPENAI_API_KEY`.
- On Linux, in your `.bashrc` file enter:

```
OPENAI_API_KEY='sk-Jq9kljMnG4XbdrPFIbnAD3BlbkFJQiS2I5dA4tYtagTirQIr'
export OPEN_API_KEY
```

- When you start any client you will have

```
import os
import openai
# Load your API key from an environment variable or secret management service
openai.api_key = os.getenv("OPENAI_API_KEY")
```

- You are ready to go. You can initiate a Chat

Run a test

```
import os
import openai

# Load your API key from an environment variable or secret management service
openai.api_key = os.getenv("OPENAI_API_KEY")

response = openai.Completion.create(model="text-davinci-003",
    prompt="Say this is a test", temperature=1, max_tokens=7)

print(response)
```

```
{
  "choices": [
    {
      "finish_reason": "length",
      "index": 0,
      "logprobs": null,
      "text": "\n\nThis is indeed a test"
    }
  ],
  "created": 1682266400,
  "id": "cmpl-78WcyD9CRkDMcdwreu2fr4je56lgz",
  "model": "text-davinci-003",
  "object": "text_completion",
  "usage": {
    "completion_tokens": 7,
    "prompt_tokens": 5,
    "total_tokens": 12
  }
}
```

GPT and Related Apps

GPT4

Fun Fact:

Training GPT-4 involved ~25,000 A100 GPUs over ~90-100 days, costing OpenAI nearly $100 million!

A100 GPU's

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
INT8 Tensor Core	624 TOPS | 1248 TOPS*	
GPU Memory	80GB HBM2e	80GB HBM2e
GPU Memory Bandwidth	1,935 GB/s	2,039 GB/s

What About the Carbon foot print ?

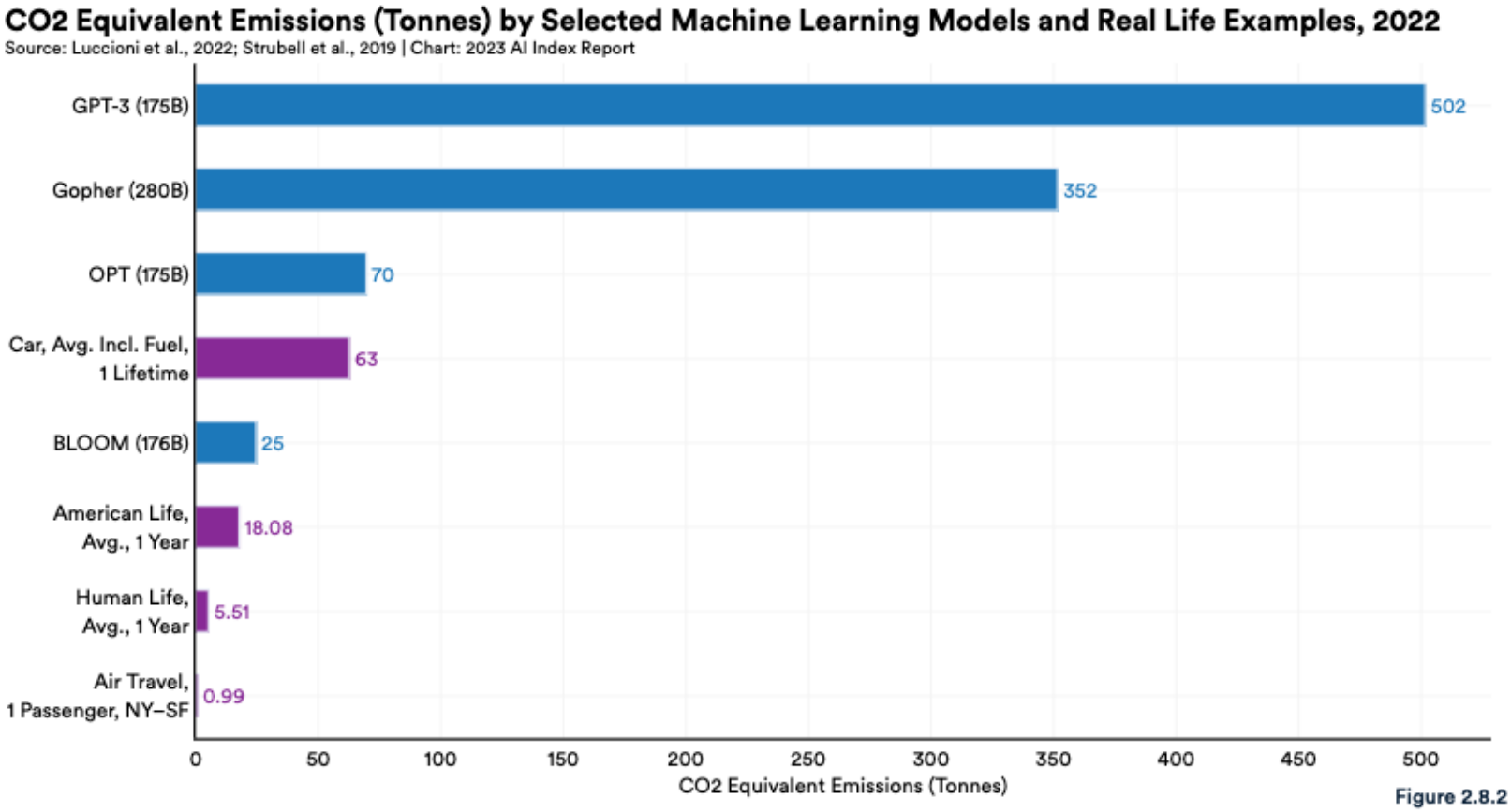
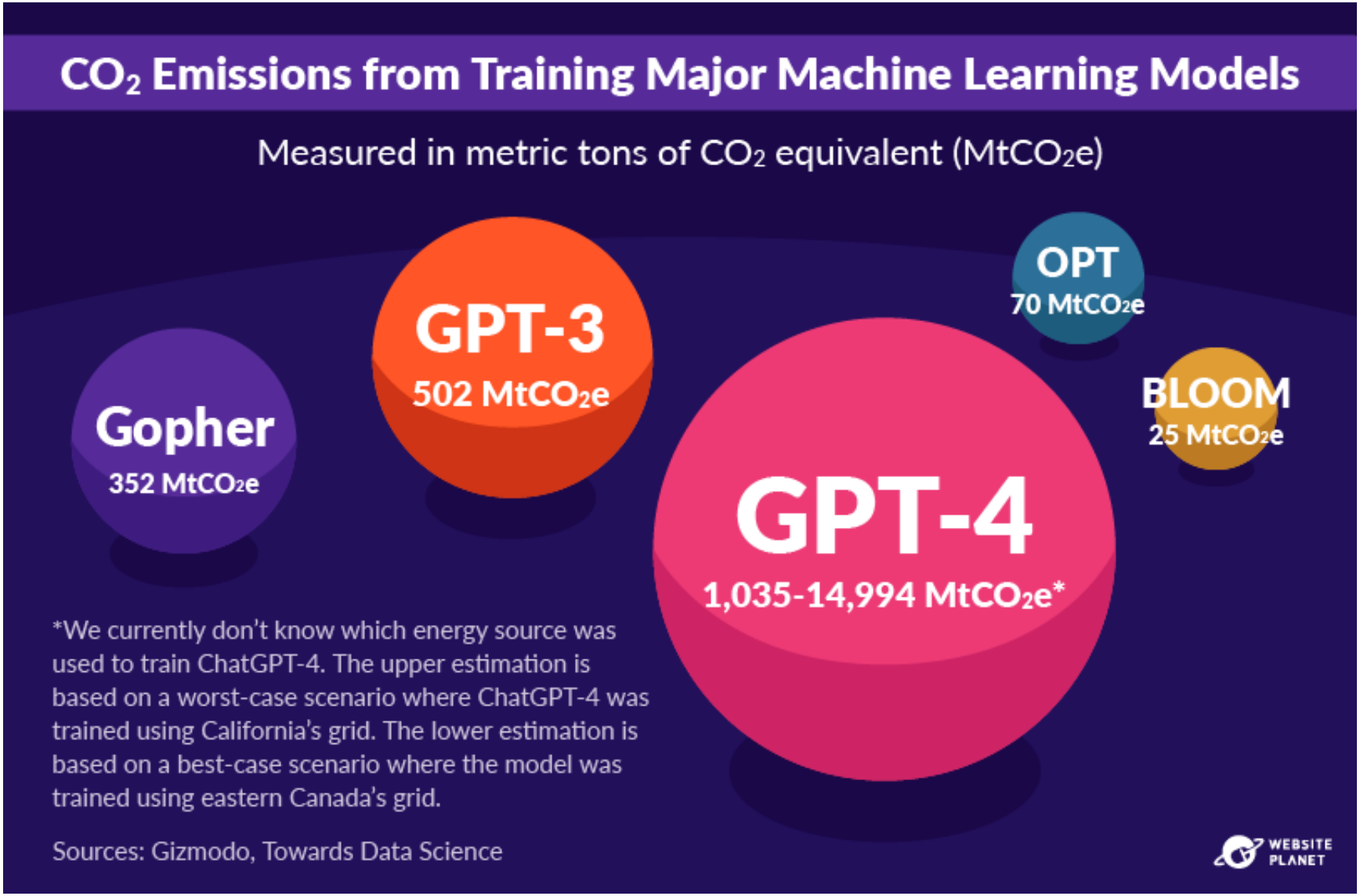


