Photo by Biel Morro on Unsplash

Machine Learning System Design Interview Cheat Sheet-LLM's Part 4

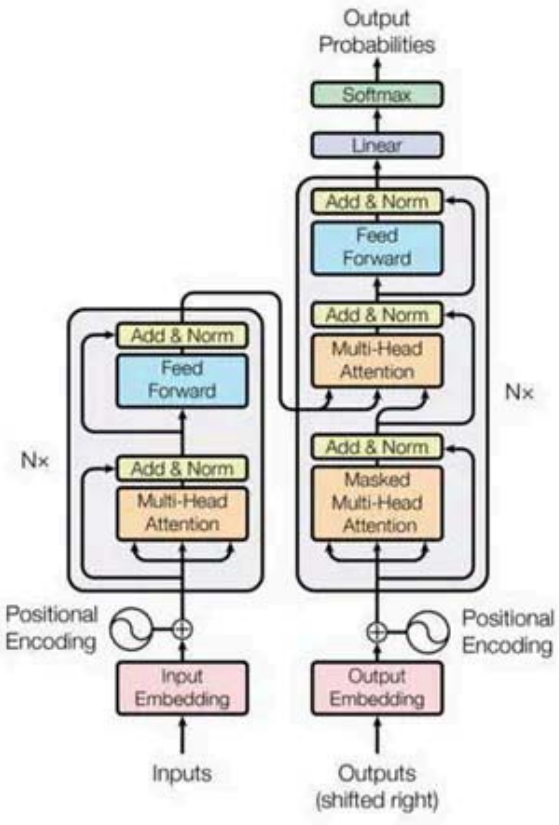
Large Language Models

Senthil E · Following
Published in Analytics Vidhya
23 min read · Apr 24, 2023

Image by the Author

Introduction:

Let's learn about the basics of the LLMs for the system design interview. Here I am covering various resources where you can learn about LLMs.



Source:Attention Is All You Need

Image by the Author

What is the Large Language Model?

From Wikipedia.org **large language model** (**LLM**) is a language model consisting of a neural network with many parameters (typically billions of weights or more), trained on large quantities of unlabelled text using self-supervised learning. LLMs emerged around 2018 and perform well at a wide variety of tasks. This has shifted

the focus of natural language processing research away from the previous paradigm of training specialized supervised models for specific tasks.

Check out the ebook from Nvidia for more info on LLMs

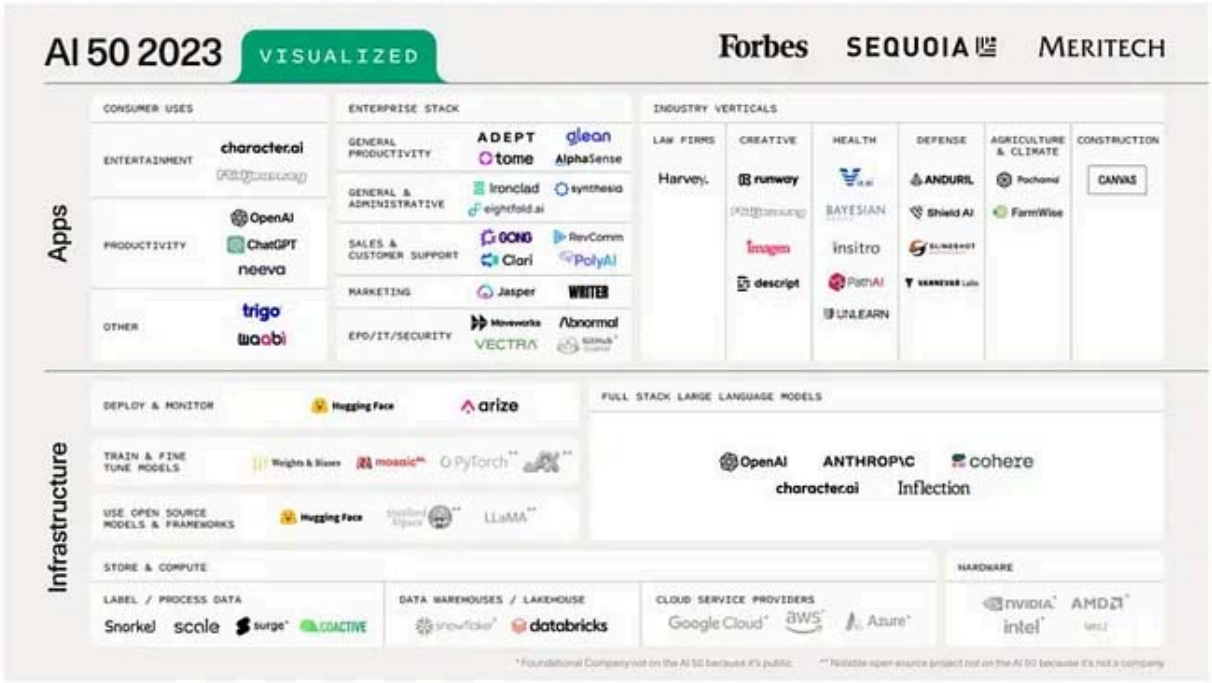
An Enterprises' Guide to Large Language Models | NVIDIA

