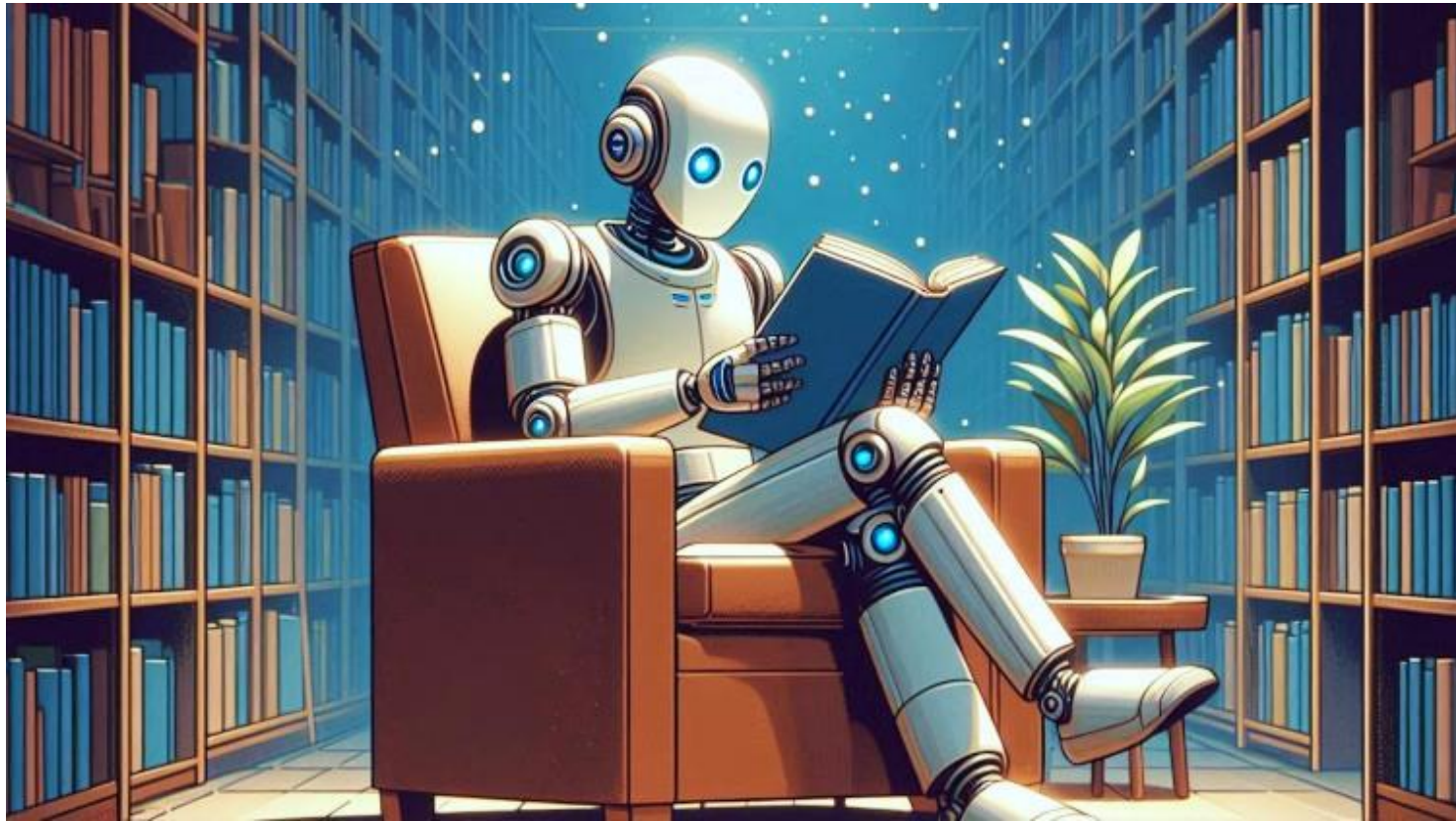


MASTER LLMs WITH LANGCHAIN



COURSE CONTENT

- Intuition about Large Language Models (LLMs)
- LLM with Hugging Face
- LLM with Langchain
- RAG Applications
- Agents and tools
- Project 1: Video transcription and summarization
- Project 2: Chatbot with memory and interface
- Project 3: Chat with your documents

- Requirements: Programming logic and Basic Python programming

WHAT ARE LLMs?

- **LLMs (Large Language Models)** – are very large Artificial Intelligence (AI) models that use Machine Learning techniques to understand and generate human language.
- They use Natural Language Processing (NLP) to interact, interpret, manipulate, and comprehend human language.

WHAT ARE LLMs?

- It is important to clarify and know the difference between all these concepts:
 - **Artificial intelligence (AI)**
 - **Machine Learning (ML)**
 - **Natural Language Processing (NLP)**
 - **Artificial Neural Networks (ANNs)**
 - **Generative AI**
 - **Large Language Models (LLMs)**

HOW LLMs WORK?

- They use unsupervised learning to understand language.
- A machine learning model is fed with datasets (hundreds of billions of words and phrases) which will be used to train it.
- The model learns from these examples without explicit human instructions.
- Extracts information from data, creates connections and "learns" about language.
- As a result, the model captures the complex relationships between words and sentences.

HOW LLMs WORK?

- LLMs require significant computational resources to process data.
- They use Graphics Processing Units (GPUs) to handle complex calculations.
- The use of GPUs is essential, which accelerates the training and operation of **transformers**

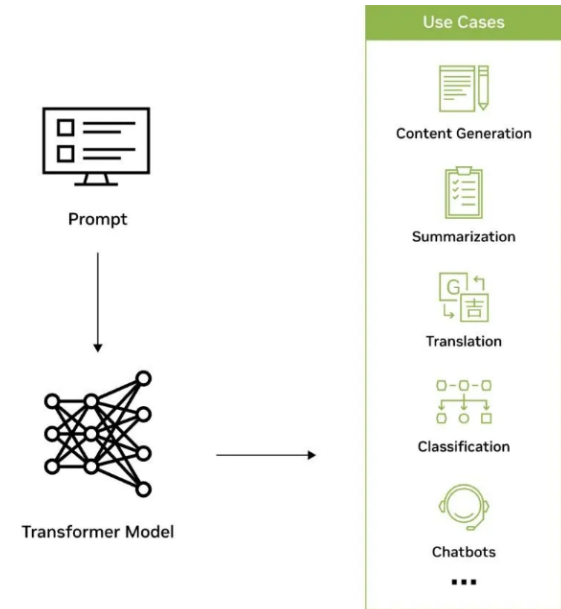
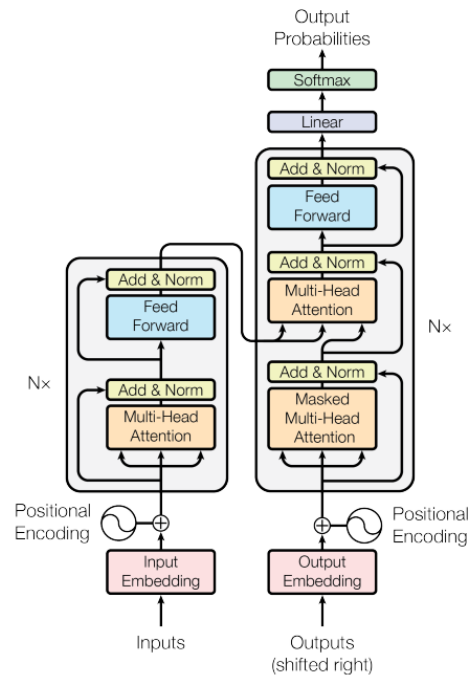


Image source: [NVIDIA](#)

TRANSFORMERS

- A transformer is a deep learning architecture that transforms or changes an input sequence into an output sequence.
- It was introduced in the paper "Attention Is All You Need", authored by eight scientists from Google.
- The transformer architecture improves the capability of ML models by capturing contextual dependencies in data sequences, such as words in a sentence.
- With billions of parameters, the transformer analyzes complex patterns and nuances of language.



Transformer Architecture.

More details in the official paper :

<https://arxiv.org/abs/1706.03762>

TRANSFORMERS

- For example, in the sentence "What color is the sky?", the model identifies the relationship between "color", "sky" and "blue" to generate as response "The sky is blue".
- It is a neural network that learns context and thus meaning by monitoring relationships in sequential data like the words in this sentence.



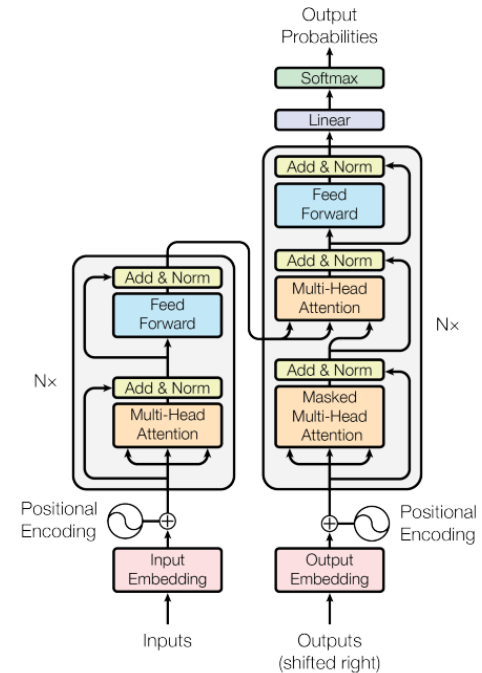
Illustrated guide to Transformers: <https://jalammar.github.io/illustrated-transformer/>

PREVIOUS ARCHITECTURES

- **RNN:** Had many limitations when processing very long sequences. Although they were able to capture temporal dependencies, their performance decreased as the sequence grew.
- **LSTM:** It can process larger sequences due to some new innovative mechanisms, which allow the preservation of important information for longer periods. However, it still faces difficulties in scalability and efficiency in training with large volumes of data.
- **Transformers:** They overcame these previous limitations by eliminating sequential dependency and using attention mechanisms that capture contextual relationships between all parts of a sequence simultaneously.

HOW TRANSFORMER ARCHITECTURE WORKS

- Traditional networks: encoder and decoder
- Challenges: details are lost in long sequences
- Proposed Solution: Self-Attention Mechanism
- This makes transformers more efficient and effective, especially with long texts and large datasets.



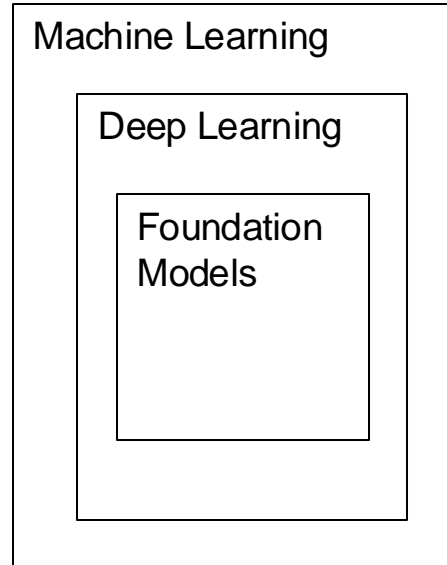
Transformer Architecture.

More details in the official paper:

<https://arxiv.org/abs/1706.03762>

FOUNDATION MODELS

- LLMs are a class of deep learning models known as **Foundation Models**.
- The term Foundation Models was popularized by Stanford University in 2021, which also founded the Center for Research in Foundation Models (CRFM) to study models such as BERT, GPT-3 and CLIP.
- While LLMs are language-specific models and therefore focus on text-related tasks, Foundation Models have a broader scope and can be applied to a variety of purposes; such as applications involving images, audio, videos, 3D signals and much more.



DELVING DEEPER INTO HOW LLMs WORKS

- Overview: Understand its main components, such as tokens (inputs) and logits (outputs).
- Encoder-Decoder and GPT: Revisit the Transformer encoder-decoder architecture and specifically the GPT architecture, which is decoder-only and used in many current LLMs.
- Tokenization: Convert raw text data into tokens that the model can understand, typically by breaking the text into words or subwords.
- Attention Mechanisms: Understand the theory behind attention mechanisms, such as self-attention and dot-product attention, which allow the model to focus on different parts of the input when producing an output.
- Text Generation: Explore the different ways to generate output sequences. Common strategies include greedy decoding, beam search, top-k sampling, and nucleus sampling.

THE MATH BEHIND LLMs

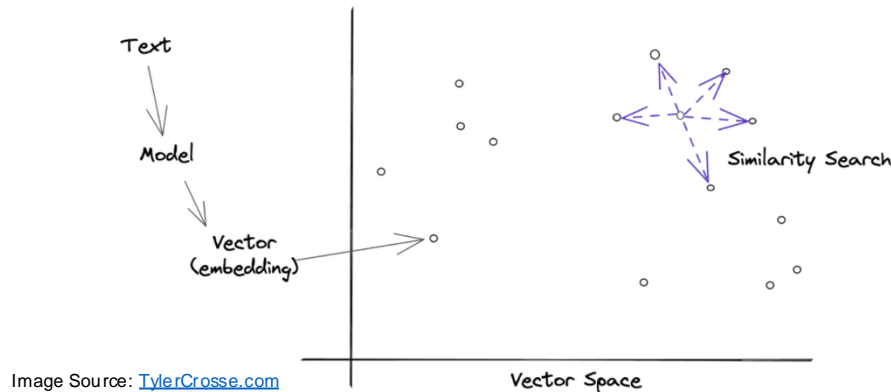
- Linear algebra: Essential for understanding many algorithms, especially in deep learning. Key concepts include: vectors, matrices, determinants, eigenvalues and eigenvectors, vector spaces, and linear transformations.
- Calculus: Required for the optimization of continuous functions in many machine learning algorithms. Important concepts include: derivatives, integrals, limits, and series, as well as multivariable calculus and gradients.
- Probability and statistics: Crucial to understanding how models learn from data and make predictions. Key concepts include: probability theory, random variables, probability distributions, expectations, variance, covariance, correlation, hypothesis testing, confidence intervals, maximum likelihood estimation, and Bayesian inference.

DELVING DEEPER INTO HOW LLMs WORKS

- [The Illustrated Transformer](#) by Jay Alammar: a visual and intuitive explanation of the Transformer model.
- [The Illustrated GPT-2](#) by Jay Alammar: Focused on GPT architecture, which is similar to Llama.
- [Visual intro to Transformers](#) by 3Blue1Brown: Simple and easy to understand visual introduction to Transformers.
- [LLM Visualization](#) by Brendan Bycroft: incredible 3D visualization of what happens inside an LLM.
- [nanoGPT](#) by Andrej Karpathy: a 2 hour YouTube video that explains how to reimplement GPT from scratch (for programmers).
- [Attention? Attention!](#) by Lilian Weng: Presents the importance and necessity of the concept of attention in a detailed and more formal way.
- [Decoding Strategies in LLMs](#): Provides code and a visual introduction to different decoding strategies for generating text.

EMBEDDINGS

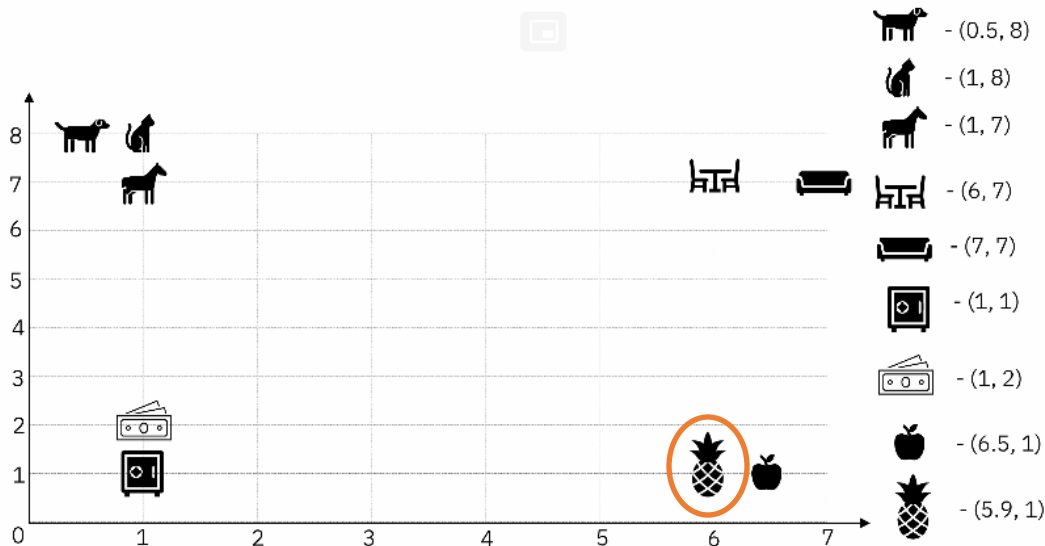
- **Embeddings** are numerical representations of textual data.
- In the context of LLMs, embeddings are used to transform words or sentences into vectors that the model can understand and process.
- For text, large language models like BERT and ChatGPT are great at generating embeddings that capture the semantic meaning.