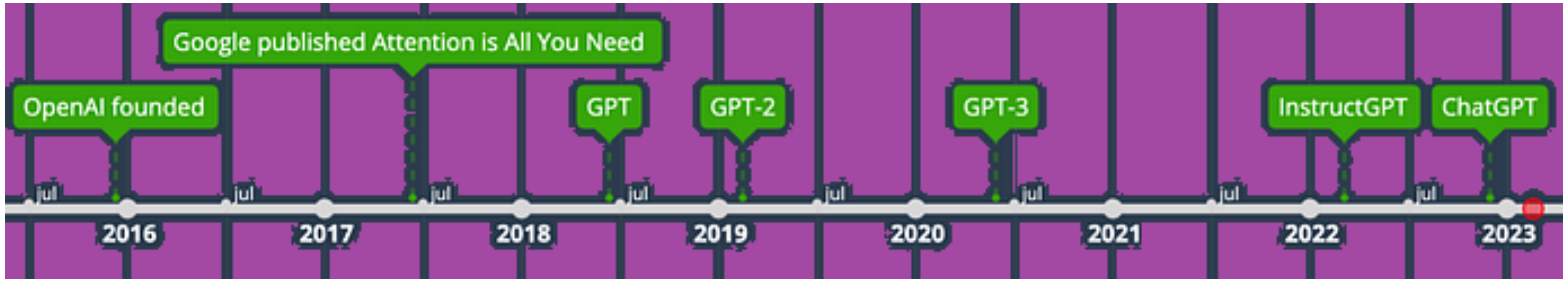
Figure 2.8.2

What About the Carbon foot print ?



Timeline

- 2015. OpenAI founded by Sam Altman, Elon Musk, Greg Brockman, Peter Thiel, and others. OpenAI develops many different AI models other than GPT.
- 2017. Google published the paper *Attention is All You Need* [2], which introduced the transformer architecture. The transformer is a neural network architecture that lays the foundation for many state-of-the-art (SOTA) large language models (LLM) like GPT.
- 2018. GPT is introduced in *Improving Language Understanding by Generative Pre-training* [3]. It is based on a modified transformer architecture and pre-trained on a large corpus.
- 2019. GPT-2 is introduced in *Language Models are Unsupervised Multitask Learners* [4]. GPT-2 could perform a range of tasks without explicit supervision when training.
- 2020. GPT-3 is introduced in *Language Models are Few-Shot Learners* [5], which can perform well with few examples in the prompt without fine-tuning.
- 2022. InstructGPT is introduced in *Training language models to follow instructions with human feedback* [6], which follow user instructions by fine-tuning with human feedback.
- 2022. ChatGPT, a sibling of InstructGPT, is introduced in *ChatGPT: Optimizing Language Models for Dialogue* [7]. It can interact with humans in conversations, thanks to the fine-tuning with human examples and reinforcement learning from human feedback (RLHF)



Other Conceptual Elements of GPT

Many concepts come together in the definition of the GPT models.

Models of the GPT family are **language models** based in the **transformer** architecture, **pre-trained** in a **generative**, **unsupervised** manner that show decent performance in **zero/one/few-shot multitask settings**.

- **Generative models:** In statistics, there are discriminative and generative models, which are often used to perform classification tasks. Discriminative models encode the conditional probability of a given pair of observable and target variables: p(y|x). Generative models encode the joint probability: p(x,y).
- Generative models can "generate new data similar to existing data," which is the key idea to take away. Apart from GPT, other popular examples of generative models are GANs (generative adversarial networks) and VAEs (variational autoencoders).
- **Semi-supervised learning:** This training paradigm combines unsupervised pre-training with supervised fine-tuning. The idea is to train a model with a very large dataset in an unsupervised way, and then adapt(fine-tune) the model to different tasks, by using supervised training in smaller datasets. This paradigm solves two problems: It doesn't need many expensive labeled data and tasks without large datasets can be tackled. It is 'worth mentioning that GPT-2 and GPT-3 are fully unsupervised.
- **Zero shot learning** refers to model's ability to classify object it has not seen before mostly based on similarity of those objects' representations in the embedded space to some other objects.

Other Conceptual Elements of GPT

- **Zero/one/few-shot learning:** Usually, deep learning systems are trained and tested for a specific set of classes. If a computer vision system is trained to classify cat, dog, and horse images, it could be tested only on those three classes. In contrast, in zero-shot learning set up the system is shown at test time — with out weight updating — classes it has *not seen* at training time (for instance, testing the system on elephant images). Same thing for one-shot and few-shot settings, but in these cases, at the test time the system sees one or few examples of the new classes, respectively. Powerful enough system could perform well in these situations, which OpenAI proved with GPT-2 and GPT-3.
- **Multitask learning:** Most deep learning systems are single-task. One popular example is AlphaZero. It can learn a few games like chess or Go, but it can only play *one type* of game at a time. If it knows how to play chess, it doesn't know how to play Go. Multitask systems overcome this limitation. They're trained to solve *different tasks* for a given input. For instance, if I feed the word 'cat' to the system, I could ask it to find the Spanish translation 'gato', I could ask it to show me the image of a cat, or I could ask it to describe its features. Different tasks for the same input.
- **Zero/one/few-shot task transfer:** The idea is to combine the concepts of zero/one/few-shot learning and multitask learning. Instead of showing the system new classes at test time, we could ask it to perform *new tasks* (either showing it zero, one, or few examples of the new task). For instance, let's take a system trained in a huge text corpus. In a one-shot task transfer setting we could write: “I love you -> Te quiero. I hate you -> ____.” We are implicitly asking the system to translate a sentence from English to Spanish (a task it has not been trained on) by showing it a single example (one-shot).