A comprehensive background on what LLMs are, how they work, and how to evaluate them, paired with use case examples and...

www.nvidia.com

What are Foundation Models?

- Foundational models are large-scale machine learning models trained on diverse and extensive data.
- They serve as a "foundation" or "base" for various applications and tasks in natural language processing, computer vision, and more.
- These models are pre-trained on a wide range of data to learn the general language and visual representations.
- Once pre-trained, foundational models can be fine-tuned for specific tasks using smaller, task-specific datasets.
- Examples of foundational models include language models like GPT-4, Dolly 2.0, etc
- Foundational models have the potential to understand, generate, and analyze human language and images at a high level of complexity.
- Their versatility makes them suitable for a wide array of applications, from language translation and image recognition to text summarization and chatbots.
- Despite their advantages, foundational models raise ethical and technical considerations, such as biases in training data, resource requirements, and potential misuse.



SEQUOIA CAPITAL

Source-Generative AI Is Exploding. These Are The Most Important Trends You Need To Know

Blogs:

1. A Step-by-step Guide to Building Large Custom Language Models

2. Understanding Transformer

3. The Transformer Family by Lilian Weng

4. Understanding Large Language Models

Comparison between classical machine learning models Vs Large Language Models:

Image by the Author

Image by the Author

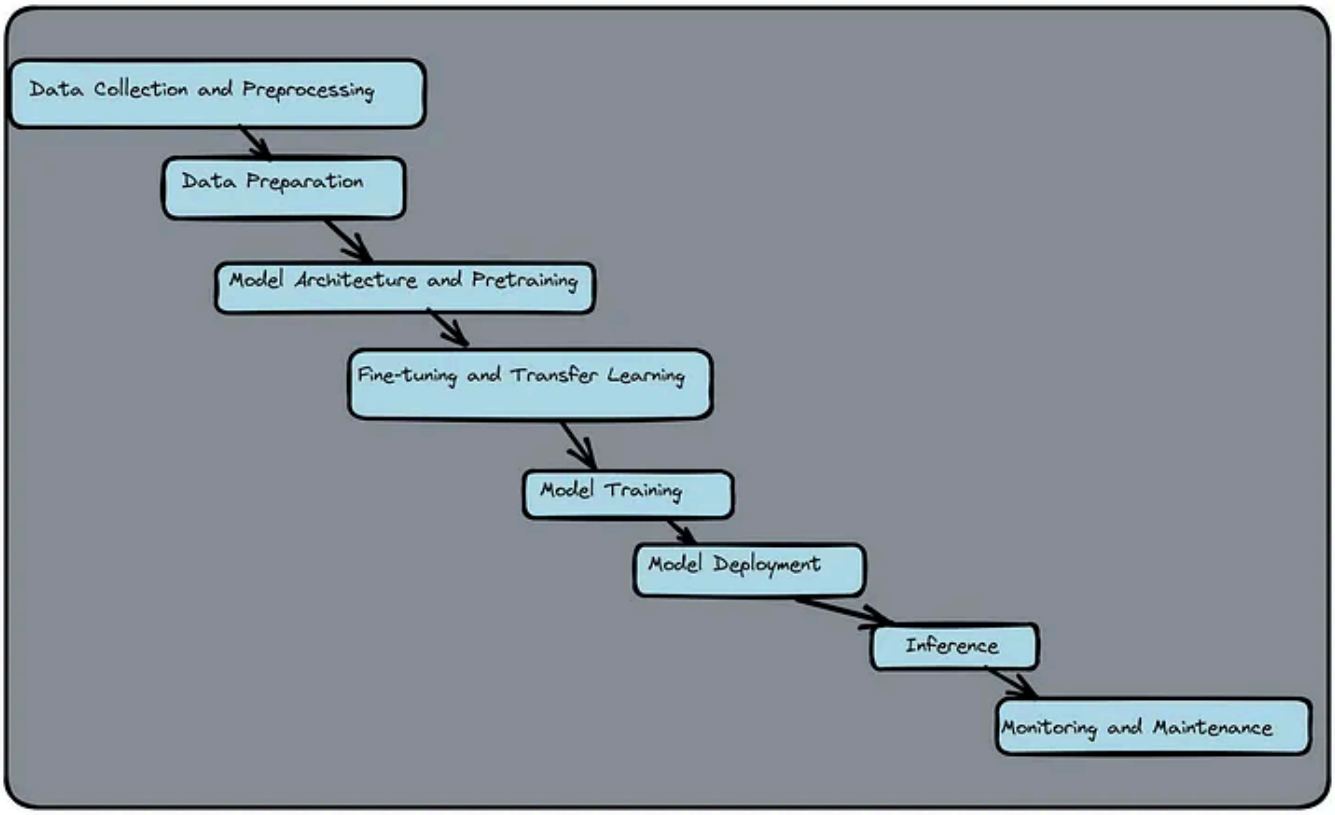


Image by the Author

Data Collection and Preprocessing

- Gather raw data (text, images, audio, etc.) from various sources.
- Perform data cleaning, preprocessing, and filtering to ensure high-quality input data.

Data Preparation

- Tokenize the input data to convert it into a format suitable for the LLM.
- Apply encoding techniques to represent tokens as numerical values.