EMBEDDINGS

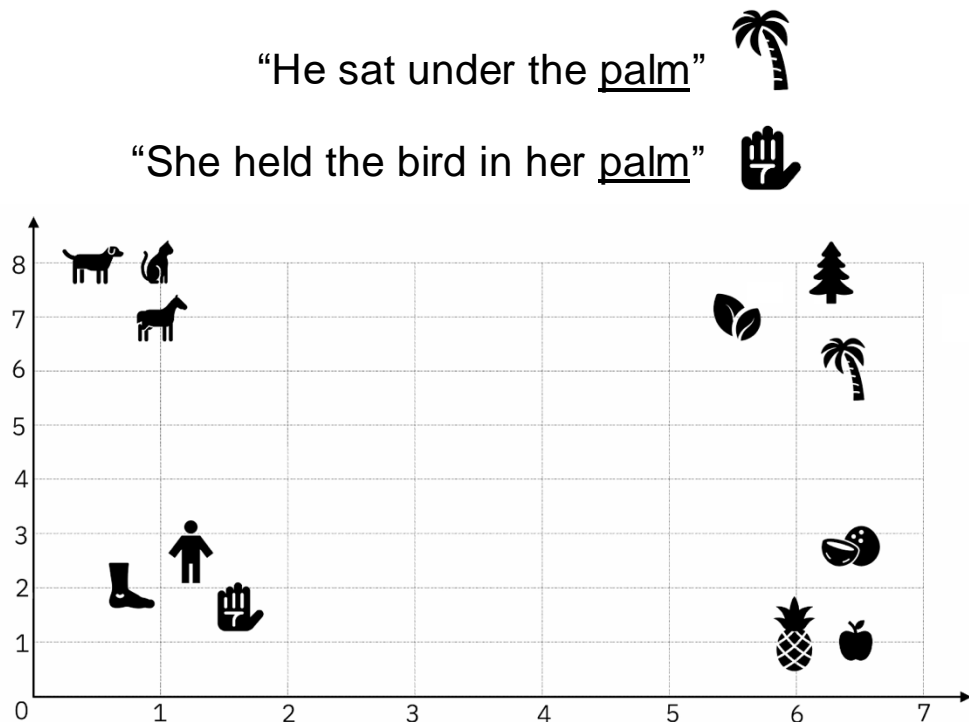
- The model learns to separate and group these texts based on their similarities.
- When this model receives a new word, like "pineapple", it knows exactly where to put it - most likely in the region where other fruits are.

Note that similar words (in this case, vectors) tend to group together in the vector space. These are points with close coordinates.



EMBEDDINGS

- And for the word "palm"?
- This is where the importance of the Attention Mechanism comes in: it allows the model to understand this context.
- For example, the word palm can refer to the inner surface of the hand or to a type of tree usually found in tropical regions.



With the context, the model correctly positions the embeddings, differentiating between the meanings that the word can have depending on the context.

EMBEDDINGS AND TOKENS

- Tokens are smaller units of text created through the process of tokenization, which converts text into numbers. In the context of LLMs, this can be an entire word, part of a word, or even an individual character.
- While Embedding is the vector representation of text in a multidimensional space, Token is a more static representation of text units.
- Tokens are converted to numbers so the model can understand and generate text.
- This mapping is done using a vocabulary. Each token has a specific index.
- Tokenization helps in preparing the text data for analysis by making it more structured and easier to work with.

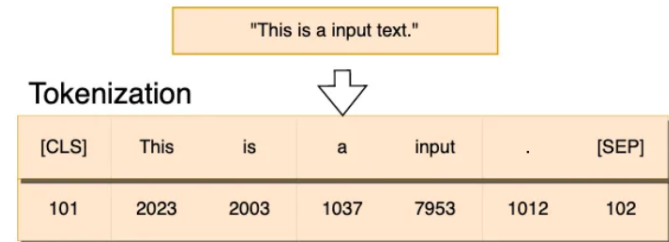


Image source: [Geoffrey Rathinapandi](#)

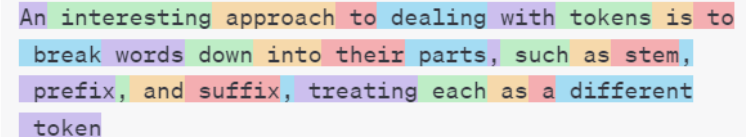
TOKENS

- An interesting approach to dealing with tokens is to break words down into their parts, such as stem, prefix, and suffix, treating each as a different token.
- Many words in our language share the same root. Likewise, suffixes in some words can be reused in others.

To further explore the Tokenization process, you can use Open AI's Tokenizer

<https://platform.openai.com/tokenizer>

Tokens	Characters
31	156



An interesting approach to dealing with tokens is to break words down into their parts, such as stem, prefix, and suffix, treating each as a different token

EVOLUTION AND HISTORICAL CONTEXT

- The first attempts at language models date back to the last century, when researchers began exploring the idea of teaching machines to understand and generate human language.
- In the 1990s, languages evolved into statistical models, analyzing linguistic patterns. Large-scale projects were limited by processing capacity.
- Advances in Machine Learning in the 2000s increased the complexity of language models. The Internet provided a wealth of training data.
- In 2012, advances in deep learning and larger datasets led to the development of generative pre-trained transformers (GPT).

EVOLUTION AND HISTORICAL CONTEXT

- The journey of LLM models as we know them today began with foundational models like GPT and BERT, which laid the foundation for more sophisticated systems.
- In 2018, Google introduced BERT, a major breakthrough in language model architecture. Two years later, OpenAI released GPT-3, with 175 billion parameters, setting a new standard for performance.
- When ChatGPT was launched in 2022, models like GPT-3 have become widely accessible via a web interface, increasing public awareness of LLMs.
- In 2023, open-source models gained attention for demonstrating impressive results, such as Dolly 2.0, LLaMA, Alpaca, and Vicuna. GPT-4 was released, setting new standards for size and performance. In 2024, open-source models were able to outperform proprietary models on several tasks, directly competing with state-of-the-art models such as ChatGPT and Gemini.

LLMs TIMELINE

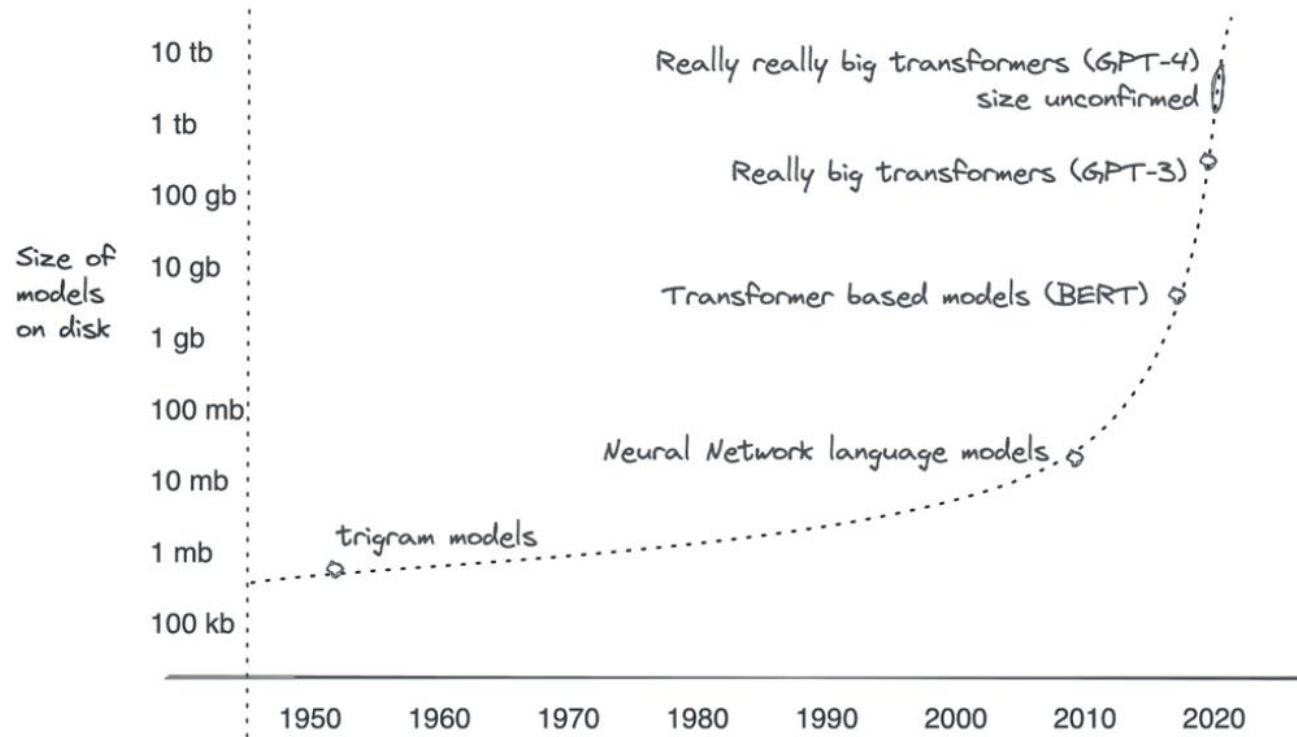


Image source: [TylerCrosse.com](https://tylercrosse.com)

WHY LLMs ARE SO IMPORTANT TODAY?

Automation and efficiency

LLMs perform language-related tasks, from customer support to data analysis and content generation. They also reduce operational costs and free up human resources for more strategic tasks.

Insight generation

They quickly scan large volumes of text, such as social media, reviews, and articles. They can help study market trends and analyze customer feedback, guiding business decisions.

Improving customer experience

They offer highly personalized content, increasing engagement and improving the user experience, e.g. chatbots for 24/7 customer service, personalized marketing messages, translations and communication between different cultures.

EXAMPLES OF APPLICATIONS WITH LLMS

- Chatbots and virtual assistants
- Code generation and debugging
- Content generation
- Content summarization
- Translation between languages
- Text classification and clustering
- Sentiment analysis

CHALLENGES AND LIMITATIONS OF LLMs

- Cost
- Privacy and Security
- Accuracy and Bias

HALLUCINATIONS IN LLMs

- Hallucinations in LLMs occur when the model generates incorrect or fictitious information that seems plausible.
- This happens because LLMs are designed to predict the next word or phrase based on learned patterns, but they have no real understanding of the world.
- They may combine words coherently, but without guaranteeing the veracity of the data.
- These errors can be critical in applications that require factual accuracy, such as customer service or medical reports.
- Therefore, it is important to validate the information provided by LLMs before using it as a reference, or implement other additional techniques that ensure greater reliability.

HALLUCINATIONS IN LLMs

Hallucinations can be caused by:

- Overfitting, encoding and decoding errors, and training bias (similar to the lack of diversity in the training data, which will limit the ability to generalize to new data), among others.

Researchers and professionals are taking several approaches to address the problem of hallucinations in LLMs.

- This includes improving the diversity of training data, eliminating inherent biases, using better regularization techniques, and employing adversarial training and reinforcement learning.

ETHICS AND CONCERNS ABOUT LLMs

- LLMs can amplify harmful biases and generate discriminatory content by relying on stereotypes present in the training data.
- The production of discriminatory and toxic content is a key concern, as training data often includes sociocultural stereotypes. This can lead to the generation of harmful texts against disadvantaged groups, based on race, gender, religion, ethnicity, etc.
- Privacy concerns are critical because LLMs train on large volumes of personal data, potentially leaking sensitive information such as social security numbers and addresses.
- LLMs can easily generate disinformation, producing fake content that appears trustworthy, which can have negative social, cultural and political impacts.

MODELS

- LLM models can be Closed Source (proprietary) or Open Source.
- Proprietary models are created by private companies and have restrictions on their use, while open-source models are free and modifiable, encouraging collaboration and innovation. Proprietary models can have trillions of parameters and capture deep linguistic nuances, but more parameters do not always guarantee better performance.
- Open source models, while generally smaller in terms of parameters, have shown the ability to compete with proprietary models, especially when well tuned and optimized by the community. Furthermore, the difference in maximum size between proprietary and open source models is increasingly smaller.

MODELS – CHOOSING BETWEEN OPEN SOURCE OR PROPRIETARY

Advantages of Open Source models

- **Transparency:** Provides full visibility into architecture, training data and processes, which is crucial for compliance and risk mitigation.
- **Customization and flexibility:** With access to the source code, you can adjust the models to meet specific needs.
- **Costs:** Generally more cost-effective, with no licensing fees, benefiting startups and small businesses.
- **Community and support:** Leverage the input of a global community of developers for continuous improvements.

Advantages of Proprietary models

- **Dedicated support:** They usually are easy to use and offer specialized technical support, essential for companies that depend on the models.
- **Optimized performance:** They often have specific features that can outperform open source models at certain tasks.
- **Security:** They can provide additional security guarantees, important for companies with sensitive data.

WHEN TO USE OPEN SOURCE MODELS

- Full control over data
- They allow customization of the source code, adjusting models to specific needs and facilitating continuous optimization.
- They have a superior cost-benefit ratio because they are generally free, which reduces costs and benefits startups and small businesses.
- While proprietary models offer technical support and optimizations, open source solutions provide greater transparency and control, with innovation driven by the global community.

SMALL LANGUAGE MODELS (SLM)

Small language models have gained a lot of popularity because they use much less computational resources, which democratizes their use in a local environment.

- SLMs can perform almost as well as larger models, but are faster and more practical for mobile devices and real-time systems.
- "Small" refers to the smaller number of parameters, ranging from a few million to a few hundred million, compared to billions in larger models.
- Some experts still prefer to call SLM very smaller models, with less than 1 billion parameters: for example GPT-2 Small (117 million), TinyBERT (15 million)
- Or even something between 1~3 billion parameters, like for example Google Gemma 2B which has 2 billion.

While LLMs like GPT-3 have billions or even trillions of parameters, SLMs have significantly fewer: in the range of millions to a few billion parameters..

	Llama 3	Meta 8 billion parameters
	Phi-3	Microsoft 3.8 billion - 7 billion parameters
	Gemma	Google 2 billion - 7 billion parameters
	Mixtral 8x7B	Mistral AI 7 billion parameters
	OpenELM	Apple 0.27 billion - 3 billion parameters

Image source: [Data Science Dojo](https://data-science-dojo.com/)

SMALL LANGUAGE MODELS (SLM)

- The downside is that they may not perform as well on complex tasks, such as understanding context or generating detailed text.
- They demonstrate excellent value for money considering how much they have been reduced in size.

Comparison of the larger (70B) and smaller (8B) versions of the Llama 3

	Meta Llama 3 8B	Mistral 7B		Gemma 7B	
		Published	Measured	Published	Measured
MMLU 5-shot	66.6	62.5	63.9	64.3	64.4
AGIEval English 3-5-shot	45.9	--	44.0	41.7	44.9
BIG-Bench Hard 3-shot, CoT	61.1	--	56.0	55.1	59.0
ARC- Challenge 25-shot	78.6	78.1	78.7	53.2 0-shot	79.1
DROP 3-shot, F1	58.4	--	54.4	--	56.3

	Meta Llama 3 70B	Gemini Pro 1.0	Mixtral 8x22B
		Published	Measured
MMLU 5-shot	79.5	71.8	77.7
AGIEval English 3-5-shot	63.0	--	61.2
BIG-Bench Hard 3-shot, CoT	81.3	75.0	79.2
ARC- Challenge 25-shot	93.0	--	90.7
DROP 3-shot, F1	79.7	74.1 variable-shot	77.6

Performance of the Llama 3 pre-trained model – Source: Meta

SMALL LANGUAGE MODELS (SLM)

The open source Phi-3-mini model is considered revolutionary for the SLM category.

It was released by Microsoft in the second quarter of 2024 and is based on a decoder-only transformer architecture, with a context length of 4,000 tokens.