From transformers to GPT, GPT2,GPT3 and GPT4

- From the name, you can see that it is a generative model, good at generating output. GPT is pre-trained, meaning it has learned from a large corpus of text data. GPT is a type of transformer.
- GPT uses only the decoder part of the transformer architecture [3]. The decoders are responsible for predicting the next token in the sequence.
- GPT repeats this process again and again by using the previously generated results as input to generate longer texts , which is called auto-regressive.
- In training the first version of GPT, researchers used **unsupervised pre-training** with the BookCorpus database, consisting of over 7000 unique unpublished books [3].
- In unsupervised learning the model is trying to learn the general rules of language and words.
- On top of the pre-training, GPT uses **supervised fine-tuning** on tasks like summarization or Q/A.
- Supervised means that they will show the model examples of requests and correct answers and ask the model to learn from those examples.
- During the unsupervised pre-training of GPT-2 developer expanded the size of the model to 1.5B parameters and used WebText, a collection of millions of web pages [4]. With such a big corpus to learn from, the model proved that it can perform very well on a wide range of language related-tasks even without supervised fine-tuning.
- In GPT-3, model is expanded to 175 billion parameters and uses a huge corpus comprising hundreds of billions of words from the web, books, and Wikipedia. With such a huge model and a big corpus in pre-training, GPT-3 can learn to perform tasks better with one (one-shot) or a few examples (few-shot) in the prompt without explicit supervised fine-tuning.
- GPT-4, GPT4 Turbo , GPT4-0 are not an Opensource models so the exact number of tokens is not known

What do we need to know about ChatGPT

- ChatGPT by OpenAI was released at the end of 2022 and created a phenomenal interest.
- ChatGPT's capabilities to generate human-like natural language text have inspired experimentation with its potential in large number of fields.
- Some people are worried that ChatGPT is the beginning of the end of us. ChatGPT knows everything better than we do.
- ChatGPT could read our emails, it could listen to our cell phones and will soon know more about ourselves than we do.
- ChatGPT is a Web application that allows you to ask for generated texts or information on particular subject.
- GPT is the engine behind ChatGPT. **GPT** stands for **Generative Pre-trained Transformers.**
- OpenAI's ChatGPT's success put pressure on other tech giants like Google, Amzon and AliBaba to rush releases of their own versions of ChatGPT.

- META on the other hand took a different approach release Llama1,2,3 as Opensource allowing individuals to fine tune their own models

ChatGPT

- ChatGPT is a sibling model to InstructGPT. The training process is similar for ChatGPT and InstructGPT, including the same methods of supervised learning and RLHF. The main difference is that ChatGPT is trained with examples from conversational tasks, like question answering, chit-chat, trivia, etc. [7]. Through this training, ChatGPT can have natural conversations with humans in dialogues. In conversations, ChatGPT can answer follow-up questions and admit mistakes, making it more engaging to interact with.
- ChatGPT is based on a decoder-only auto-regressive transformer model to take in a sequence of text and output a probability distribution of tokens in the sequence, generating one token at a time iteratively.
- Since it doesn't have the ability to search for references in real-time, it is 'making probabilistic predictions in the generation process based on the corpus it has been trained on, which can lead to false claims about facts.
- It's pre-trained on a huge corpus of web and book data and fine-tuned with human conversation examples through supervised learning and reinforcement learning from human feedback (RLHF).
- Its capability is mainly based on its model size and the quality and size of the corpus and examples it learned from. With some additional supervised learning or RLHF, it can perform better in specific contexts or tasks.
- Since the corpus comes from web content and books, they might have biases that the model can learn from, especially for social, cultural, political, or gender-based biases, resulting in biased responses to some requests.

GPT vs. ChatGPT

- ChatGPT is an app; GPT is the brain behind that app
- ChatGPT is a web app (you can access it in your browser) designed specifically for chatbot applications—and optimized for dialogue. It relies on GPT to produce text, like explaining code or writing poems.
- GPT is a language model, not an app. There is an OpenAI playground that lets you play around with GPT, but GPT itself is not an app.
- GPT can be tailored to build different functions, like text summarizing, copy-writing, parsing text, and translating languages. GPT has an open API that lets you tap into GPT-3 or GPT-4 to build AI applications with its functions.
- GPT is behind ChatGPT. It is also the brain behind other tools like Jasper and Writesonic, or Bing's new AI-powered search features.
- With Zapier (http://zapier.com), you can connect OpenAI or ChatGPT to thousands of other apps to automate workflows.
- For example, you can get AI to write a business email from scratch—then automatically save it as a Gmail draft. Or it can score your leads based on their willingness to commit—then notify your sales team.

Models

Models

- The OpenAI API is powered by a diverse set of models with different capabilities and price points. You can also make limited customizations to the original base models for your specific use case with fine-tuning.

Models	Description
GPT-4/GPT-4T/GPT-40	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
GPT-3.5	A set of models that improve on GPT-3 and can understand as well as generate natural language or code
DALL-E	A model that can generate and edit images given a natural language prompt
Whisper	A model that can convert audio into text
Embeddings	A set of models that can convert text into a numerical form
Moderation	A fine-tuned model that can detect whether text may be sensitive or unsafe
GPT-3	A set of models that can understand and generate natural language
Codex deprecated	A set of models that can understand and generate code, including translating natural language to cod

Open Source Models

Model	Description	gitHub URL
Point-E	A System for Generating 3D Point Clouds from Complex Prompts.	https://github.com/openai/point-e
CLIP	Natural language model to predict the most relevant text snippet, given an image	https://github.com/openai/CLIP
Whisper	Whisper is a general-purpose speech recognition model, translation and identification	https://github.com/openai/whisper
Jukebox	A Generative Model for Music	https://github.com/openai/jukebox

- All OpenAI models are non-deterministic, meaning that identical inputs can yield different outputs. Setting temperature to 0 will make the outputs mostly deterministic, but a small amount of variability may remain.

Model Endpoint Compatibility

ENDPOINT	MODEL NAME
/v1/chat/completions	gpt-4, gpt-4-0314, gpt-4-32k, gpt-4-32k-0314, gpt-3.5-turbo, gpt-3.5-turbo-0301
/v1/completions	text-davinci-003, text-davinci-002, text-curie-001, text-babbage-001, text-ada-001
/v1/edits	text-davinci-edit-001, code-davinci-edit-001
/v1/audio/transcriptions	whisper-1
/v1/audio/translations	whisper-1
/v1/fine-tunes	davinci, curie, babbage, ada
/v1/embeddings	text-embedding-ada-002, text-search-ada-doc-001
/v1/moderations	text-moderation-stable, text-moderation-latest

Detailed Capabilities, GPT-40

GPT-4o ("o" for "omni") is most advanced GPT model. It is multimodal (accepting text or image inputs and outputting text), and it has the same high intelligence as GPT-4 Turbo but is much more efficient—it generates text 2x faster and is 50% cheaper. Additionally, GPT-4o has the best vision and performance across non-English languages of any of our models.

MODEL	CONTEXT WINDOW	MAX OUTPUT TOKENS	KNOWLEDGE CUTOFF
gpt-4o	128,000 tokens	16,384 tokens	Oct 2023
gpt-4o-2024-11-20	128,000 tokens	16,384 tokens	Oct 2023
gpt-4o-2024-08-06	128,000 tokens	16,384 tokens	Oct 2023
gpt-4o-2024-05-13	128,000 tokens	4,096 tokens	Oct 2023

Detailed Capabilities, GPT-4

- **GPT-4** is a large multimodal model (accepting text inputs and emitting text outputs today, with image inputs coming in the future) that can solve difficult problems with greater accuracy than any of our previous models, thanks to its broader general knowledge and advanced reasoning capabilities. Like gpt-3.5-turbo, GPT-4 is optimized for chat but works well for traditional completions tasks both using the Chat Completions API.

LATEST MODEL	DESCRIPTION	MAX TOKENS	TRAINING DATA
gpt-4	More capable than any GPT-3.5 model, able to do more complex tasks, and optimized for chat. Will be updated with our latest model iteration.	8,192 tokens	Up to Sep 2021
gpt-4-0314	Snapshot of gpt-4 from March 14th 2023. Unlike gpt-4, this model will not receive updates, and will be deprecated 3 months after a new version is released.	8,192 tokens	Up to Sep 2021
gpt-4-32k	Same capabilities as the base gpt-4 mode but with 4x the context length. Will be updated with our latest model iteration.	32,768 tokens	Up to Sep 2021
gpt-4-32k-0314	Snapshot of gpt-4-32 from March 14th 2023. Unlike gpt-4-32k, this model will not receive updates, and will be deprecated 3 months after a new version is released.	32,768 tokens	Up to Sep 2021

Details, GPT-3.5

- GPT-3.5 models can understand and generate natural language or code. The most capable and cost effective model in the GPT-3.5 family is `gpt-3.5-turbo` which has been optimized for chat but works well for traditional completions tasks as well.

LATEST MODEL	DESCRIPTION	MAX TOKENS	TRAINING DATA
gpt-3.5-turbo	Most capable GPT-3.5 model and optimized for chat at 1/10th the cost of text-davinci-003. Will be updated with our latest model iteration.	4,096 tokens	Up to Sep 2021
gpt-3.5-turbo-0301	Snapshot of gpt-3.5-turbo from March 1st 2023. Unlike gpt-3.5-turbo, this model will not receive updates, and will be deprecated 3 months after a new version is released.	4,096 tokens	Up to Sep 2021
text-davinci-003	Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports inserting completions within text.	4,097 tokens	Up to Jun 2021
text-davinci-002	Similar capabilities to text-davinci-003 but trained with supervised fine-tuning instead of reinforcement learning	4,097 tokens	Up to Jun 2021
code-davinci-002	Optimized for code-completion tasks	8,001 tokens	Up to Jun 2021

Other Models

- **DALL·E** is a AI system that can create realistic images and art from a description in natural language. OpenAI currently supports the ability, given a prompt, to create a new image with a certain size, edit an existing image, or create variations of a user provided image.
- The current DALL·E model available through OpenAI API is the 2nd iteration of DALL·E with more realistic, accurate, and 4x greater resolution images than the original model. You can try it through the Labs interface or via the API.
- **Whisper** is a general-purpose speech recognition model. It is trained on a large dataset of diverse audio and is also a multi-task model that can perform multilingual speech recognition as well as speech translation and language identification. The Whisper v2-large model is currently available through OpenAI API with under model name: whisper-1.
- Currently, there is no difference between the open source version of Whisper and the version available through the API. However, through API, OpenAI offers an optimized inference process which makes running Whisper through the API much faster than doing it through other means.
- **Embeddings** are a numerical representation of text that can be used to measure how related are two pieces of text. The second generation embedding model, text-embedding-ada-002 is designed to replace the previous embedding models at a fraction of the cost. Embeddings are useful for search, clustering, recommendations, anomaly detection, and classification tasks.
- **Codex** models are now deprecated. They were descendants of the GPT-3 models that would understand and generate code. Their training data contains both natural language and billions of lines of public code from GitHub

Pricing

- OpenAI offers a spectrum of models with different capabilities and price points. Models: text-davinci-003 or gpt-3.5-turbo should be used while experimenting since they yield the best results. Once you have things working, you can see if the other models can produce the same results with lower latency and costs. Or if you might need to move to a more powerful model like gpt-4.
- The total number of tokens processed in a single request (both prompt and completion) can not exceed the model's maximum context length. For most models, this is 4,096 tokens or about 3,000 words. As a rough rule of thumb, 1 token is approximately 4 characters or 0.75 words for English text.
- Pricing is pay-as-you-go per 1,000 tokens, with $5 in free credit that can be used during the first 3 months.

Prices per use of different models

- GPT40
  - $2.50 / 1M input tokens
  - $10 / 1M output tokens
- GPT40-mini
  - $0.150 / 1M input tokens
  - $0.600 / 1M output tokens
- GPT-4
  - prompt: $0.03/1K tokens
  - completion: $0.06/1K tokens
- Chat
  - gpt-3.5-turbo: $0.002/1K tokens
- InstructGPT
  - Ada (Fastest): $0.0004/1K token
  - Babbage: $0.0005/1K token
  - Curie: $0.0020/1K token
  - Davinci (the most powerful): $0.0200/1K token
- Fine-tuning models:
  - Ada $0.0004/1K tokens
  - Babbage $0.0006/1K tokens s
  - Curie $0.0030/1K tokens
  - Davinci $0.0300/1K tokens
- Embedding models
  - Ada $0.0004/1K tokens
- Image models
  - Resolution Price
  - 1024×1024 $0.020/image
  - 512×512 $0.018/image
  - 256×256 $0.016/image
- Audio models
- Whisper $0.006/minute (rounded to the nearest second)

Book with 500 pages and 90,000 words
Will contain ~120,000 tokens

Count your tokens
https://gptforwork.com/tools/tokenizer

Making Requests

RESTful Requests

- You can make your requests by submitting an HTTP POST to one of End Points listed on slide 47.
- For the Chat Completion, we could try:

```
$ curl https://api.openai.com/v1/chat/completions \
   -H "Content-Type: application/json" \
   -H "Authorization: Bearer sk-Jq9kljMnG4XbdrPFIbnAD3BlbkFJQiS2I5dA4tYtagTirQIr" \
   -d '{
      "model": "gpt-3.5-turbo",
      "messages": [{"role": "user", "content": "Say this is a test!"}],
      "temperature": 0.7
}'
```

HTTP Response

- The response would read:


```
{
 "id":"chatcmpl-78efw8hImQDOkMCrIbGtrfT6KmMOD",
 "object":"chat.completion",
 "created":1682297336,
 "model":"gpt-3.5-turbo-0301",
 "usage":{
          "prompt_tokens":14,
          "completion_tokens":5,
          "total_tokens":19
 },
 "choices":[
    {
       "message":{
          "role":"assistant",
          "content":"This is a test!"
    },
     "finish_reason":"stop",
     "index":0
   }
 ]
}
```

Newer vs. Older

- The ChatGPT and GPT-4 models are language models that are optimized for conversational interfaces. The models behave differently than the older GPT-3 models.
- Previous models were text-in and text-out, meaning they accepted a prompt string and returned a completion to append to the prompt.
- ChatGPT and GPT-4 models are conversation-in and message-out. The models expect input formatted in a specific chat-like transcript format, and return a completion that represents a model-written message in the chat.
- While this format was designed specifically for multi-turn conversations, you'll find it can also work well for non-chat scenarios too.
- In OpenAI there are two different options for interacting with these models:
  - **Chat Completion API**.
  - **Completion API with Chat** Markup Language (ChatML).
- The Chat Completion API is a dedicated API for interacting with the ChatGPT and GPT-4 models. This API is the preferred method for accessing these models. **It is also the only way to access the new GPT-4 models**.
- ChatML uses the same completion API that you use for other models like text-davinci-002, it requires a unique token based prompt format known as Chat Markup Language (ChatML). This provides lower level access than the dedicated Chat Completion API, but also requires additional input validation, only supports ChatGPT (gpt-35-turbo) models, and **the underlying format is more likely to change over time**.
- This text is produced by OpenAi's department for confused terminology.