	Phi-3-mini 3.8b	Phi-3-small 7b	Phi-3-medium 14b	Phi-2 2.7b	Mistral 7b	Gemma 7b	Llama-3-In 8b	Mixtral 8x7b	GPT-3.5 version 1106
MMLU (5-Shot) [HBK*21]	68.8	75.7	78.0	56.3	61.7	63.6	66.5	70.5	71.4
HellaSwag (5-Shot) [ZHB*19]	76.7	77.0	82.4	53.6	58.5	49.8	71.1	70.4	78.8
ANLI (7-Shot) [NWD*20]	52.8	58.1	55.8	42.5	47.1	48.7	57.3	55.2	58.1
GSM-8K (8-Shot; CoT) [CKB*21]	82.5	89.6	91.0	61.1	46.4	59.8	77.4	64.7	78.1
MedQA (2-Shot) [JFO*20]	53.8	65.4	69.9	40.9	50.0	49.6	60.5	62.2	63.4
AGIEval (0-Shot) [ZCG*23]	37.5	45.1	50.2	29.8	35.1	42.1	42.0	45.2	48.4
TriviaQA (5-Shot) [JCWZ17]	64.0	58.1	73.9	45.2	75.2	72.3	67.7	82.2	85.8
Arc-C (10-Shot) [CCE*18]	84.9	90.7	91.6	75.9	78.6	78.3	82.8	87.3	87.4
Arc-E (10-Shot) [CCE*18]	94.6	97.0	97.7	88.5	90.6	91.4	93.4	95.6	96.3
PIQA (5-Shot) [BZGC19]	84.2	86.9	87.9	60.2	77.7	78.1	75.7	86.0	86.6
SociQA (5-Shot) [BZGC19]	76.6	79.2	80.2	68.3	74.6	65.5	73.9	75.9	68.3

More information can be found in the paper: <https://arxiv.org/abs/2404.14219>

SMALL LANGUAGE MODELS (SLM)

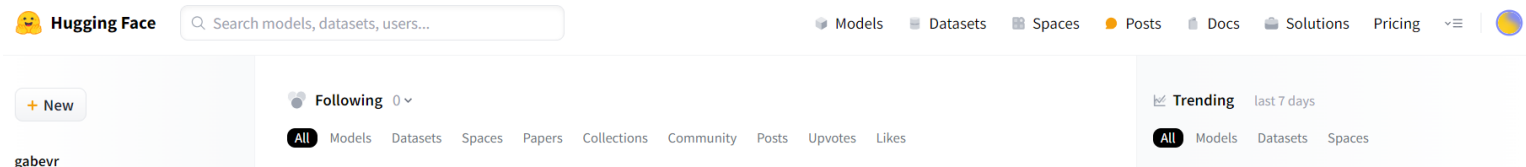
- **What makes SLMs so interesting?**
 - More affordable and accessible: Can be run (in inference mode) on devices with limited resources, such as laptops or less powerful GPUs.
 - Easier to customize: Small models can be fine-tuned more quickly and on a single GPU.
 - More energy-efficient: Small language models require fewer computational resources, making them more energy-efficient.
 - Valuable for educational purposes: They are more manageable and therefore easier to understand and tune.

HOW TO CHOOSE MODELS

- In order to choose the most modern (state-of-the-art) LLM models, it is crucial to be constantly updated with the latest releases in technology repositories, as new models emerge frequently.
- Given the rapid evolution and growing interest in the field, the models presented here can quickly be considered “outdated” in a way, as new advances are made every quarter and new models are published all the time.
- Therefore, keeping up with innovations is essential to ensure the relevance and effectiveness of the models used.
- In this course, we will cover an implementation that simplifies the exchange of LLMs, allowing you to test different models without having to rewrite all the code, which greatly facilitates experimentation and consequently leads to better results.

WHERE TO FIND MODELS

- To stay up to date with the latest LLM models, it's essential to constantly research repositories like Hugging Face, to check comparative articles, and follow leaderboards, as we'll see below.
- The key is to learn to adapt and incorporate new models and techniques as they emerge, ensuring that you're always working with the most advanced and relevant solutions.



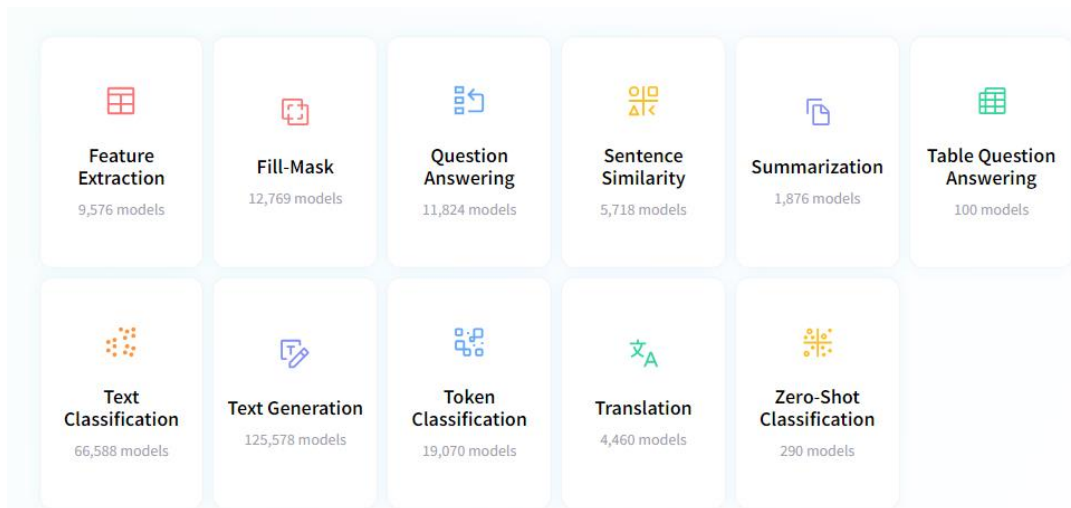
<https://huggingface.co>

HUGGING FACE

- Hugging Face is an open-source repository that offers models, datasets, and tools.
- It is especially known for its Transformers library, designed primarily for NLP applications.
- In addition to the Transformers library, there are thousands of language models available for free on Hugging Face.
- Hugging Face has emerged as a key player in the LLM ecosystem by offering an open-source platform and tools that simplify the use of these models.
- This accessibility has enabled developers and researchers to harness the power of LLMs with ease and efficiency.

HUGGING FACE - HOW TO BROWSE AND FIND MODELS

Access: <https://huggingface.co/tasks>



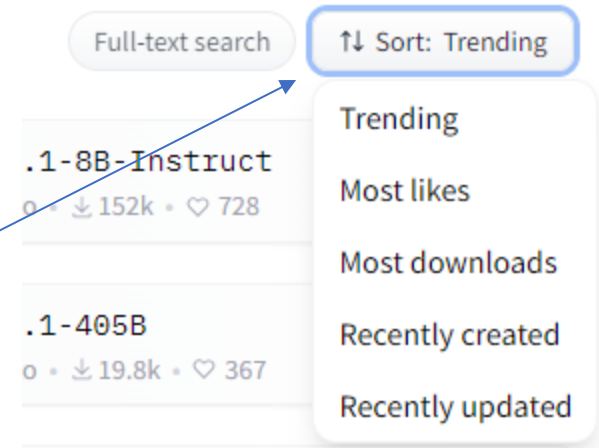
In addition to NLP models, Hugging Face also brings together numerous models from other areas of AI, such as Computer Vision. In addition, within the platform itself you have access to demos, use cases, datasets, forums and more.

HUGGING FACE - HOW TO BROWSE AND FIND MODELS

To access LLMs, go to the “Text Generation” option, or access the link below

https://huggingface.co/models?pipeline_tag=text-generation&sort=trending

Tip: Filter by trending models, or those with the most likes or downloads.

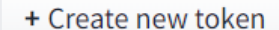


HUGGING FACE – CREATING AN ACCOUNT AND GENERATING AN API TOKEN

- Although it is not mandatory in general, there are some specific cases where it is necessary to have an account, such as models that require authorization.
- Therefore, it is a good idea to register now: <https://huggingface.co/join>

Generating an API key

- Access <https://huggingface.co/settings/tokens>
- Select “Create new token”
- Set “Read” permission
- Copy the key



Creating this token is completely free and there is no charge for uploading models.

HUGGING FACE – CREATING AN ACCOUNT AND GENERATING AN API TOKEN

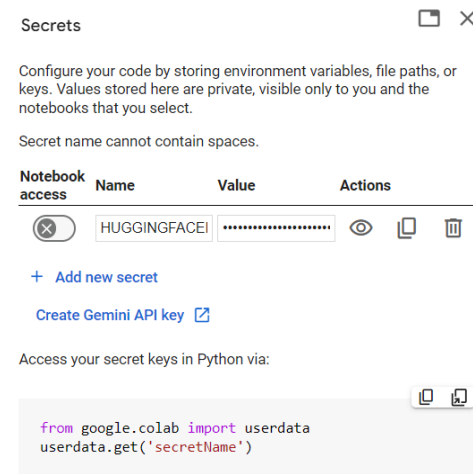
To use the token in Google Colab notebook, use this code below:

- `os.environ["HF_TOKEN"] = getpass.getpass()`

And in the field that appears, paste your Hugging Face key.

Alternative: You can also add it to Colab's secret keys like this:

- Open your Google Colab notebook.
- On the left side of the page, you will see a key icon. Click on it.
- Click on “Add a key” and enter your access token.



HUGGING FACE

- The Hugging Face API integrates well with LangChain, an open source framework that connects LLMs, data sources, and other functionality with a unified syntax.
- Throughout this course, we will explore this powerful tool. But first, we will learn how to implement LLMs using the Transformers library.
- While LangChain provides more options, knowing this approach can be useful if you are testing new and recently published model that does not yet have much support. Even with LangChain, when dealing with literally thousands of different models, there may be some incompatibility when loading them.




ACCESSING MODELS ON HUGGING FACE

- Some specific models require approval to be downloaded and used. However, for most of them this is not necessary.
- If so, then you usually just need to fill out a simple form to obtain access permission.
- The approval usually takes a few minutes, but depending on the number of requests, the time may vary.

For example, the Meta Llama 3 model

<https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct>

 You need to agree to share your contact information to access this model

The information you provide will be collected, stored, processed and shared in accordance with the [Meta Privacy Policy](#).

LLAMA 3.1 COMMUNITY LICENSE AGREEMENT

Llama 3.1 Version Release Date: July 23, 2024

"Agreement" means the terms and conditions for use, reproduction, distribution and modification of the Llama Materials set forth herein.

"Documentation" means the specifications, manuals and documentation accompanying Llama 3.1 distributed by Meta at

<https://llama.meta.com/doc/overview>.

"Licensee" or "you" means you, or your employer or any other person or entity (if you are entering into this Agreement on such person or...

Once released, you should receive a response in your email.

This is to let you know your request to access model "meta-llama/Meta-Llama-3.1-70B" on huggingface.co has been accepted by the repo authors.

You can now access the repo [here](#), or view all your gated repo access requests [in your settings](#).

Cheers,

The Hugging Face team

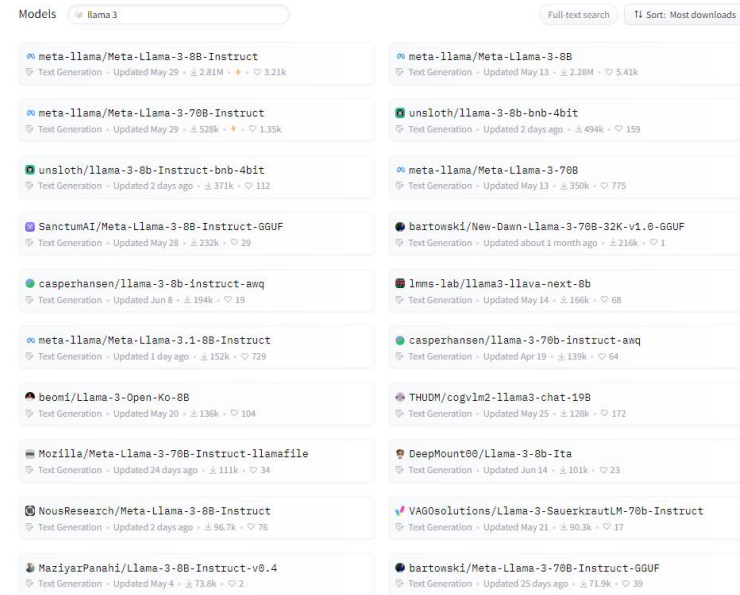
Click on the button within this section and fill in the requested data.

ACCESSING MODELS ON HUGGING FACE

Alternatively, you can use variations of these models, fine-tuned versions published by the community.

For example, see below how many options appear when we search for “llama 3”

We can also be more specific, searching by the model format (example: GGUF) or other characteristics, such as language e.g. “portuguese” (we will see later why it may be interesting to be more specific in our search)



https://huggingface.co/models?pipeline_tag=text-generation&sort=trending

TYPES OF MODELS

- When browsing generative AI model repositories like Hugging Face, you will notice models listed with the suffixes 'instruct' or 'chat'.
- This is because there are at least three main types of LLMs:
 - **Base models** - they only go through pre-training and complete texts with the most likely words.
 - **Instruct-tuned** - Instruction-Tuned Models go through an additional step of tuning for instructions, improving the ability to follow specific commands.
 - **Chat models** - have been better tuned to work in chatbots, so they may be more appropriate for conversations. Some chat models can be called instructional and vice versa depending on the situation, so to make it easier you will find situations where a model used for chat is classified as instructional.

TYPES OF MODELS

- The 'instruct' version of the model has been adjusted to follow provided instructions. These models 'expect' to be asked to do something.
- Models with the 'chat' suffix are tuned to work on chatbots, 'expecting' to be engaged in a conversation with different actors.
- In contrast, models not tuned for instructions simply generate output that continues from the prompt.
- Example of which model to choose:
 - For building chatbots, implementing RAG or using agents, use 'instruct' or 'chat' models.
 - When in doubt, use an 'instruct' model.

TYPES OF MODELS

Base models are good at predicting subsequent words from large volumes of text, but they tend to generate generic responses because they are trained to complete sentences, not to answer questions directly.

- For this reason, if you ask a base model *“Where do penguins live?”* a plausible answer would be: *“What do they eat? What do they look like? [...]”*

Why does this happen? Remember that LLMs are designed to complete sentences by predicting the most likely words based on previous text.

- For example, for *“The sky is,”* they will likely complete with “blue” because it is the most common sequence.
- This is because base models are trained to predict the next word in large volumes of text.
- They do not answer questions directly, because statistically, questions often follow other questions in their training data.
- Therefore, their behavior reflects these statistics, not a lack of intelligence on the part of the model.

TYPES OF MODELS

A possible solution would be to modify the prompt a little, to give the model a hint.

- Taking the example of penguins, we could add after the question, changing from “*Do penguins live?*” to: “*Where do penguins live? They live*”.
- By doing this we should have the expected result, because remember that this type of model completes the sentence.

This is also known as **prompt engineering**.

It can work well to a certain extent, but there are limitations.

Furthermore, this is not how the large and impressive LLM models we know work. For example, we don't need to give these hints to ChatGPT, it already understands what to do. This is because these models are *Instruct-tuned*.

To do this and thus arrive at these superior results that are more appropriate to certain scenarios (such as expect of an answer or simulate a conversation) it is used a technique known as **fine-tuning**.

FINE-TUNING

- Fine-tuning is an additional step in the process of creating a model, improving its ability to perform specific tasks.
- It involves taking the pre-trained base model and training it further on a smaller, more specialized dataset that is relevant to the desired task.
- For instructions: The goal is not to teach the model about the information itself (such as *"Where do penguins live?"* or *"What is the capital of England?"*), but rather to teach it that when a question is asked, we expect an answer in that format.
- The fine-tuning process adjusts the model's parameters to better align with the nuances and requirements of these tasks, such as question answering, without losing the general understanding of the language it already has.

The data provided for training is labeled, with example generations for a specific prompt.

Example of a prompt dataset:

- *Input: "Where do penguins live?"*
- *Output: " They live mainly in the Antarctic region. [...]"*
- *Input: " What is the capital of England?"*
- *Output: "The capital of England is London".*

INSTRUCT-TUNED MODELS

Therefore, when developing applications with LLM, we are usually more interested in Instruct-tuned models, as they are tuned to receive instructions.

- After pre-training, where they assimilate a huge amount of text without a specific purpose other than to reproduce these texts, the models go through another round of training.
- This second round is known as Tuning, or more precisely, **Instruct Tuning**.
- First, the model acquires knowledge and then learns how to make this knowledge more useful to users inputs.
- At this point, the model learns that, when given a question, it should generate an answer, and not another question.

INSTRUCT-TUNED MODELS

- This adjustment of the model to make it follow instructions is done through a second training phase.
- We feed the LLM with a dataset of instructions during this new training phase.
- The model is taught that when there is a question in the input, the output should be an answer to that question.
- When there is a request for an activity, the model should perform that activity.

TYPES OF MODELS

- For example, the Llama 3 model, an open source model from the company Meta, which in the second quarter of 2024 proved that it can outperform in several tasks the most modern proprietary models such as ChatGPT.

<https://huggingface.co/meta-llama/Meta-Llama-3-8B>

[∞ meta-llama/](https://huggingface.co/meta-llama/Meta-Llama-3-8B)**Meta-Llama-3-8B**

<https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct>

[∞ meta-llama/](https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct)**Meta-Llama-3-8B-Instruct**

CHOOSING MODELS

What is interesting to pay attention when choosing the model:

- Data Freshness – the maximum date of the most recent training data. That is, this show how up-to-date the data is.

For example, Meta-Llama-3.1-8B-Instruct has until December 2023

- On the model description page there may also be other interesting information, such as supported languages.

Evaluate models with Benchmarks

- Additionally, it may be interesting to pay attention to the metrics, although it is more relevant to access other locations - such as leaderboards or curations and comparative tests - to see how the models behave in widely accepted benchmarks.