Completions vs. Chats, from API references

Completions

- Given a prompt, the model will return one or more predicted completions, and can also return the probabilities of alternative tokens at each position.

Create completion

- POST https://api.openai.com/v1/completions
- Creates a completion for the provided prompt and parameters.

Request body

- model string Required ID of the model to use.
- **Prompt** string or array Optional Defaults to <|endoftext|>
- The prompt(s) to generate completions for, encoded as a string, array of strings, array of tokens, or array of token arrays.

Chat

- Given a list of messages describing a conversation, the model will return a response

Create chat completion Beta

- POST https://api.openai.com/v1/chat/completions
- Creates a model response for the given chat conversation.

Request body

- model string Required ID of the model to use.
- **Messages** array Required
- A list of messages describing the conversation so far.
- **Role** string Required
- The role of the author of this message. One of system, user, or assistant.
- **Content** string Required

Chat

- Chat models take a series of messages as input, and return a model-generated message as output.
- Although the chat format is designed to make multi-turn conversations easy, it is just as useful for single-turn tasks without any conversations (such as those previously served by instruction following models like text-davinci-003).
- Crafting good instructions is important for getting good results, but sometimes they aren't enough.

- In many cases, it is 'helpful to both show and tell the model what you want.
- Adding examples to your prompt can help communicate patterns or nuances.
- Try submitting a prompt which includes a couple examples. For example, if you want a nice name for horse that is a superhero, you should suggest two or three such names
- The completions endpoint is the core of OpenAI API and provides a simple interface that is extremely flexible and powerful. You input some text as a prompt, and the API will return a text completion that attempts to match whatever instructions or context you gave it.

Chat

- Given a prompt, the model will return one or more predicted completions, and can also return the probabilities of alternative tokens at each position.
- An API call looks as follows:

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

- The main input is the `messages` parameter. Messages must be an array of message objects, where each object has a `role` (either "`system`", "`user`", or "`assistant`") and `content` (the content of the message). Conversations can be as short as 1 message or fill many pages.
- Typically, a conversation is formatted with a `system` message first, followed by alternating `user` and a`ssistant` messages.
- The `system` message helps set the behavior of the `assistant`. In the example above, the assistant was instructed with "`You are a helpful assistant.`"

Chat, response

```
print(response)
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "The World Series in 2020 was played at Globe Life Field, a baseball
park in Arlington, Texas.",
        "role": "assistant"
      }
    }
  ],
  "created": 1682266647,
  "id": "chatcmpl-78WgxYFAWeYCgpFR10Y5vJkCKr0VV",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 22,
    "prompt_tokens": 57,
    "total_tokens": 79
  }
}
```

Again

- The following code snippet shows the most basic way to use the ChatGPT and GPT-4 models with the Chat Completion API.

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

response = openai.ChatCompletion.create(
    engine="gpt-3.5-turbo", # The name you choose for ChatGPT or GPT-4 model.
    messages=[
        {"role": "system", "content": "Assistant is a large language model trained by
OpenAI."},
        {"role": "user", "content": "What's the difference between garbanzo beans and
chickpeas?"}
    ]
)

print(response)

print(response['choices'][0]['message']['content'])
```

Responses


```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "There is no difference between garbanzo beans and chickpeas \u2014 the terms are used interchangeably to describe the same round, beige-colored legume. Garbanzo beans are the Spanish name for the legume, while chickpeas are the English name for the same legume. They are a common ingredient in many traditional Middle Eastern and Mediterranean dishes, such as hummus, falafel, and stews.",
        "role": "assistant"
      }
    }
  ],
  "created": 1679014551,
  "id": "chatcmpl-6usfn2yyjkbmESe3G4jaQR6bsScO1",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 86,
    "prompt_tokens": 37,
    "total_tokens": 123
  }
}
```


There is no difference between garbanzo beans and chickpeas — the terms are used interchangeably to describe the same round, beige-colored legume. Garbanzo beans are the Spanish name for the legume, while chickpeas are the English name for the same legume. They are a common ingredient in many traditional Middle Eastern and Mediterranean dishes, such as hummus, falafel, and stews.

finish_reason

- Every response includes a **`finish_reason`**. The possible values for finish_reason are:

  - **`stop:`** API returned complete model output.
  - **`length`**: Incomplete model output due to max_tokens parameter or token limit.
  - **`content_filter`**: Omitted content due to a flag from our content filters.
  - **`null`**:API response still in progress or incomplete.

- Consider setting m**`ax_tokens`** to a slightly higher value than normal such as 300 or 500. This ensures that the model doesn't stop generating text before it reaches the end of the message.

Working with the Chat Completion API

- OpenAI trained the ChatGPT and GPT-4 models to accept input formatted as a conversation. The messages parameter takes an array of dictionaries with a conversation organized by role.
- The format of a basic Chat Completion is as follows:

```
{"role": "system", "content": "Provide some context and/or instructions to the model"},
{"role": "user", "content": "The users messages goes here"}
```

- A conversation with one example answer followed by a question would look like:

```
{"role": "system", "content": "Provide some context and/or instructions to the model."},
{"role": "user", "content": "Example question goes here."},
{"role": "assistant", "content": "Example answer goes here."},
{"role": "user", "content": "First question/message for the model to actually respond to."}
```

System role

- The **system role** also known as the **system message** is included at the beginning of the array. This message provides the initial instructions to the model. You can provide various information in the system role including:
  - A brief description of the assistant
  - Personality traits of the assistant
  - Instructions or rules you would like the assistant to follow
  - Data or information needed for the model, such as relevant questions from an FAQ
- You can customize the system role for your use case or just include basic instructions. The system role/message is optional, but it is 'recommended to at least include a basic one to get the best results.

Messages

- After the system role, you can include a series of messages between the user and the assistant.

```
{"role": "user", "content": "What is thermodynamics?"}
```

- To trigger a response from the model, **you should end with a user message** indicating that it is 'the assistant's turn to respond. You can also include a series of example messages between the user and the assistant as a way to do few shot learning.

Message prompt examples

- The following are examples of different styles of prompts that you could use with the ChatGPT and GPT-4 models. You can experiment with different prompts to customize the behavior for your own use cases.

Basic example

- If you want the ChatGPT model to behave similarly to chat.openai.com, you can use a basic system message like "Assistant is a large language model trained by OpenAI."

```
{"role": "system", "content": "Assistant is a large language model trained by OpenAI."},
{"role": "user", "content": "What's the difference between garbanzo beans and chickpeas?"}
```

Example with instructions

- For some scenarios, you may want to give additional instructions to the model to define guardrails for what the model is able to do.

```
{"role": "system", "content": "Assistant is an intelligent chatbot designed to help users answer their tax related questions.
Instructions:
- Only answer questions related to taxes.
- If you're unsure of an answer, you can say "I do not know" or "I'm not sure" and recommend users go to the IRS website for more information. "},
{"role": "user", "content": "When are my taxes due?"}
```

Using data for grounding

- You can also include relevant data or information in the system message to give the model extra context for the conversation. If you only need to include a small amount of information, you can hard code it in the system message. If you have a large amount of data that the model should be aware of, you can use embeddings to retrieve the most relevant information at query time.

```
{"role": "system", "content": "Assistant is an intelligent chatbot designed to help users answer technical questions about Azure OpenAI Serivce. Only answer questions using the context below and if you're not sure of an answer, you can say 'I do not know'.

Context:

- Azure OpenAI Service provides REST API access to OpenAI's powerful language models including the GPT-3, Codex and Embeddings model series.

- Azure OpenAI Service gives customers advanced language AI with OpenAI GPT-3, Codex, and DALL-E models with the security and enterprise promise of Azure. Azure OpenAI co-develops the APIs with OpenAI, ensuring compatibility and a smooth transition from one to the other.

- At Microsoft, we're committed to the advancement of AI driven by principles that put people first. Microsoft has made significant investments to help guard against abuse and unintended harm, which includes requiring applicants to show well-defined use cases, incorporating Microsoft's principles for responsible AI use."

},

{"role": "user", "content": "What is Azure OpenAI Service?"}
```

Few shot learning with Chat Completion

- You can give few shot examples to the model. The approach for few shot learning has changed slightly because of the new prompt format. You can now include a series of messages between the user and the assistant in the prompt as few shot examples. These examples can be used to seed answers to common questions to prime the model or teach particular behaviors to the model.

- This is only one example of how you can use few shot learning with ChatGPT and GPT-4. You can experiment with different approaches to see what works best for your use case.

```
{"role": "system", "content": "Assistant is an intelligent chatbot designed to help users answer their tax related questions. "},
{"role": "user", "content": "When do I need to file my taxes by?"},
{"role": "assistant", "content": "In 2023, you will need to file your taxes by April 18th. The date falls after the usual April 15th deadline because April 15th falls on a Saturday in 2023. For more details, see https://www.irs.gov/filing/individuals/when-to-file."},
{"role": "user", "content": "How can I check the status of my tax refund?"},
{"role": "assistant", "content": "You can check the status of your tax refund by visiting https://www.irs.gov/refunds"}
```

Chat Completion for non-chat scenarios

- The Chat Completion API is designed to work with multi-turn conversations, but it also works well for non-chat scenarios.
- For example, for an entity extraction scenario, you might use the following prompt:

```
{"role": "system", "content": "You are an assistant designed to extract
entities from text. Users will paste in a string of text and you will
respond with entities you've extracted from the text as a JSON object.
Here's an example of your output format:
{
   "name": "",
   "company": "",
   "phone_number": ""
}"},
{"role": "user", "content": "Hello. My name is Robert Smith. I'm calling
from Contoso Insurance, Delaware. My colleague mentioned that you are
interested in learning about our comprehensive benefits policy. Could you
give me a call back at (555) 346-9322 when you get a chance so we can go
over the benefits?"}
```

Creating a basic conversation loop

- This example shows you how to create a conversation loop that performs the following actions:
  - Continuously takes console input, and properly formats it as part of the messages array as user role content.
  - Outputs responses that are printed to the console and formatted and added to the messages array as assistant role content.
- This means that every time a new question is asked, a running transcript of the conversation so far is sent along with the latest question. Since the model has no memory, you need to send an updated transcript with each new question or the model will lose context of the previous questions and answers.

Conversation Loop

```
import os
import openai
openai.api_key = os.getenv("OPENAI_API_KEY")
conversation=[{"role": "system", "content": "You are a helpful
assistant."}]

while(True):
    user_input = input()
    conversation.append({"role": "user", "content": user_input})

    response = openai.ChatCompletion.create(
        engine="gpt-3.5-turbo", # The deployment name you chose when you
deployed the ChatGPT or GPT-4 model.
        messages = conversation
    )

    conversation.append({"role": "assistant", "content":
response['choices'][0]['message']['content']})
    print("\n" + response['choices'][0]['message']['content'] + "\n")
```

- When you run the code above you will get a blank console window. Enter your first question in the window and then hit enter. Once the response is returned, you can repeat the process and keep asking questions.

Managing Conversations

- The previous example will run until you hit the model's token limit. With each question asked, and answer received, the messages array grows in size. The token limit for gpt-35-turbo is 4096 tokens, whereas the token limits for gpt-4 and gpt-4-32k are 8192 and 32768 respectively. These limits include the token count from both the message array sent and the model response. The number of tokens in the messages array combined with the value of the `max_tokens` parameter must stay under these limits or you'll receive an error.
- It's your responsibility to ensure the prompt and completion falls within the token limit. This means that for longer conversations, you need to keep track of the token count and only send the model a prompt that falls within the limit.
- The following code sample shows a simple chat loop example with a technique for handling a 4096 token count using OpenAI's `tiktoken` library.
- The code requires `tiktoken` 0.3.0. If you have an older version run:

```
$ pip install tiktoken --upgrade
```

- In the following example, once the token count is reached, the oldest messages in the conversation transcript will be removed. `del` is used instead of `pop()` for efficiency, and we start at index 1 so as to always preserve the system message and only remove `user/assistant` messages. Over time, this method of managing the conversation can cause the conversation quality to degrade as the model will gradually lose context of the earlier portions of the conversation.
- An alternative approach is to limit the conversation duration to the max token length or a certain number of turns. Once the max token limit is reached and the model would lose context if you were to allow the conversation to continue, you can prompt the user that they need to begin a new conversation and clear the messages array to start a brand new conversation with the full token limit available.

Managing Conversations

```
import tiktoken
import openai
import os
openai.api_key = os.getenv("OPENAI_API_KEY")
system_message = {"role": "system", "content": "You are a helpful assistant."}
max_response_tokens = 250
token_limit= 4096
conversation=[]
conversation.append(system_message)
def num_tokens_from_messages(messages, model="gpt-3.5-turbo-0301"):
    encoding = tiktoken.encoding_for_model(model)
    num_tokens = 0
    for message in messages:
        num_tokens += 4  # every message follows
<im_start>{role/name}\n{content}<im_end>\n
        for key, value in message.items():
            num_tokens += len(encoding.encode(value))
            if key == "name":  # if there's a name, the role is omitted
                num_tokens += -1  # role is always required and always 1 token
    num_tokens += 2  # every reply is primed with <im_start>assistant
    return num_tokens
```

Managing Conversations

```
while(True):
    user_input = input("")
    conversation.append({"role": "user", "content": user_input})
    conv_history_tokens = num_tokens_from_messages(conversation)

    while (conv_history_tokens+max_response_tokens >= token_limit):
        del conversation[1]
        conv_history_tokens = num_tokens_from_messages(conversation)