LEADERBOARDS

Leaderboards are rankings that compare different LLM models based on performance metrics. They help to:

- Evaluate and compare the effectiveness of different models.
- Identify the most up-to-date and effective models.
- Provide insights into which models are best suited for different applications.
- Help developers and researchers choose models by showing which ones are the best rated in terms of accuracy and efficiency.

These rankings are very important if you want to stay up to date on LLMs.

- Example of a highly referenced leaderboard:

<https://lmarena.ai/?leaderboard>

MODELS FOR SPECIFIC LANGUAGES

- If we are going to develop an application where the answers will be returned in English, then we can use only these best-known leaderboards as a reference.
- However, if you want to develop apps in other language, it is interesting to search for models that perform well on datasets in the language you want.
- To do this, we can look for leaderboards focused on the desired language.
- You can find these leaderboards usually on Hugging Face Spaces.
- For example, for Portuguese there's this leaderboard:
https://huggingface.co/spaces/eduagarcia/open_pt_llm_leaderboard

And here is an updated list with the best models: <https://huggingface.co/collections/eduagarcia/portuguese-llm-leaderboard-best-models-65c152c13ab3c67bc4f203a6>

MODELS FOR SPECIFIC LANGUAGES

Here you can filter by several different attributes – such as model size (how many billion parameters), or type (open source or proprietary).

Additionally, you can sort the table based on the most interesting metric/dataset, just click on the column name in the table.

LLM Benchmark

Metrics

About

Submit here!

Changelog

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

Select columns to show

☒ Average

☒ ENEM

☒ BLUEX

☒ OAB Exams

☒ ASSIN2 RTE

☒ ASSIN2 STS

☒ FAQUAD NLI

☒ HateBR

☒ PT Hate Speech

☒ tweetSentBR

☐ Type

☐ Architecture

☐ Precision

☐ Merged

☐ Hub License

☐ #Params (B)

☐ Hub

☐ Model sha

☐ Revision

☐ Evaluation Time (s)

☐ Leaderboard Average

☐ NPM (Average)

☐ Main Language

Hide models

☐ Proprietary

☐ Private or deleted

☐ Contains a merge/merge

☒ Flagged

☐ MoE

Model types

☒ pretrained

☒ language adapted (FP, FT, ...)

☒ fine-tuned/tp on domain-specific datasets

☐ chat (RLHF, DPO, IFT, ...)

☒ base merges and moerges

☒ proprietary (closed)

☐ ?

Precision

☒ float16

☒ bfloat16

☒ 8bit

☒ 4bit

☒ GPTQ

☐ ?

Model sizes (in billions of parameters)

☐ ?

☒ ~1B

☒ ~3B

☒ ~8B

☒ ~16B

☒ ~35B

☒ ~70B

☒ 100B+

Model Main Language

☒ Portuguese

☒ English

☒ Chinese

☒ Spanish

☒ Other

☐ ?

ENVIRONMENT FOR IMPLEMENTATION

We will use Colab to take advantage of the free GPU, we recommend doing so, at least initially. Later, you'll learn how to use with CPU only.

Note about Colab: if you reach your GPU usage quota, you will need to wait a few hours before you can use it again.

If this happens, you can:

- Run locally (if you don't have good hardware, it needs to be via API)
- Use paid solutions (like Open AI)
- Using another service that provides free GPUs, such as SageMaker Studio Lab or Kaggle

IMPLEMENTATION WITH TRANSFORMERS LIBRARY (HUGGINGFACE)

Model chosen for initial tests: Phi-3 Mini-4K-Instruct

<https://huggingface.co/microsoft/Phi-3-mini-4k-instruct>

- Why we chose it: it is open source, accessible and has a good multilingual support. You will see that many models do not understand other languages besides english, and those that do understand are too heavy to run in our environment, that is, we need to access it via an API or web interface, such as ChatGPT. However, at this moment we want to explore open source solutions, to obtain more control.
- Phi models offer many of the same capabilities as LLMs, but are smaller in size and trained on smaller datasets, making them more cost-effective and accessible.
- The Phi-3 family, including models like the Phi-3-mini, performs impressively across multiple benchmarks.

MODEL DETAILS

microsoft/Phi-3-mini-4k-instruct

Organization (*microsoft*):

- This can be a company or individual that maintains the model. This part indicates that the model is developed and maintained by Microsoft, a leading technology company with extensive research and development in AI and machine learning.

Model Family (*Phi-3-mini*):

- Phi-3: This denotes the third iteration of Microsoft's Phi model series.
- Mini: This indicates that this particular model is a smaller version within the Phi-3 family. It is designed to be efficient and lightweight, while still offering robust performance.

MODEL DETAILS

microsoft/Phi-3-mini-4k-instruct

4k (size): This refers to the size of the model's context window or sequence length, which in this case is 4,000 tokens. This means that the model can handle up to 4,000 tokens in a single input, allowing it to process and generate longer text sequences more efficiently.

Instruct: this indicates that the model is tuned for instruction-following tasks, as mentioned earlier. Models with "instruct" in their name are typically optimized to understand and generate responses based on specific instructions or prompts, making them well-suited for tasks like question answering, guided text generation, and more.

TOKENIZER

- In our setup, we also need to load the tokenizer associated with the model. The tokenizer is crucial for preparing text data in a format that the model can understand.
- A tokenizer converts raw text into tokens, which - as discussed earlier - are numeric representations that the model can process.
- It also converts the model output tokens back into human readable text.
- Tokenizers handle tasks like splitting text into words or subwords, adding special tokens, and managing vocabulary mapping.

TOKENIZER

Tokenizer Definition

- Vocabulary management: The tokenizer includes a vocabulary that maps each token to a unique identifier. This ensures that the same words or subwords are consistently represented by the same tokens.
- Special tokens: The tokenizer handles special tokens such as [CLS], [SEP], and [PAD], which are used for specific purposes in model processing (e.g., indicating the start or end of a sequence, padding shorter sequences).
- Handling long sequences: For example, the Phi-3-mini-4k-instruct model. Since this model is optimized for a context window of up to 4,000 tokens (4k), the tokenizer handles long sequences efficiently, truncating or padding them appropriately as needed.

PIPELINE

What is a pipeline?

- A pipeline is an abstraction that simplifies the use of pre-trained models for a variety of NLP tasks. It provides a unified API for different tasks such as text generation, text classification, translation, and more.
- Pipelines handle the entire process of taking raw input data, pre-processing it using the tokenizer, running it through the model, and post-processing the model output to produce human-readable results.
- This high-level interface abstracts away the complexities of working directly with models and tokenizers, making it easy to perform common tasks with minimal code.

PROMPT ENGINEERING

- **Prompt engineering** is the art of formulating and optimizing instructions given to generative AI models to obtain more accurate and relevant responses.
- It involves customizing the input or adding commands that guide the model to produce the desired output, maximizing efficiency and response quality.
- It is a highly empirical process, so it is important to test variations thoroughly.

Simple example:

- **Before Prompt engineering :**

Question: "Tell a story."

Response: " Once upon a time there was a princess who lived in a castle."

- **After Prompt engineering :**

Question: " Tell a story about a brave princess who saves her kingdom from a dragon using her wits"

Response: " Once upon a time there was a brave princess who, using her wits, saved her kingdom from a terrifying dragon by using an ingenious plan."

IMPROVING RESULTS WITH PROMPT ENGINEERING

- Always check if your prompt could be more specific. If even after improving the prompt you are having difficulty achieving the expected result (and after trying changing the parameters), then the model is not so appropriate for this task.
- For example, for code generation: An idea/suggestion for a prompt that could be convenient for code generation, if you want to use LLM as your co-pilot: *"Refactor using concepts such as SOLID, Clean Code, DRY, KISS and if possible apply one or more appropriate design patterns aiming at scalability and performance, creating an organized folder structure and separating by files"* (and of course, here you can modify as you wish)
- Note: sometimes it is better to keep the prompt simple and not too elaborate. Including too many different information or references can "confuse". Therefore, it is quite interesting to add or remove term by term if you are experimenting and looking for better results.

PROMPT ENGINEERING

Some examples of online tools and services that can help with prompt engineering



For example LangSmith, which is part of the LangChain ecosystem <https://smith.langchain.com/hub>

IMPROVING RESULTS WITH OTHER MODELS

- Prompt engineering can only improve to a certain extent, depending on the complexity of the request. To achieve more assertive results, you can look for larger and more modern models with more parameters (remember the trade-off between efficiency and quality of responses).
- Here it can be very useful the tips provided above, about looking for newer models, that are trending or that have higher scores for specific benchmarks.
- An alternative is to look for models that focus on the desired task, for example code generation or conversations/chat.
- Let's imagine we want to use it for code generation; for this purpose, you could use the *6.7B deepseek-coder* model for example (or look for others with this goal).

QUANTIZATION

- **Quantization** reduces the precision of the numbers used to represent the model parameters, reducing memory usage and computational requirements.
- Instead of using 32-bit floating-point numbers (float32), the model can use 16-bit numbers (float16) or even 8-bit numbers (int8).
- This process can significantly reduce the size of the model and speed up inference without causing a substantial loss in accuracy.
- **Benefits for LLMs:**
 - Allows you to run large models on hardware with limited resources.
 - Maintains good performance without significantly compromising accuracy.
 - Facilitates the execution of LLMs on mobile devices and real-time systems.
 - Ideal for those who use platforms such as Google Colab with free resources.
- In our initial examples, we used 4-bit quantization (from the BitsAndBytesConfig module of the Transformers library) <https://huggingface.co/blog/4bit-transformers-bitsandbytes>

QUANTIZATION

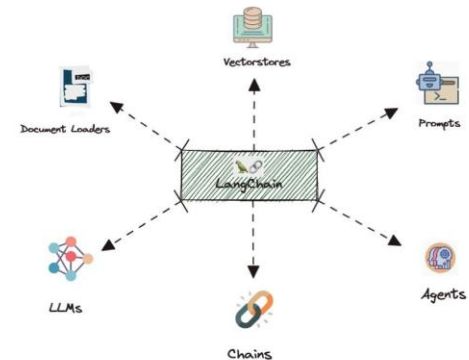
- Fine-tuning can also be used to improve performance, in addition to adding desirable behaviors (or removing undesirable behaviors).
- However, fine-tuning very large models is expensive; for example, regular 16-bit fine-tuning of a 65 billion parameter LLama model requires over 780 GB of GPU memory.
- Quantization approaches such as QLoRA help reduce the memory requirements for fine-tuning a 65 billion parameter model from over 780 GB of GPU memory to less than 48 GB, without degrading runtime or predictive performance compared to a fully tuned 16-bit baseline.

LANGCHAIN

- It is an open source library designed to facilitate the integration of LLMs with various data sources and additional functionalities.
- Developed by Harrison Chase and released in October 2022.
- It provides a unified syntax that simplifies the use of LLMs in different contexts, such as chatbots, text analysis, and question-and-answer systems.
- LangChain makes it easy to combine LLMs with other tools and services.
- In addition, it allows developers to create advanced natural language processing solutions in an efficient and scalable way.
- It is a very important solution to run LLMs programmatically and create own applications.



LangChain



LANGCHAIN – MAIN COMPONENTS

- **Models:** Provides a standard interface for interact with a wide range of LLMs.
- **Prompts:** Tools to simplify the creation and handling of dynamic prompts.
- **Chains:** Standard interface for chaining LLMs into complex applications, allowing chaining between multiple models or other specialized modules.
- **Memory:** Modules that allow the management and modification of previous conversations, essential for chatbots that need to remember past interactions to maintain coherence.
- **Agents:** Equipped with a comprehensive toolkit, they can choose which tools to use based on user input.
- **Indexes:** Methods for organizing documents (containing proprietary data, for example) in ways that facilitate effective interaction with LLMs.

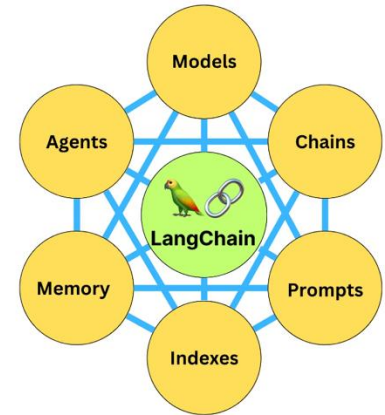


Image Source: [ByteByteGo](#)

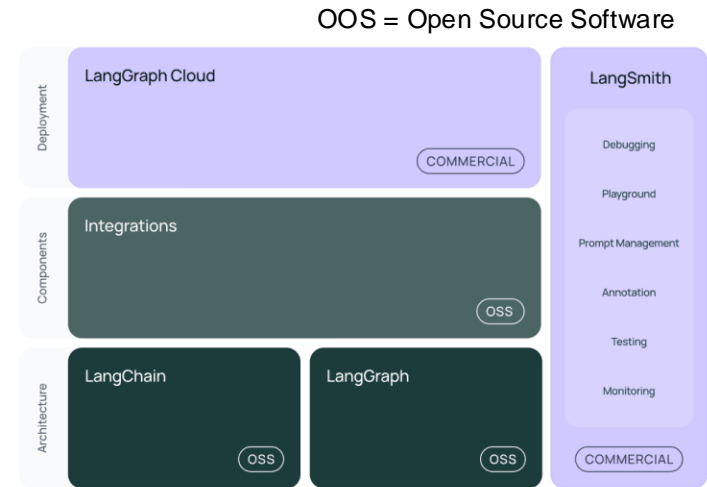
IMPLEMENTATION WITH LANGCHAIN

Important considerations:

- LangChain is a library in constant development, designed to facilitate the construction of applications powered by large language models.
- It is important to pay attention to the warnings that appear during use, as some functions may be deprecated and stop working in future versions.
- Keeping the library always in the same version can avoid deprecation issues, but it is recommended to update periodically to have access to the most modern and optimized features.
- LangChain represents a cutting-edge solution, and frequent updates are the price to pay if we want to enjoy the best tools available for building advanced applications.
- Therefore, depending on when you are watching, some functions may need to be updated. If necessary, we will update the code in Colab (or you can check the warning message, which in some cases explains what you need to change)

LANGCHAIN ECOSYSTEM

- langchain-core: Basic abstractions and LangChain Expression Language (LCEL).
- langchain-community: Third-party integrations. Partner packages (e.g. langchain-openai, langchain-anthropic, etc.): Some integrations have been split into their own lightweight packages that only depend on langchain-core.
- langchain: Chains, Agents and Retrieval Strategies that make up the cognitive architecture of an application.
- LangGraph: For building robust, stateful, multi-actor applications with LLMs, modeling steps as edges and nodes in a graph. Integrates seamlessly with LangChain, but can be used without it.
- LangServe: To implement LangChain chains as REST APIs.
- LangSmith: A platform for developers to debug, test, evaluate and monitor LLM applications.



<https://python.langchain.com/v0.2/docs/introduction/>

LANGCHAIN + HUGGING FACE

- The integration of Hugging Face with LangChain brings several benefits, as mentioned above.
- Hugging Face provides a wide range of pre-trained models that can be easily incorporated into your applications.
- LangChain makes this easy by providing a uniform interface and additional tools to improve performance and efficiency.
- With these tools combined, you can focus more on business logic and less on technical issues.

LANGCHAIN – MODELS

- One of the main advantages of LangChain is that it allows you to easily work with a variety of models. Some models are better for certain tasks or offer better value for money. Therefore, you will probably explore different models during your experiments.
- LangChain has integrations with many model providers (OpenAI, Cohere, Hugging Face, Anthropic, Google, etc.) and provides a standard interface to interact with all of these models.
- In LangChain, when working with models we define what is called an LLM Wrapper. A wrapper is like a "packaging", that makes it easier to use Large Language Models in applications.
- The LLM itself acts as the "brain" or engine of the application, performing natural language processing.

LANGCHAIN – MODEL INPUT AND OUTPUT

LangChain does not host its own LLMs, but relies on third-party integrations for interacting with many different LLMs.

The library divides model management into **LLMs** and **Chat Models**.

- LLMs - To be specific, the interface takes as input a string and returns a string.
- Chat Models – similar, but uses chat messages as inputs and returns chat messages as outputs.

As we develop our application, we will understand the differences and use cases for each.

LangChain provides the building blocks to interact with any language model.

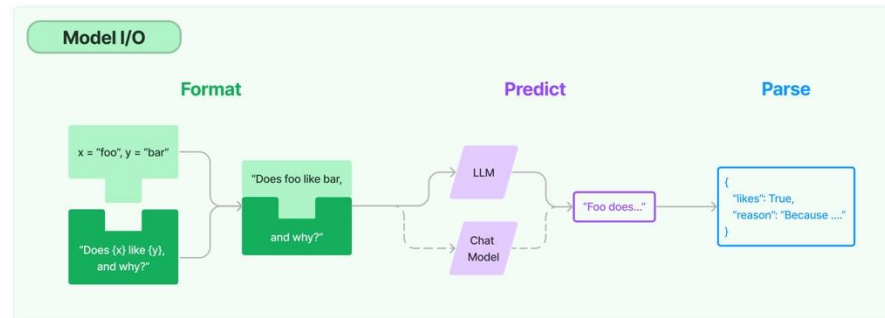


Image source: [LangChain Docs](#)

COMPONENTS - LLMS AND CHAT MODELS

Component - Chat Models

- Newer language models take message sequences as inputs and return chat messages as outputs.
- These models allow messages to be assigned distinct roles, differentiating between AI, users, and system instructions.
- While they work with both input and output messages, LangChain wrappers allow these models to take strings as input. This makes it easier to use chat models in place of traditional LLMs.
- When a string is passed as input, it is converted to a HumanMessage and then processed by the model.

Component - LLMs

- Language models that take a string as input and return a string.
- Traditionally, these are older models (newer models loaded in LangChain are usually chat models).
- While the underlying models work with string in/string out, LangChain wrappers allow these models to take messages as input.
- Incoming messages are formatted into a string before being passed to the model.

COMPONENTS - LLMS AND CHAT MODELS

- In other words: Plain text input/output LLMs tend to be older or lower level.
- Many popular models are best used as chat completion models, even for non-chat use cases.
- LangChain prioritizes Chat Models because they are more closely associated with more modern models (at least in the current version).

MESSAGES

Some language models take a list of messages as input and return a message. There are a few different types of messages.

In LangChain, all messages have a ``role``, ``content``, and ``response_metadata`` property.

- The role describes who is saying the message (e.g. human, system). LangChain has different message classes for different roles.
- The content property describes the content of the message, which can be either: a string (most models handle this type of content); or a list of dictionaries (this is used for multimodal input, where the dictionary contains information about this input type and this input location).
- The *response_metadata* property contains additional metadata about the response. The data here is often specific to each model provider. This is where information such as *log-probs* and token usage can be stored.

QUANTIZATION SOLUTIONS

Alternatively, instead of performing the quantization ourselves using BitsAndBytesConfig, we can download already quantized models and also in compact and efficient file formats such as **GGUF**, compared to conventional formats such as Pytorch and Safetensors.

Other optimization approaches :

- **GPTQ** - Generalized Post-Training Quantization
- **AWQ** - Activation-aware Weight Quantization

At Hugging Face we can find hundreds of models converted to this format.

- For example for the GGUF format, you can check it out here
https://huggingface.co/models?pipeline_tag=text-generation&sort=trending&search=gguf

To perform model inference in this format we can use the Llama.cpp library, which is compatible with LangChain.

COMPACT AND EFFICIENT FILE FORMATS FOR LLMs

GGML: Created by Georgi Gerganov. Makes it easy to use models on various hardware, including CPUs.

- GGML limitations: Compatibility issues and need for complex manual adjustments.

GGUF: The Evolution of GGML, released in August 2023. Developed by the AI community, including Gerganov.

- Goal: Address limitations of GGML.
- Offers greater flexibility and stability while maintaining compatibility with older models.

Advantages and Disadvantages of GGUF

- Pros: Improved user experience, support for a variety of models including Meta's Llama and many others.
- Cons: Conversion of existing models can be time-consuming and require users to adapt.

PROMPT TEMPLATES

- Prompt templates help translate user input and parameters into instructions for a language model.
- It can be used to guide a model's response, helping it understand the context and generate relevant and coherent language-based output.
- It accepts a set of parameters from the user that can be used to generate a prompt.
- With Langchain, we have an efficient way to connect this to the different LLMs that exist.
- To change the LLM, simply change the previous lines of code, and the following code remains the same. This is a much more efficient way if you are looking to develop professional applications and test with different models.

```
[ ] from langchain_core.prompts import PromptTemplate

    prompt_template = PromptTemplate.from_template("Write a poem about {topic}")

    prompt_template.invoke({"topic": "avocados"})
```

PROMPT TEMPLATES

Prompt Templates in LangChain

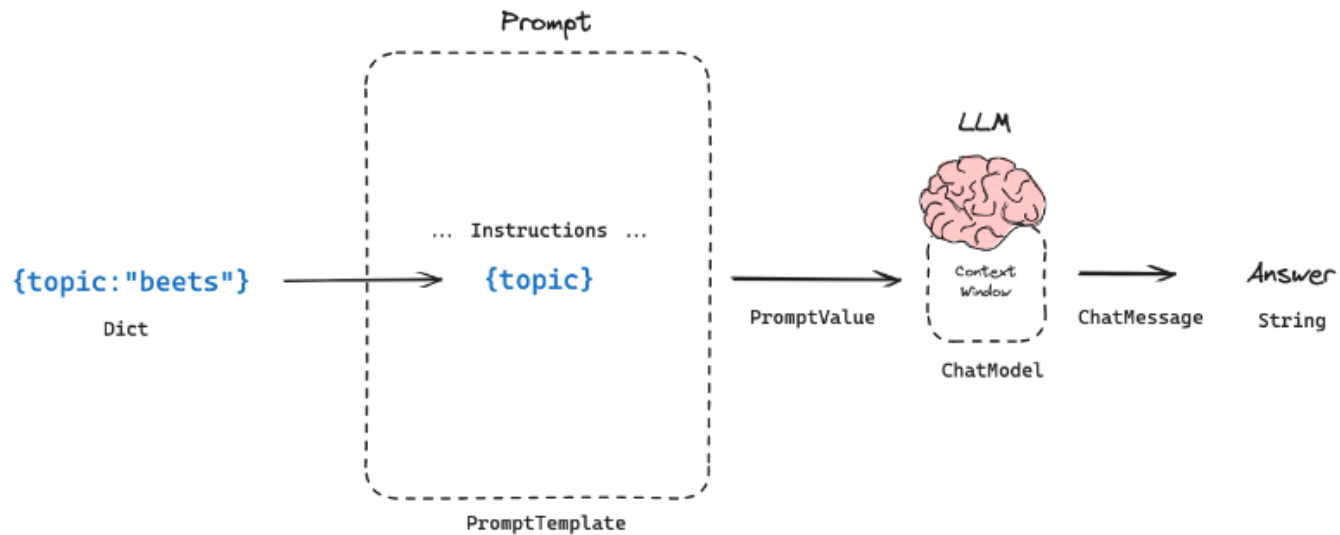
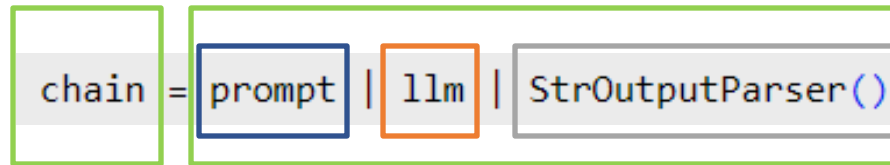


Image source: [Jacob Lee](#)

CHAINS

- Chains in LangChain allow you to connect multiple prompts to build complex applications, making it easier to create an AI application by breaking down larger problems into smaller parts.
- They work by chaining components together, where the output of one becomes the input of the next, creating a logical sequence of operations.
- In other words, they allow you to connect and "tie" different components together, enabling the creation of sequences of prompts.
- Chains are static (hard-coded), always expecting a specific type of input.



LCEL

- LangChain Expression Language (LCEL) is a declarative way of chaining library components together.
- LCEL is an abstraction of some interesting Python concepts in a format that allows a "minimalist" code layer to build chains of LangChain components.
- It allows for rapid chain development as it provides the syntax is quite practical and flexible. In addition, it allows for the incorporation of advanced features such as streaming, asynchronous, parallel execution, and much more.

LCEL

- Since version 0.2, the use of LCEL is encouraged for greater practicality, flexibility and additional features
- To see the versatility, you can compare the syntaxes:

Legacy

```
from langchain.chains import LLMChain
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI

prompt = ChatPromptTemplate.from_messages(
    [{"user", "Tell me a {adjective} joke"}],
)

chain = LLMChain(llm=ChatOpenAI(), prompt=prompt)

chain.invoke({"adjective": "funny"})
```

With LCEL

```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI

prompt = ChatPromptTemplate.from_messages(
    [{"user", "Tell me a {adjective} joke"}],
)

chain = prompt | ChatOpenAI() | StrOutputParser()

chain.invoke({"adjective": "funny"})
```

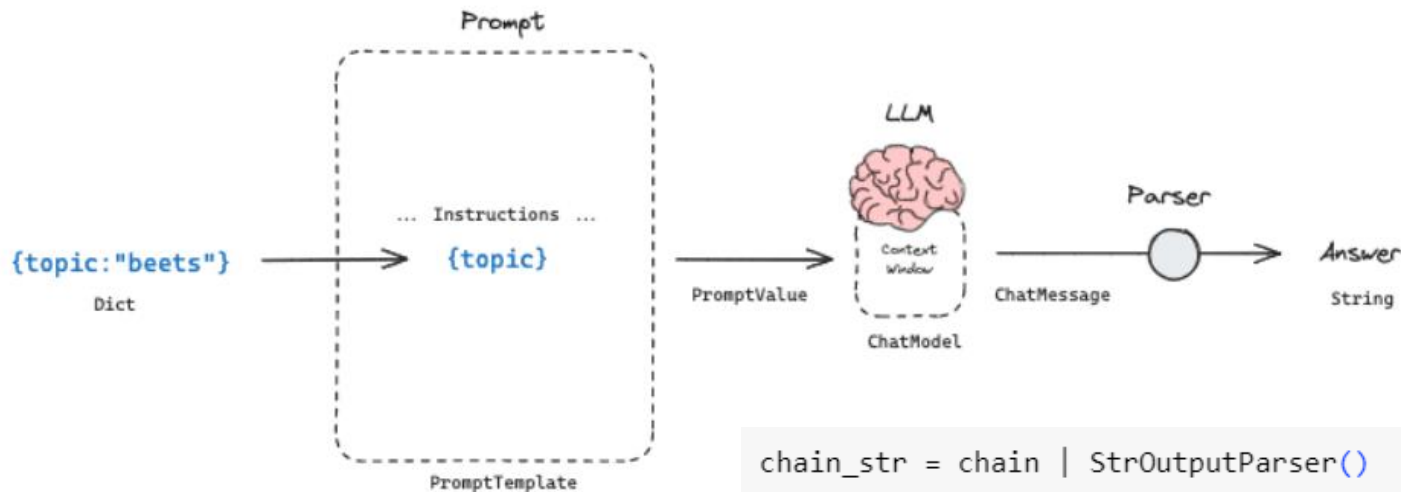
OUTPUT PARSER

- LangChain has a component called Output Parser. As the name suggests, it is responsible for processing the output of a model into a more accessible or suitable format for our purpose.
- This is very useful when you are using LLMs to generate any form of structured data.

For example, the StrOutputParser: this parser just converts the output of a language model into a string. If the model is loaded by the LLM component (and therefore produces a string), it just passes that string. If the output is a ChatModel (and therefore produces a message), it passes the .content attribute of the message.

CHAINS / OUTPUT PARSER

Extending a chain using StrOutputParser



```
chain_str = chain | StrOutputParser()
```

which is equivalent to:

```
# chain_str = prompt | llm | StrOutputParser()
```

Image source: [Jacob Lee](#)

USING LANGCHAIN LOCALLY

Set up your local environment

- Install Python
- Install Visual Studio Code (we recommend it because we are going to use it along the course, but you can use another IDE if you want)

USING LANGCHAIN LOCALLY

Run the pip commands to install langchain and other libraries

- To do this in VS Code, select: Terminal > New Terminal
(Ctrl + Shift + ')

Then, type the pip installation commands for the libraries we installed in Colab:

```
pip install -q transformers einops accelerate bitsandbytes
```

```
pip install -q langchain langchain_community [...]
```

Note: for pytorch, we recommend using this command to avoid incompatibility :

```
pip install torch==2.3.1 torchvision torchaudio --index-url  
https://download.pytorch.org/whl/test/cu121
```

USING LANGCHAIN LOCALLY – COMMON PROBLEMS

If you get a message 'pip' is not recognized (on Windows)

```
PS C:\Users\gabriel> pip --version
pip : O termo 'pip' não é reconhecido como nome de cmdlet, função, arquivo de script ou programa operável. Verifique a grafia do nome ou, se um caminho tiver sido incluído, veja se o caminho está correto e tente novamente.
No linha:1 caractere:1
+ pip --version
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (pip:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException
```

Solution:

- Search for the Python version (Start Menu), right-click > Open file location
- Then, select Python 3.12 (or the version you are using) and again choose Open file location
- After that, open the Scripts folder, and with the folder open, copy its path. Example: C:\Users<your_username>\AppData\Local\Programs\Python\Python312\Scripts
- Once copied, go to "This PC" and in an empty space, right-click and select Properties
- Then, go to Advanced system settings > Environment Variables
- Under "User variables," open the Path folder
- Click New and paste the copied path
- After that, click OK on everything and close the windows.

USING LANGCHAIN LOCALLY – COMMON PROBLEMS

If VS code cannot find the installed library

- Select *View > Command Pallete*
(or press F1, or Ctrl + Shift + P)
- Type: Python: Select Interpreter
And select *Enter*

IMPLEMENTATION ON LOCAL MACHINE - API KEYS

Setting API Keys

- Once the necessary libraries have been installed, we can create a new project where we will place the files.
- To load the tokens, instead of `os.environ` we will use a more practical method (and recommended for this case).
- We will use the **dotenv** library, which simplifies the management of environment variables by storing them in a **.env** file.
- This file contains key-value pairs that represent sensitive and environment-specific settings, such as passwords and tokens.
- The *python-dotenv* library reads the contents of the file and loads these variables into the application's execution environment.
- This improves security, makes configuration easier, and offers flexibility across different development environments. This means that configuration changes can be made without altering the source code, which simplifies local development.

IMPLEMENTATION ON LOCAL MACHINE - API KEYS

Setting API Keys

To perform this method, follow the steps:

- First, install the dotenv on terminal => `pip install python-dotenv`
- Inside the.py file, import => `from dotenv import load_dotenv`
- Create the .env file and leave it in the project root directory.
- In this .env file, you will place all the Keys that we have used so far (and others that will be used in the future), in the format below, with 1 key per line

```
HUGGINGFACE_API_KEY=#####
```

```
OPENAI_API_KEY=#####
```

- Finally, call the `load_dotenv()` method in the code (preferably in the first line right after the imports).
- Then, the Keys will be loaded into your program.

IMPLEMENTATION ON LOCAL MACHINE

To run locally, we will test a proprietary solution via API and an open source solution that can run 100% locally for free.

For proprietary models

- We'll be using ChatGPT. It's especially useful when you don't have the hardware to run your models locally, or you don't want to worry about it.
- So, if you want to run your projects on your local machine, you can use Open AI's ChatGPT integration (or another service of your choice that connects via API, like Google's Gemini or Anthropic's Claude).

For open source models in local environment

- We will use **Ollama**. If you are looking for a free solution to run open source models locally, we recommend Ollama, highly recommended due to its higher performance and its compatibility with LangChain.
- With Ollama we can run LLMs more practically with CPU, so it is not necessary to have a GPU (although it will be much faster to process)

IMPLEMENTATION ON LOCAL MACHINE

Using ChatGPT models (or other proprietary models)

You must first install it using the pip command in the terminal:

```
pip install -qU langchain-openai
```

As for the code, it's the same as the one on Colab.

Other proprietary models or via API

If you want to use other services that provide LLMs, the logic is the same for the other models (if you have direct integration with LangChain)

RUNNING LLMs LOCALLY WITH OLLAMA

- **Ollama** is an open source tool that allows you to run, create and share LLMs directly on your own computer.
- The tool is compatible with operating systems such as macOS, Linux and Windows.
- Its installation is easy and can be done directly from the official website or through Docker images.
- Ollama can be integrated with LangChain. At first, we will use it through the Ollama interface itself. And then we will connect it to LangChain.



ollama.com

RUNNING LLMs LOCALLY WITH OLLAMA

Ollama Installation

- 1) Access the website <https://ollama.com/>
 - Click the download button and select your operating system
- 2) Installation process - no configuration required, just click to continue
- 3) To run and download the model, execute the following command:
`ollama run phi3`

List of models available for download via Ollama:

<https://ollama.com/library>

RUNNING LLMs LOCALLY WITH OLLAMA

Tests on Ollama

- Type something like “hello, how are you?” to run your first example.
- The result you are seeing is generated 100% locally, meaning it could be done without internet access.

More options

- Type ``/?`` to show available commands
- Examples:
 - `/show info`
 - `/show template`

USING OLLAMA WITH LANGCHAIN

- 1) Package installation: `pip install -qU langchain-ollama`
- 2) Download the desired model using the command below, which you must also run in the terminal:

`ollama pull <nome do modelo>`

Example: `ollama pull llama3`

Note: if you have previously downloaded and run the command `ollama run <model>` for this model then you don't need to run the pull command now, as the model is saved locally.

But if you are using this model for the first time on your machine (or if you get any error while loading) use the **ollama pull** command to pull the model from the repository.

To list all downloaded models: `ollama list`

They are stored in:

- Windows: `C:\Users\<name>\.ollama\models`
- MacOS: `~/ .ollama/models`
- Linux: `/usr/share/ollama/.ollama/models`

Check the updated Ollama documentation if you cannot find it in these locations.