    response = openai.ChatCompletion.create(
        engine="gpt-3.5-turbo", # The deployment name for ChatGPT or GPT-4 model.
        messages = conversation,
        temperature=.7,
        max_tokens=max_response_tokens,
    )
    conversation.append({"role": "assistant", "content":
response['choices'][0]['message']['content']})
    print("\n" + response['choices'][0]['message']['content'] + "\n")
```

Prompt Design

- OpenAI's models can do everything from generating original stories to performing complex text analysis. Because they can do so many things, you have to be explicit in showing what you want. **Showing, not just telling, is often the secret to a good prompt.**
- The models try to predict what you want from the prompt. If you send the words "Give me a list of cat breeds," the model would not automatically assume that you are asking for a list of cat breeds. You could as easily be asking the model to continue a conversation where the first words are "Give me a list of cat breeds" and the next ones are "and I'll tell you which ones I like." If the model only assumed that you wanted a list of cats, it would not be as good at content creation, classification, or other tasks.

There are three basic guidelines to creating prompts:

- **Show and tell.** Make it clear what you want either through instructions, examples, or a combination of the two. If you want the model to rank a list of items in alphabetical order or to classify a paragraph by sentiment, show it that is what you want.
- **Provide quality data.** If you're trying to build a classifier or get the model to follow a pattern, make sure that there are enough examples. Be sure to proofread your examples — the model is usually smart enough to see through basic spelling mistakes and give you a response, but it also might assume that the mistakes are intentional and it can affect the response.
- **Check your settings**. The temperature and top_p settings control how deterministic the model is in generating a response. If you're asking it for a response where there's only one right answer, then you'd want to set these settings to lower values. If you're looking for a response that is not obvious, then you might want to set them to higher values. The number one mistake people use with these settings is assuming that they're "cleverness" or "creativity" controls.

Troubleshooting

- If you're having trouble getting the API to perform as expected, follow this checklist:
  1. Is it clear what the intended generation should be?
  2. Are there enough examples?
  3. Did you check your examples for mistakes? (The API won't tell you directly)
  4. Are you using temp and top_p correctly?

Fine-tuning

Fine Tuning

Fine-tuning very often means instruction fine-tuning.

An instruction dataset, comprising pairs of instructions, answers, and sometimes context, is required for such fine-tuning.

Fine Tuning changes models **Parameters** , In-Context learning **does not** change the Model weights

Fine-tuning

- Fine-tuning lets you get more out of the models available through the API by providing:
  - Higher quality results than prompt design
  - Ability to train on more examples than can fit in a prompt
  - Token savings due to shorter prompts
  - Lower latency requests
- GPT-4 has been pre-trained on a vast amount of text from the open internet. When given a prompt with just a few examples, it can often intuit what task you are trying to perform and generate a plausible completion. This is often called "few-shot learning."
- Fine-tuning improves on few-shot learning by training on many more examples than can fit in the prompt, letting you achieve better results on a wide number of tasks. Once a model has been fine-tuned, you won't need to provide examples in the prompt anymore. This saves costs and enables lower-latency requests.
- At a high level, fine-tuning involves the following steps:
  - Prepare and upload training data
  - Train a new fine-tuned model
  - Use your fine-tuned model
- Fine-tuning has it own pricing:
  - Ada $0.0004/1K tokens
  - Babbage $0.0006/1K tokens s
  - Curie $0.0030/1K tokens
  - Davinci $0.0300/1K tokens

Fine Tuning PEFT

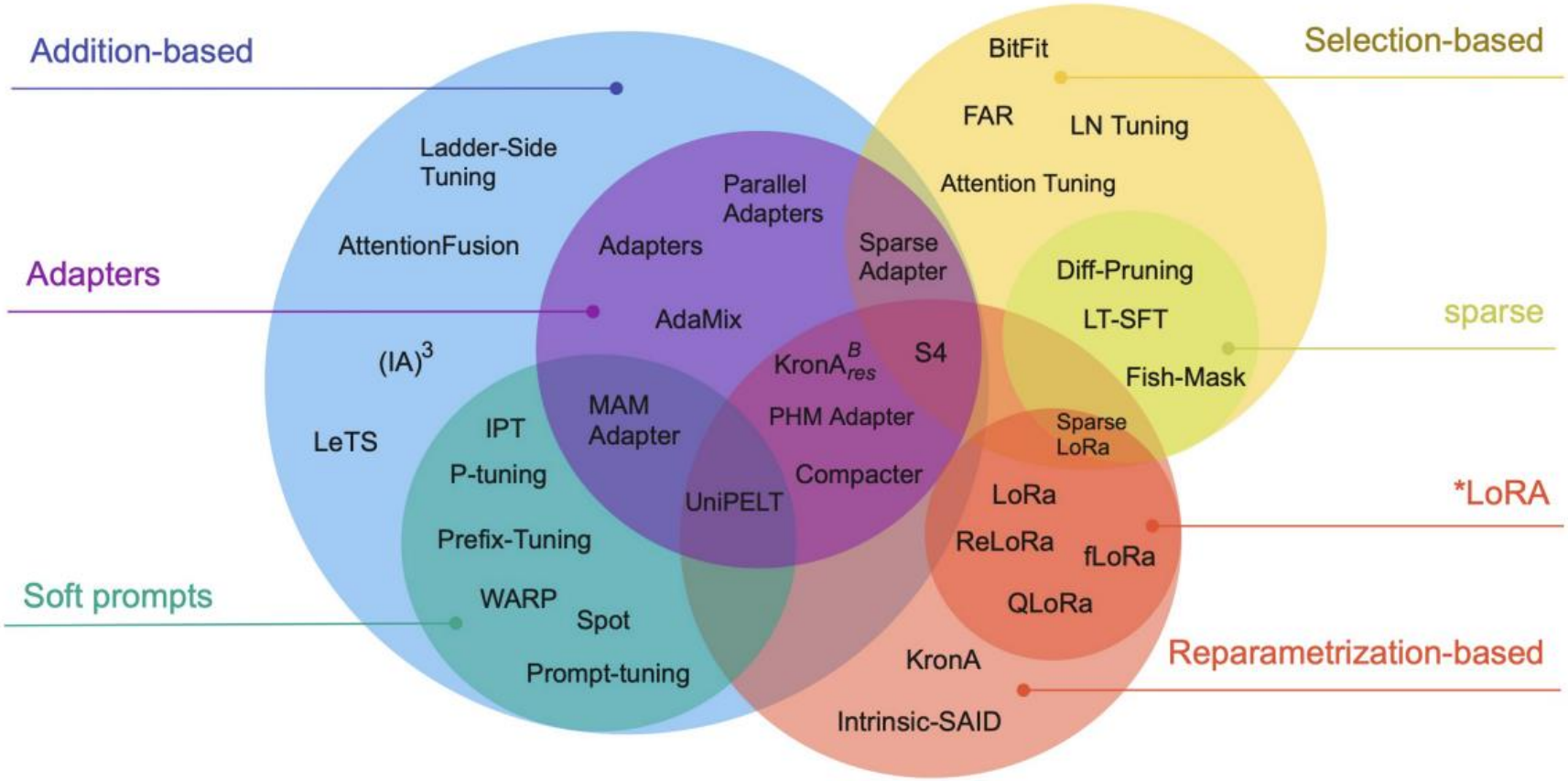
PEFT stands for Parameter Efficient Finetuning.

Unlike full parameter finetuning, PEFT preserves the vast majority of the model's original weights. Think Transfer learning analogy from CNNs.

There are majorly three methods to do PEFT.

1. Additive
2. Selective
3. Reparameterization

PEFT methods



https://arxiv.org/pdf/2303.15647

Models that can be fine-tuned

- Fine-tuning is currently only available for the following base models:
  - **davinci,**
  - **curie,**
  - **babbage**, and
  - **ada.**
- These are the original models that do not have any instruction following training (like text-davinci-003 does for example). You are also able to continue fine-tuning a fine-tuned model to add additional data without having to start from scratch.
- For fine-tuning, it is recommended to use OpenAI Command-line Interface (CLI). Install it, run:

```
$ pip install --upgrade openai
```

- Verify on Windows:

```
(base) D:\..> where openai
C:\ProgramData\Anaconda3\Scripts\openai.exe
```

- Verify on Linux:

```
$ which openai
/home/ubuntu/.local/bin/openai
If you have not done that already, set OPENAI_API_KEY
$ OPENAI_API_KEY=sk-arhTry674….
$ export OPENAI_API_KEY
```

JSON Lines Text Format

- **JSON Lines text format** is also called newline-delimited JSON. JSON Lines is a convenient format for storing structured data that may be processed one record at a time. It works well with unix-style text processing tools and shell pipelines. it is 'a great format for log files. it is 'also a flexible format for passing messages between cooperating processes.
- The JSON Lines format has three requirements:

1. UTF-8 Encoding

- JSON allows encoding Unicode strings with only ASCII escape sequences, however those escapes will be hard to read when viewed in a text editor. The author of the JSON Lines file may choose to escape characters to work with plain ASCII files.
- Encodings other than UTF-8 are very unlikely to be valid when decoded as UTF-8 so the chance of accidentally misinterpreting characters in JSON Lines files is low.

2. Each Line is a Valid JSON Value

- The most common values will be objects or arrays, but any JSON value is permitted.
- See json.org for more information about JSON values.

3. Line Separator is '\n'

- This means '\r\n' is also supported because surrounding white space is implicitly ignored when parsing JSON values.
- The last character in the file may be a line separator

Prepare training data

- Training data is how you teach GPT-3 what you would like it to say.
- Your data must be a JSONL document, where each line is a prompt-completion pair corresponding to a training example. You can use OpenAI CLI data preparation tool to easily convert your data into this JSONL format.

```
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
{"prompt": "<prompt text>", "completion": "<ideal generated text>"}
```

- Designing your prompts and completions for fine-tuning is different from designing your prompts for use with our base models (Davinci, Curie, Babbage, Ada).
- In particular, while prompts for base models often consist of multiple examples ("few-shot learning"), for fine-tuning, each training example generally consists of a single input example and its associated output, without the need to give detailed instructions or include multiple examples in the same prompt.
- The more training examples you have, the better. We recommend having at least a couple hundred examples. In general, we've found that each doubling of the dataset size leads to a linear increase in model quality.

CLI data preparation tool

- OpenAI developed a tool which validates, gives suggestions and reformats your data:

```
openai tools fine_tunes.prepare_data -f <LOCAL_FILE>
```

- This tool accepts different formats, with the only requirement that they contain a prompt and a completion column/key. You can pass a **CSV, TSV, XLSX, JSON** or **JSONL** file, and it will save the output into a JSONL file ready for fine-tuning, after guiding you through the process of suggested changes.
- To fine-tune a model, you'll need a set of training examples that each consist of a single input ("prompt") and its associated output ("completion"). This is notably different from using our base models, where you might input detailed instructions or multiple examples in a single prompt.
- Each prompt should end with a fixed separator to inform the model when the prompt ends and the completion begins. A simple separator which generally works well is \n\n###\n\n. The separator should not appear elsewhere in any prompt.
- Each completion should start with a whitespace due to our tokenization, which tokenizes most words with a preceding whitespace.
- Each completion should end with a fixed stop sequence to inform the model when the completion ends. A stop sequence could be \n, ###, or any other token that does not appear in any completion.
- For inference, you should format your prompts in the same way as you did when creating the training dataset, including the same separator. Also specify the same stop sequence to properly truncate the completion.

Create a fine-tuned model

- When the job is done, it should display the name of the fine-tuned model.
- In addition to creating a fine-tune job, you can also list existing jobs, retrieve the status of a job, or cancel a job.

```
# List all created fine-tunes
openai api fine_tunes.list

# Retrieve the state of a fine-tune. The resulting object includes
# job status (which can be one of pending, running, succeeded, or failed)
# and other information
openai api fine_tunes.get -i <YOUR_FINE_TUNE_JOB_ID>

# Cancel a job
openai api fine_tunes.cancel -i <YOUR_FINE_TUNE_JOB_ID>
```

Use a fine-tuned model

- When a job has succeeded, the `fine_tuned_model` field will be populated with the name of the model. You may now specify this model as a parameter to OpenAI Completions API, and make requests to it use it.
- After your job first completes, it may take several minutes for your model to become ready to handle requests. If completion requests to your model time out, it is likely because your model is still being loaded. If this happens, try again in a few minutes.
- You can start making requests by passing the model name as the model parameter of a completion request:

OpenAI CLI:

```
$ openai api completions.create -m <FINE_TUNED_MODEL> -p <YOUR_PROMPT>
```

cURL

```
$ curl https://api.openai.com/v1/completions \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"prompt": YOUR_PROMPT, "model": FINE_TUNED_MODEL}'
```

Python:

```
import openai
openai.Completion.create(
    model=FINE_TUNED_MODEL,
    prompt=YOUR_PROMPT)
```

- You may continue to use all the other Completions parameters like `temperature`, `frequency_penalty`, `presence_penalty`, etc, on these requests to fine-tuned models.

Delete a fine-tuned model

- To delete a fine-tuned model, you must be designated an "owner" within your organization.

OpenAI CLI:

```
openai api models.delete -i <FINE_TUNED_MODEL>
```

cURL

```
curl -X "DELETE" https://api.openai.com/v1/models/<FINE_TUNED_MODEL> \
  -H "Authorization: Bearer $OPENAI_API_KEY"
```

Python

```
import openai
openai.Model.delete(FINE_TUNED_MODEL)
```

Instead of Summary

- Many articles in press and on the Web claim that ChatGPT (and GPT) will eliminate all middle-class jobs. All we do anyway is fill-in PPTs and MS Word document that are not really meant to be read carefully. ChatGPT, i.e., GPT could do that at a fraction of human cost.
- Many other articles claim that “Prompt Programming” is the main IT skill of the future. You need to know how to formulate your prompts and messages to get “good” answers.
- Another issue is what happens with all those companies who live by providing NLP software for many niche markets and many industry dependent use case. Those companies will adopt.
- Companies will use fine-tuning mechanism to create specialized models.

Appendix

Library

- **Transformers** is an open-source Python library for building models using the Transformer architecture, which is developed and maintained by Hugging Face.
- **DeepSpeed** is a PyTorch-based deep learning optimization library developed by Microsoft, which has been used to train a number of LLMs, such as GPT-Neo and BLOOM [66]. It provides various optimization techniques for distributed training, such as memory optimization (ZeRO technique), gradient checkpointing, and pipeline parallelism. Additionally, it provides the API for fine-tuning and evaluating these models.
- **Megatron-LM** is a PyTorch-based deep learning library developed by NVIDIA for training large-scale language models. It also provides rich optimization techniques for distributed training, including model and data parallelism, mixed-precision training, FlashAttention, and gradient checkpointing.
- **JAX** is a Python library for high-performance machine learning developed by Google Brain, allowing users to easily perform computations on arrays with hardware acceleration (GPU or TPU) support
- **Colossal-AI** is a deep learning library developed by EleutherAI for training large-scale language models. It is built on top of JAX and supports optimization strategies for training such as mixed-precision training and parallelism. Recently, a ChatGPT-like model called ColossalChat
- **BMTrain** is an efficient library developed by OpenBMB for training models with large-scale parameters in a distributed manner, which emphasizes code simplicity, low resource, and high availability. BMTrain has already incorporated several common LLMs (e.g., Flan-T5 and GLM [80]) into its ModelCenter, where developers can use these models directly
- **FastMoE** is a specialized training library for MoE (i.e., mixture-of-experts) models. It is developed on top of PyTorch, prioritizing both efficiency and user-friendliness in its design.
- Existing deep learning frameworks **PyTorch**, **TensorFlow**, **MXNet** , **PaddlePaddle**, **MindSpore**, and **OneFlow**) have also provided support for training large-scale models.