USING OLLAMA WITH LANGCHAIN

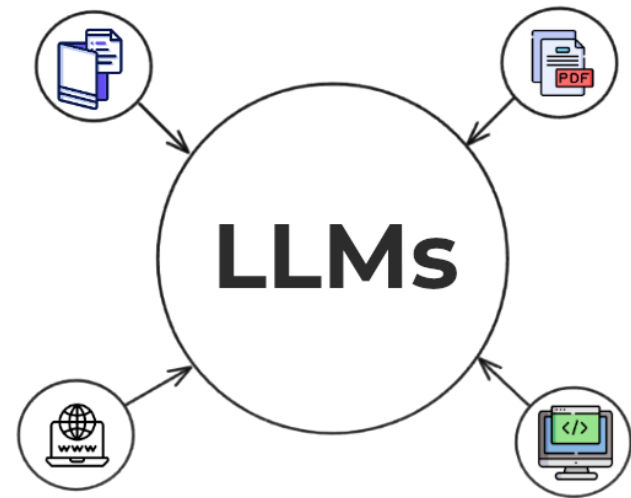
- After configuring and downloading the model, simply import the method `ChatOllama()`
- Here you must pass the model name. The other parameters are optional.
- They are basically the same as the other classes with “Chat” in the name that we have already explored previously. But if you want to see more details about the other parameters, go to:

https://api.python.langchain.com/en/latest/chat_models/langchain_ollama.chat_models.ChatOllama.html

– *check the .py file*

RAG

- **Retrieval-Augmented Generation** (RAG) is a technique that combines language models with information retrieval mechanisms to improve text generation.
- Instead of relying only on data learned by the model, RAG retrieves relevant information from external sources, such as documents or databases, to provide more accurate and reliable answers.



MAIN LLMs CHALLENGES THAT RAG HELPS TO SOLVE

- **Domain knowledge** - LLMs may not have specific information for a specific area, making their answers less accurate.
 - Solution: RAG enables the integration of domain-specific data from external sources, improving the accuracy and relevance of responses.
- **Hallucinations** - LLMs sometimes generate factually incorrect or incoherent responses, known as "hallucinations".
 - Solution: RAG uses retrieval systems to access verified information from external sources in real time, reducing the likelihood of hallucinations and increasing the reliability of responses.
- **Training data cut off** - LLMs have knowledge limited to the period in which they were trained, ignoring recent events or information.
 - Solution: RAG can search for updated information in databases or on the internet, ensuring that the answers are always up-to-date and relevant.

KEY ADVANTAGES OF RAG

- Access to up-to-date knowledge
- Reduction in hallucinations
- Integration of domain knowledge
- Versatility and more relevant answers
- Context-aware answers
- Dynamic retrieval capability
- Improved linguistic comprehension

RAG – APPLICATION EXAMPLES

- Customer Service
- Academic Research
- Legal Assistants
- Medicine and Diagnostics
- Product Development

RAG – BASIC PIPELINE

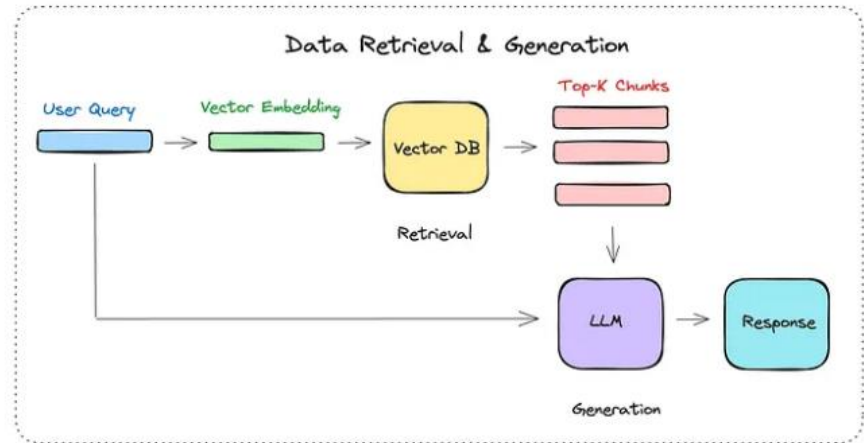
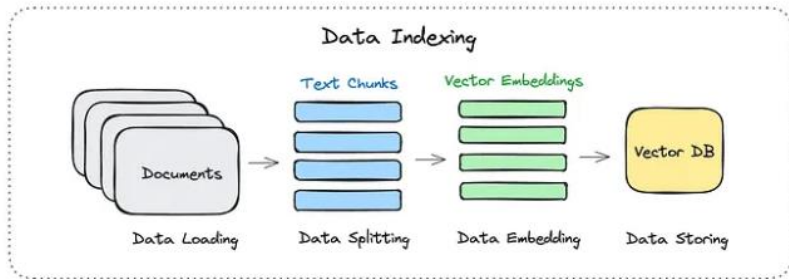


Image source: [DrJulija's Notebook](#)

RAG - CONCEPTS

A conventional RAG application can be divided into two processes:

- **Indexing:**
 - It consists of a pipeline for ingesting data from a source and indexing it.
 - The typical method is to convert all private data into embeddings stored in a vector database.
- **Retrieval and generation:**
 - It is the chain itself, which processes the user request and retrieves the relevant data from the index, then passes it to the model to use as context.
- LangChain has several components designed to help build Q&A applications and RAG applications in general.

RAG - PIPELINE

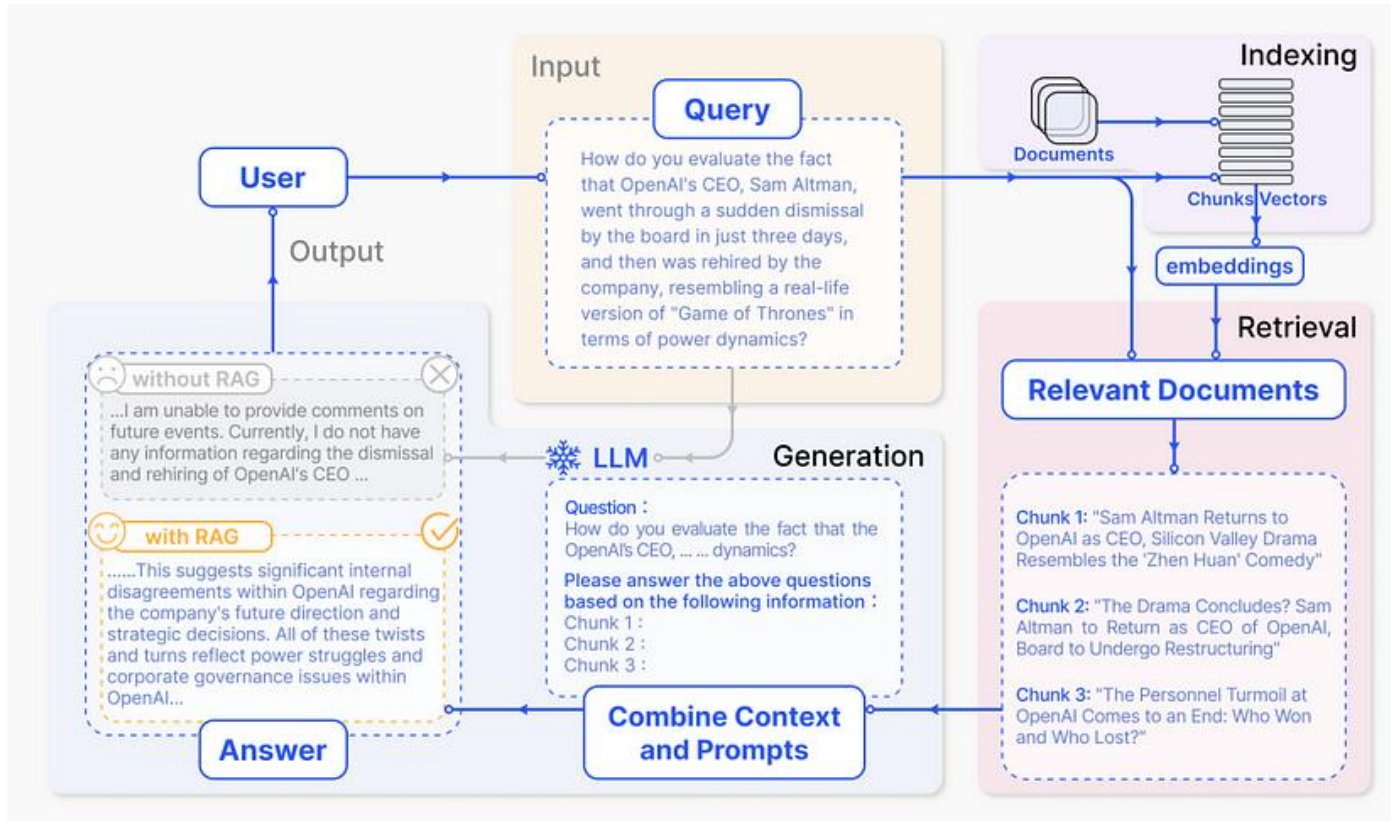


Image source: paper "RAG for Large Language Models: A Survey"

RAG - PIPELINE

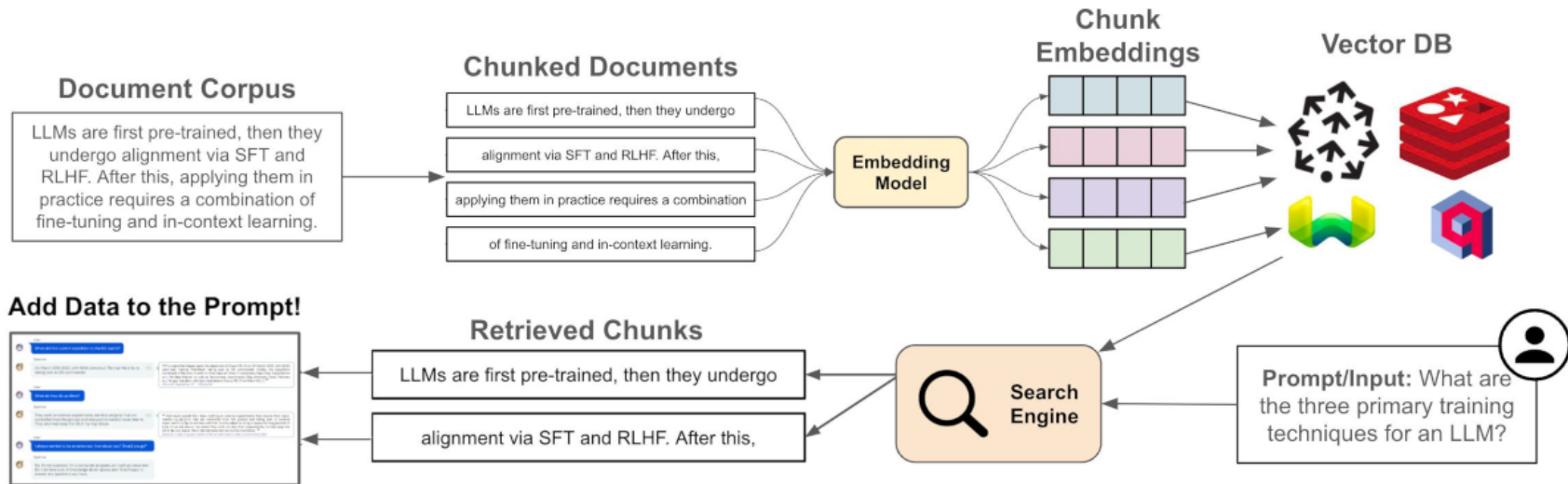


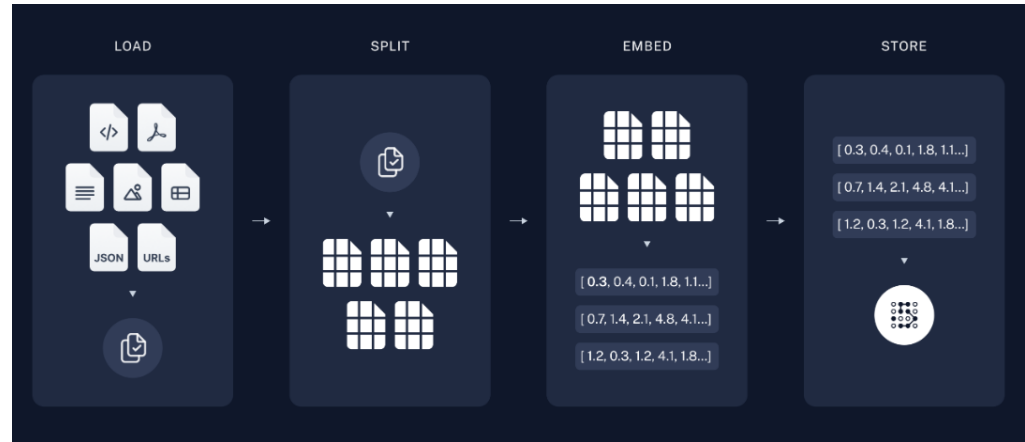
Image Source: A Practitioners Guide to Retrieval Augmented Generation (RAG)

RAG - INDEXING

1) Load: First, we need to load our data. In LangChain we have several functions (called Document Loaders) that make this process easier.

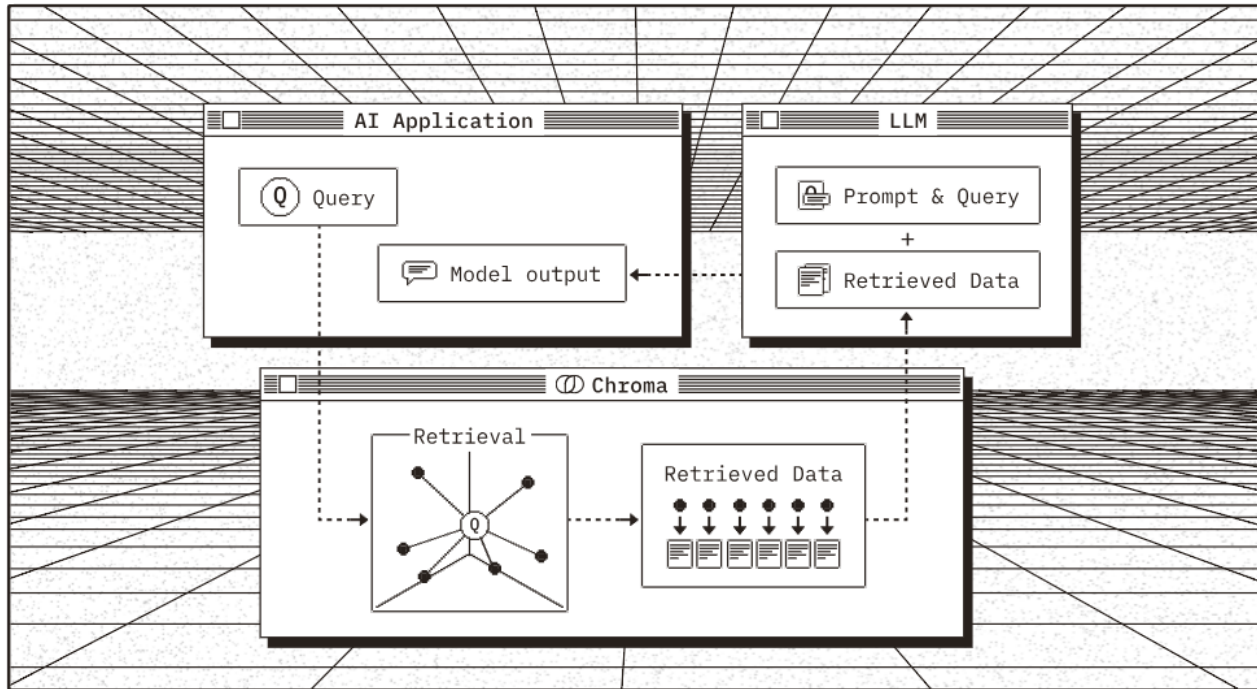
2) Split: Text dividers break large documents into smaller parts, making them easier to index and to pass through to the model, since large parts are more difficult to search (for example, entire books or documents with hundreds of pages) and therefore do not fit in the model context window.

3) Embed and Store: We need a place to store and index these chunks so that they can be searched later. After embedding, the data can then be stored in a vector database (Vector Store) for later retrieval.



VECTOR STORE

Summary of the process: Convert a query into embeddings so that they can later be retrieved and provided as context along with the question, which will be passed as input to the LLM



Source: Chroma DB website (<https://www.trychroma.com>)

VECTOR STORE

There are several tools and services for vector stores and semantic databases

For a complete comparison, visit:

<https://superlinked.com/vector-db-comparison?via=topaitools>

Access the Langchain documentation to see which ones have integration

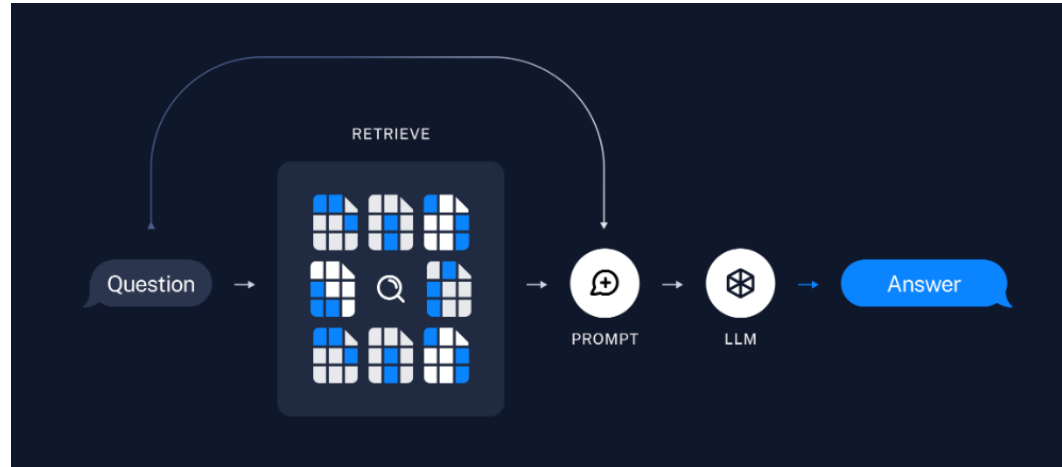
<https://python.langchain.com/v0.2/docs/integrations/vectorstores/>

RAG – RETRIEVAL AND GENERATION

4) Retrieval: Given a user's input, the relevant partitions are retrieved from storage using a retriever.

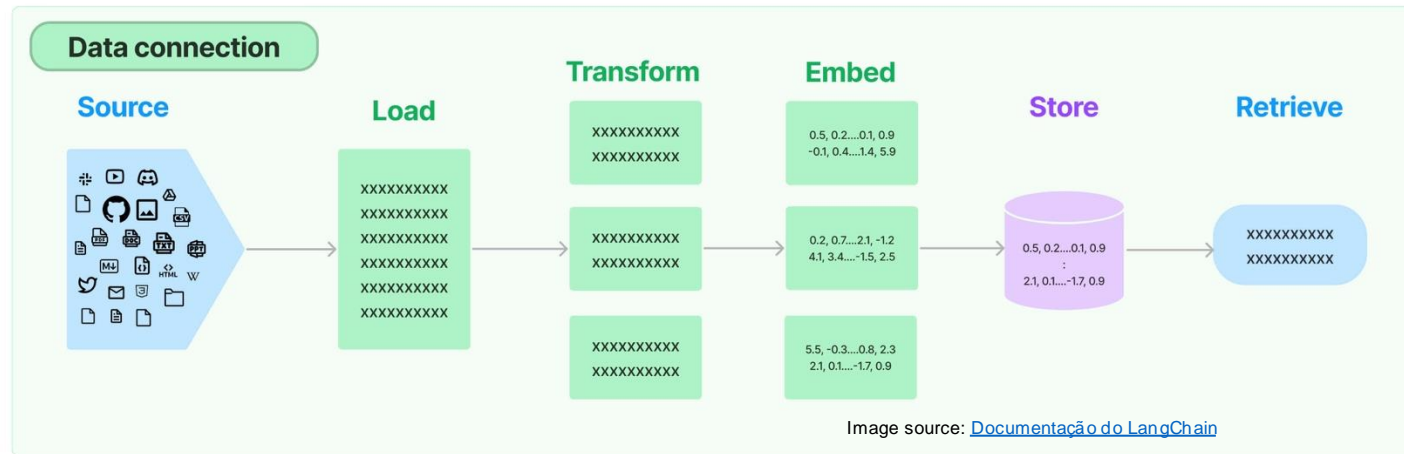
5) Generation: the model then produces a response using a prompt that includes the question and the retrieved data.

- The model will receive as input the user's question + context



LANGCHAIN – DATA CONNECTION AND RETRIEVAL

- External data is retrieved and then passed to the LLM when doing the generation step.
- This means that it is possible to use LLMs to process our documents or other resources such as web pages.
- LangChain provides all the building blocks for RAG applications - from simple to complex.



LANGCHAIN – TERMS AND CLASSES FOR RETRIEVAL

- **Document Loaders** - Document loaders load documents from a variety of different sources. LangChain offers hundreds of document loaders and integrations with other major providers such as AirByte and Unstructured.
- **Text Splitting** - An essential part of information retrieval is searching for only the relevant parts of documents. LangChain offers several transformation algorithms to split large documents into smaller parts, optimized for specific document types.
- **Text Embedding Models** - Creating embeddings for documents captures the semantic meaning of text, allowing you to quickly find similar parts. LangChain integrates several embedding providers, providing a standard interface to easily switch between models.
- **Vector Stores** - With the rise of embeddings, there is a need for databases to store and search them efficiently. LangChain integrates over 50 different services, allowing you to choose the one that best suits your needs and providing a standard interface to switch between them.
- **Retrievers** - Once the data is in the database, it needs to be retrieved. LangChain supports a variety of retrieval algorithms, offering simple semantic search methods and a collection of additional algorithms to increase performance.
- **Indexing** - LangChain's Indexing API synchronizes your data from any source into a vector store, helping to avoid content duplication, rewriting of unchanged content, and recalculation of embeddings on unchanged content.

RAG – PRACTICAL EXAMPLE

Diagram of a RAG pipeline for Q&A application

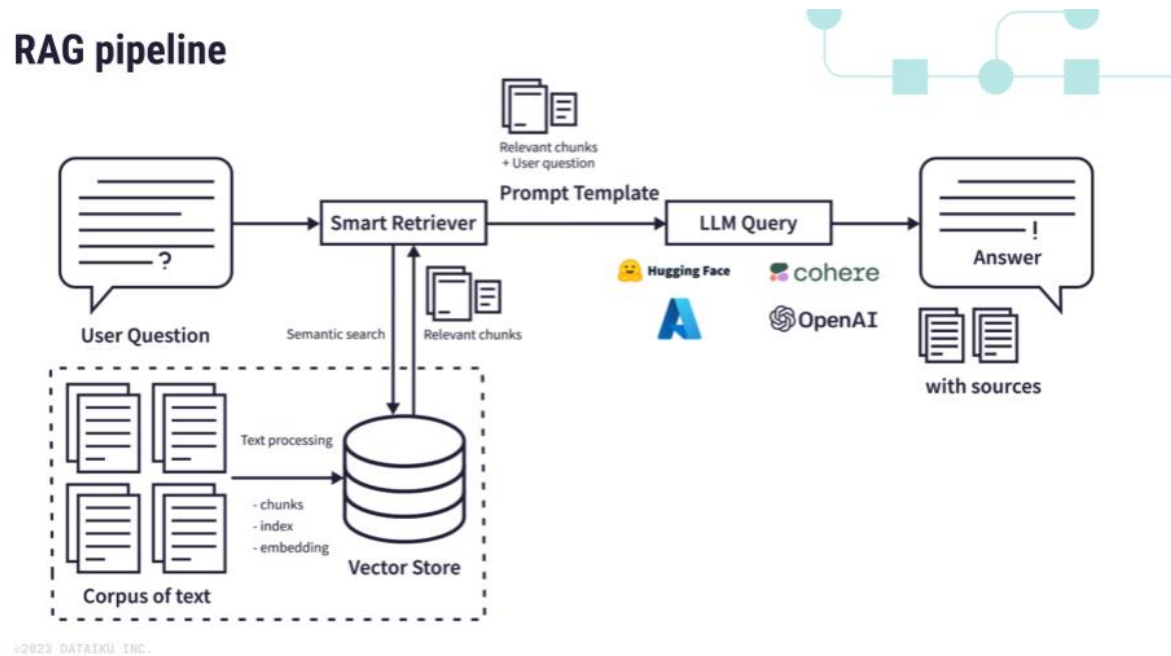


Image source: [data iku](https://dataiku.com)

RAG – PRACTICAL EXAMPLE

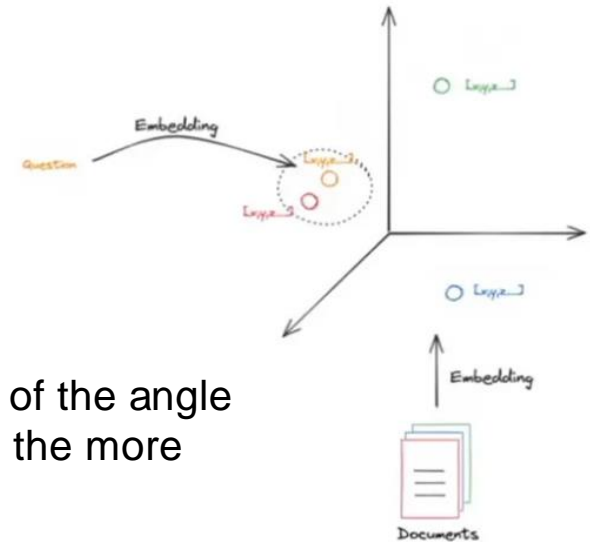
Diagram of a RAG pipeline for Q&A application

1. Gather documents to supplement an LLM's knowledge base on a given subject area, such as internal policies, financial documents, technical documentation, or research papers.
2. Use a vector store (such as FAISS, Pinecone, or ChromaDB) to break down textual data into smaller chunks and vectorize them, storing the results in a knowledge base optimized for semantic search.
3. When a user asks a question, the intelligent retrieval system uses the vector store to supplement the question with relevant information.
4. The augmented prompt is used to query the LLM, providing more accurate answers along with their sources.

RAG – RETRIEVAL

Similarity Search Examples

- It consists of finding documents or information that are similar to a query based on their numerical representations.
- Similarity Search: Compares the numerical vectors (embeddings) of the query and the documents to find the most similar ones via KNN.
- Cosine Similarity: a method that measures the cosine of the angle between two vectors. The higher the cosine similarity, the more similar the vectors are.
- It's like to search for a book in a library by describing its contents. The system finds books with contents similar to its description by comparing the numerical representations of the books' contents.



AGENTS AND TOOLS

- LLMs have difficulty with certain tasks that involve logic, mathematical calculations or research.
- The definition of LLM agents is quite broad: they are all systems that use large language models as an engine and can perform actions in their environment based on observations.
- The agent uses a reasoning engine to determine what actions to take to obtain a result – e.g. an agent integrated with the Google search engine and Wikipedia.

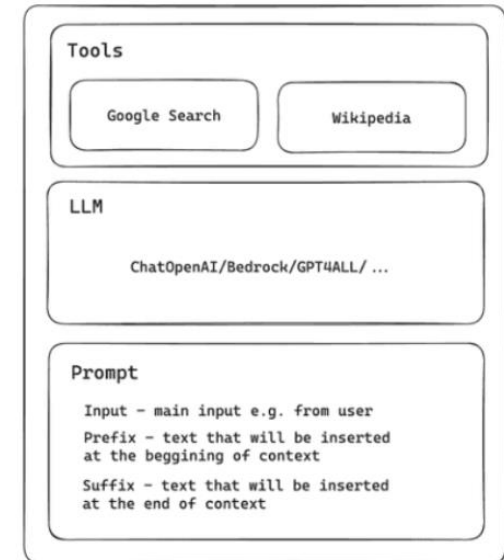


Image source: [Bright Inventions](#)

AGENTS AND TOOLS

- **Tools** are the main components of agents that perform individual tasks.
- Tools are basically just methods/classes that the agent has access to that can do things like: perform a web search, interact with a stock market index through an API, update a Google Calendar event, or run a query on a database.
- A collection of Tools in LangChain is called **Toolkit**.

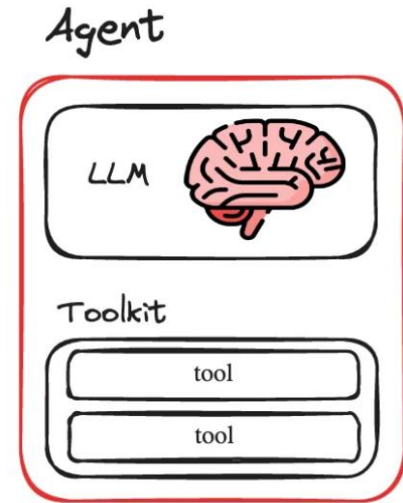


Image source:
[Sami Maameri](#)
[- Towards Data Science](#)

AGENTS VS. CHAINS

Agents



- Agents have a higher level of flexibility, real autonomy and reasoning ability.
- Unlike chains, they can select by themselves the tools as needed, in a sequence they consider appropriate.

Chains

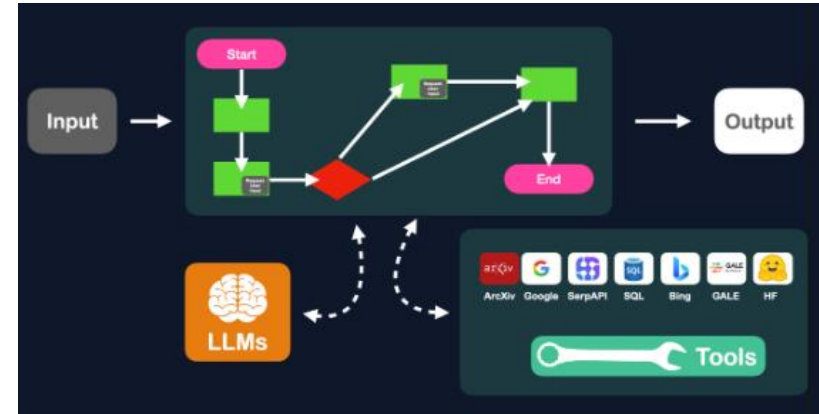


Image source: [Cobus Greyling](#)

- We can have direct control over the task or direction of the conversation.
- Valuable especially in situations where it is crucial to have detailed control over the flow of the task or conversation.

ADVANTAGES OF AGENTS

- Flexibility
- Dynamic reasoning
- Simplified maintenance
- Integration capability
- Action production
- Suitability for complex tasks
- Results feedback

AGENTS – USE CASES

- Business
- Healthcare
- Customer service
- Content generation and management
- Emerging trends and Future directions

AGENTS – EXPLORING THE POTENTIAL

An overview of the current (and future) state of agents and their potential

- You can find a good review of the general landscape of agents in the article **The Rise and Potential of Large Language Model Based Agents: A Survey**, by Xi et al., 2023.

< Papers arxiv:2309.07864

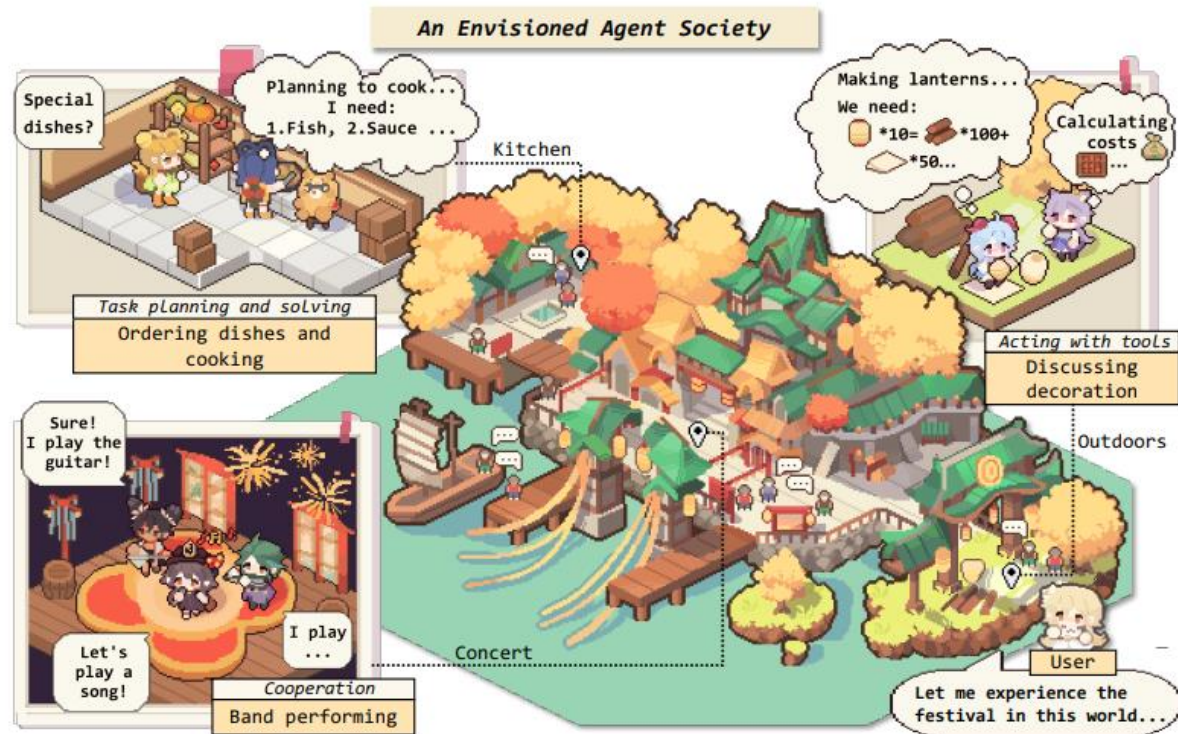
The Rise and Potential of Large Language Model Based Agents: A Survey

Published on Sep 14, 2023

Authors: Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He,  Yixen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Qin Liu, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou + 8 authors

<https://arxiv.org/abs/2309.07864>

AGENTS – EXPLORING THE POTENTIAL



Source: [The Rise and Potential of Large Language Model Based Agents: A Survey](#)

CHAIN-OF-THOUGHT

- One way to solve the problem of complex reasoning in LLM is to use Chain-Of-Thought prompting.
- Complex reasoning can be challenging even for today's largest LLMs, especially tasks that include multiple steps.
- To achieve their task, agents may use multiple iterations of the prompting cycle:

Observation \Rightarrow Thought \Rightarrow Action

Standard Prompting	Chain-of-Thought Prompting
<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p>Model Output</p> <p>A: The answer is 27. ❌</p>	<p>Model Output</p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅</p>

For more in-depth details about the theory, check the paper **Chain-of-Thought Prompting Elicits Reasoning in Large Language Models** (<https://arxiv.org/abs/2201.11903>)

ReAct (REASONING AND ACTING)

- React is an approach to building agents. It is defined based on the concatenation of two words “**Reasoning**” and “**Acting**”.
- In the prompt, we describe the model, what tools it can use, and ask it to think "step by step" (also called *Chain-of-Thought* behavior) to plan and execute his next actions to reach the final answer.

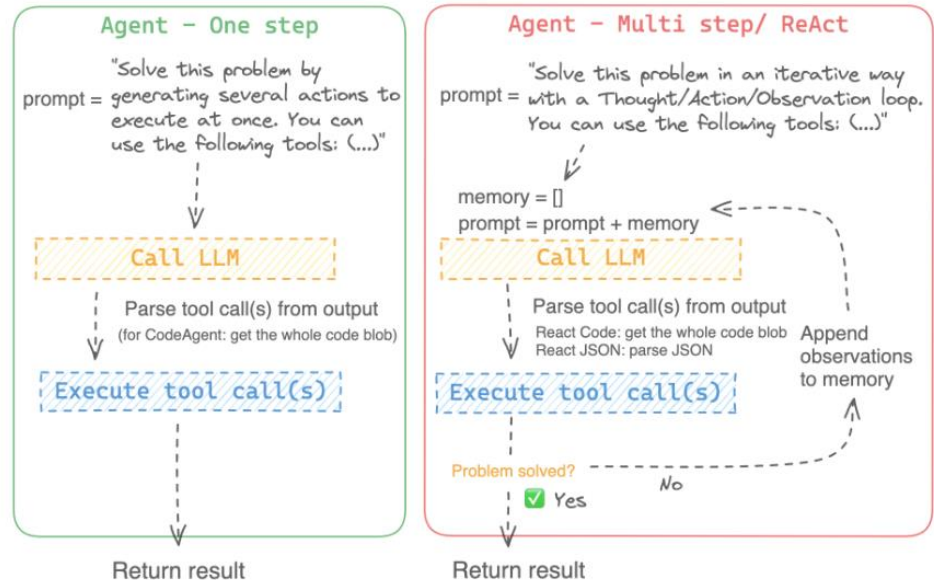


Image source: [Aymeric Roucher, Hugging Face](#)

AGENTS – ReAct AND CHAIN-OF-THOUGHT

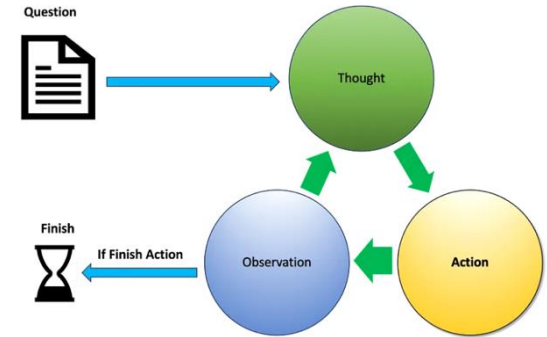
- ReAct enables LLMs to perform task-specific reasoning and actions.
- Combines chain of thought with action planning, improving performance in various tasks and overcoming problems such as hallucinations.
- To solve complex questions, LLM adopts an iterative process.
- First, LLM generates a thought about the problem and identifies an action to take. Actions can include API calls, such as fetching data from Wikipedia.
- The LLM observes the results of actions and, if necessary, generates new thoughts and actions until it finds the answer.

AGENTS AND ReAct

In the ReAct framework, the LLM can choose from a limited number of actions that are defined by a set of instructions that are appended to the text of the LLM question prompt.

For example, the ReAct paper covers three action spaces. It designs a simple Wikipedia Web API with three types of actions to support interactive information retrieval.

- *search[entity]* which returns the first 5 sentences from the wiki page of the matching entity, if it exists, or else suggests the top 5 similar entities from the Wikipedia search engine
- *lookup[string]* which would return the next sentence on the page that contains the string, simulating the search (Ctrl+F) functionality in the browser.
- *finish[answer]* which would end the current task with the answer



More details and in-depth theory can be found in the paper “ReAct: Synergizing Reasoning and Acting in Language Models”

<https://arxiv.org/pdf/2210.03629>

AGENTS IN LANGCHAIN – POSSIBLE LIMITATIONS

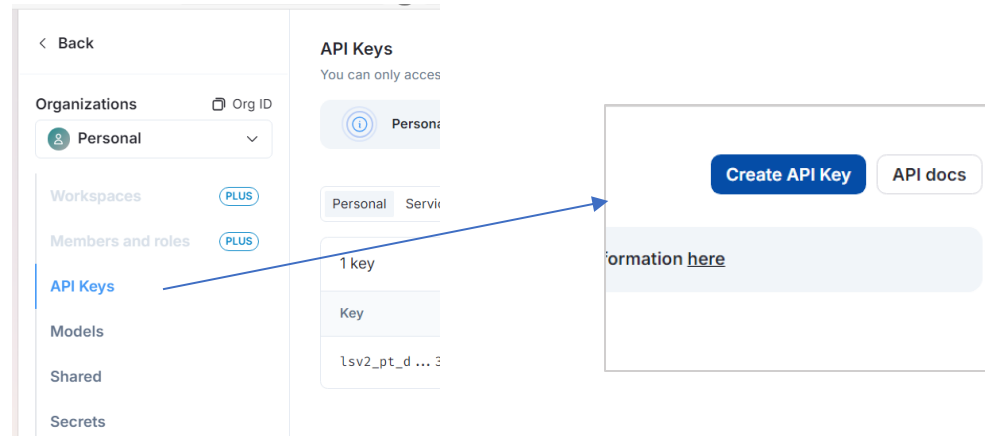
- LangChain is a very rich and complete framework, but currently perhaps its weakest point is the lack of compatibility with open source models for creating AI agents.
- The documentation is often outdated and lacking in information, making it difficult to solve problems. The AI community points out these limitations, especially regarding the clarity and lack of update of the information.
- Agents and tools have great potential for use, allowing complex interactions between different tools and LLMs to be orchestrated. However, because the documentation on how they work is still vague and often impossible to produce (leading to unexpected and undesirable behaviors), it makes it difficult to understand and implement effectively.
- This will probably be resolved in the near future. In any case, if the same behaviors persist, it is worth considering testing another framework for AI agents (such as **LlamaIndex** or **Crew AI**), or also creating a solution manually.

LANGSMITH

- In addition to providing a hub containing thousands of community prompts, LangSmith offers a tracing tool, which is used for debugging.
- It contains complete information about all inputs and outputs of each step of the application.
- As these applications become increasingly complex, it becomes crucial to be able to inspect exactly what is happening inside your chain or agent.

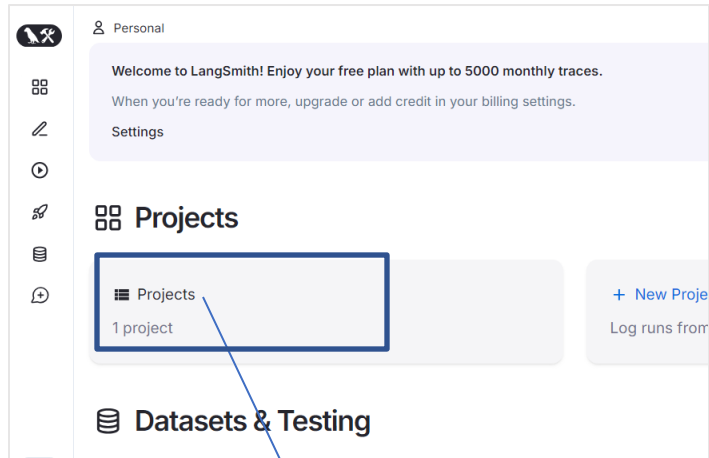


To generate a key, go to:
<https://smith.langchain.com>



LANGSMITH

To view your records, go to:



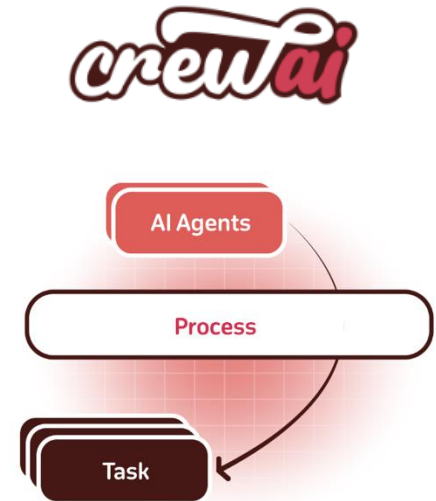
Name ↑↓	Feedback (7D)	Run Count (7D)	Error Rate (7D) ↑↓
default		16	19%

>	✓	AgentExecutor	Onde fica Linz? Qual ...	Linz é a terceira ...
>	✓	AgentExecutor	Qual é o valor de mer...	O valor de merca...
>	✓	AgentExecutor	Qual o valor de merca...	O valor de merca...

The screenshot shows the 'AgentExecutor' trace view. It includes a 'TRACE' section with 'Collapse', 'Stats', and 'Filter' options. The 'Calls' section lists various components like 'PromptTemplate', 'ChatOpenAI', and 'tavily_search_results_json'. The 'Input' section shows the user's query: 'input: Onde fica Linz? Qual a população?'. The 'Output' section shows the response: 'Linz é a terceira maior cidade da Áustria, localizada na região da Alta Áustria, com uma população de cerca de 208.000 habitantes.' A blue arrow points from the 'AgentExecutor' row in the table above to this trace view.

SOLUTIONS FOR MORE COMPLEX APPLICATIONS WITH AGENTS

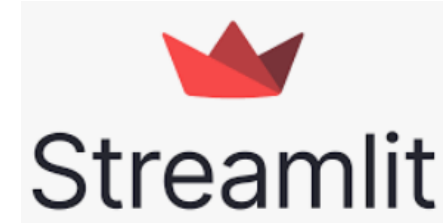
- CrewAI is a cutting-edge framework for orchestrating autonomous AI agents that perform specific roles.
- It facilitates collaborative intelligence, allowing agents to work together seamlessly and tackle complex tasks.
- Using CrewAI, you can build intelligent assistant platforms, automated customer service teams, or multi-agent research groups, where each agent takes on a role and shares goals in a cohesive system.
- In addition, CrewAI may be more interesting and complete than LangChain for this task due to its better integration with open source models, offering a robust and flexible solution for various AI applications.



crewai.com

CREATING USER INTERFACES WITH STREAMLIT

- Streamlit is an open source framework that makes it easy to build interactive machine learning or data science web applications using just a few lines of Python code.
- It enables developers to quickly and efficiently transform data scripts into web applications.
- Streamlit is known for its flexibility, simplicity, and ability to easily integrate with visualization and machine learning libraries.
- Using Streamlit, we will be able to create interactive interfaces for our LLM applications more quickly and efficiently, accelerating the development and deployment of our solutions.



USING STREAMLIT IN A LOCAL ENVIRONMENT

Setting up Streamlit

Installation

- use the command: `pip install -q streamlit`

To launch the UI

- use the command: `streamlit run <your_file>.py`


USING STREAMLIT IN COLAB


- Solutions like ngrok and localtunnel are necessary to expose local servers to public internet addresses securely and quickly by generating a Public URL for your Localhost.
- They are especially useful during web application development, allowing developers to share their work in progress with colleagues or clients without having to set up a public server.
- The concept of tunneling refers to the process of creating a secure connection that encapsulates network traffic between two points. In the context of ngrok and localtunnel, a tunnel creates a bridge between a local server and the internet, allowing external access to the local server.


For more information on how to use it, visit the [Colab of Project 2](#)

CUSTOMIZED CHATBOT WITH MEMORY AND INTERFACE


Your AI assistant 🤖

 Hi, I'm your virtual assistant! How can I help you?

 What is the largest planet in the solar system?

 The largest planet in our solar system is Jupiter! It's a gas giant, and it's truly massive, with a diameter of approximately 142,984 kilometers (88,846 miles). That's more than 11 times the diameter of the Earth! Jupiter is also the fifth planet from the Sun and is known for its distinctive banded appearance and its many moons.

 and the smaller?

 The smallest planet in our solar system is Mercury! It's a rocky, terrestrial planet with a diameter of approximately 4,879 kilometers (3,031 miles). That's about 40% the diameter of the Earth! Mercury is also the closest planet to the Sun, with an average distance of about 58 million kilometers (36 million miles). It's a very hot planet, with surface temperatures reaching up to 427°C (801°F) during the day, and dropping to -173°C (-279°F) at night.

Enter your message here...



Testing with Streamlit to generate our application's UI

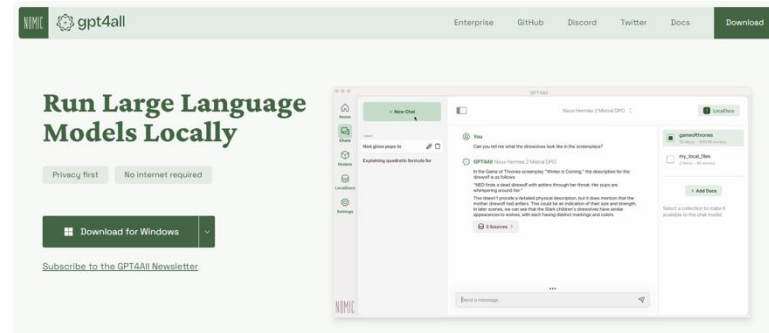
ALTERNATIVE OPEN SOURCE SOLUTIONS FOR RUNNING LLMs IN A LOCAL ENVIRONMENT

- In addition to Ollama, there are other solutions that allow for an optimized implementation of LLMs on local machines.
- These solutions have become popular for democratizing the use of large language models, allowing them to run entirely on your local machine with minimal overhead and without even requiring a GPU.
- In addition, many of these solutions already come with an user interface, which allows a more lay audience to make use of it.
- Since we focus on a more “programmatic” solution, the choice of our presented stack based on LangChain makes more sense.
- However, these are solutions that may be interesting depending on your use case, especially if the idea is to obtain fast inference with little configuration effort.

ALTERNATIVE OPEN SOURCE SOLUTIONS FOR RUNNING LLMs IN A LOCAL ENVIRONMENT

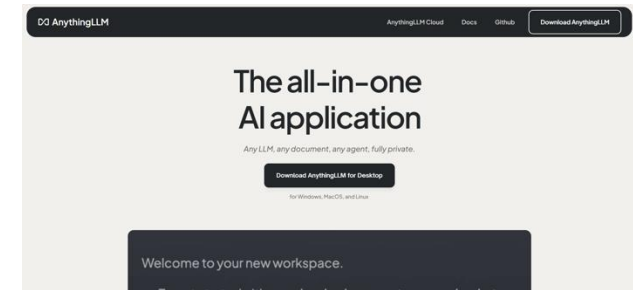
- GPT4All is designed to run on modern or relatively modern computers without needing an internet connection or even a GPU.
- The idea behind GPT4All is to provide a free and open source platform where people can run large language models on their computers.
- Its use has become popular due to its ease of use.
- It offers a ready-to-use and very intuitive interface.
- It also has integration with LangChain.

<https://gpt4all.io/index.html>



ALTERNATIVE OPEN SOURCE SOLUTIONS FOR RUNNING LLMs IN A LOCAL ENVIRONMENT

- AnythingLLM simplifies the integration of multiple AI services such as OpenAI, GPT-4, and vector databases (e.g. LangChain, Pinecone, and ChromaDB) into a cohesive package that increases productivity exponentially.
- One of its main strengths is the ability to run entirely on your local machine with minimal overhead, without requiring GPU.
- It allows you to transform any document, resource or piece of content into a context that any LLM can use as references during the chat.
- It is a practical way of implementing solutions with RAG for example, similar to the system we developed in project 3.



<https://anythingllm.com>