- Organize the data into batches and apply padding, if necessary.
- Split the data into training, validation, and testing sets.

Model Architecture and Pretraining

- Choose an appropriate transformer-based architecture (e.g., GPT, BERT, RoBERTa).
- Pretrain the model on a large corpus of text to learn general language patterns.

Fine-tuning and Transfer Learning

- Fine-tune the pretrained model on the specific task or domain using the prepared data.

- Modify the model architecture if needed, such as by adding task-specific output layers.

Model Training

- Choose an appropriate loss function and optimizer for the task.
- Set and tune hyperparameters, such as learning rate, batch size, and the number of epochs.
- Implement a training loop to update model weights iteratively using backpropagation.
- Monitor and save model checkpoints to track progress and prevent overfitting.

Model Evaluation

- Evaluate the trained model on the validation and test sets using relevant metrics (e.g., accuracy, F1-score, perplexity).
- Perform error analysis to understand model weaknesses and areas for improvement.
- Visualize model performance using plots and charts, such as learning curves or confusion matrices.

Model Deployment

- Optimize and compress the model for efficient deployment and low-latency inference.
- Containerize the model using technologies like Docker for easy deployment and scaling.
- Deploy the model on a cloud or on-premises infrastructure, depending on the requirements.

Inference

- Preprocess input data for real-time or batch inference using the deployed model.
- Generate predictions or embeddings by passing input data through the LLM.
- Postprocess the model's output, such as by decoding tokens back into human-readable text.

Monitoring and Maintenance

- Continuously monitor the model's performance, system health, and resource usage.
- Update the model with new data or retrain it if performance degrades or requirements change.
- Ensure system security, data privacy, and compliance with relevant regulations.

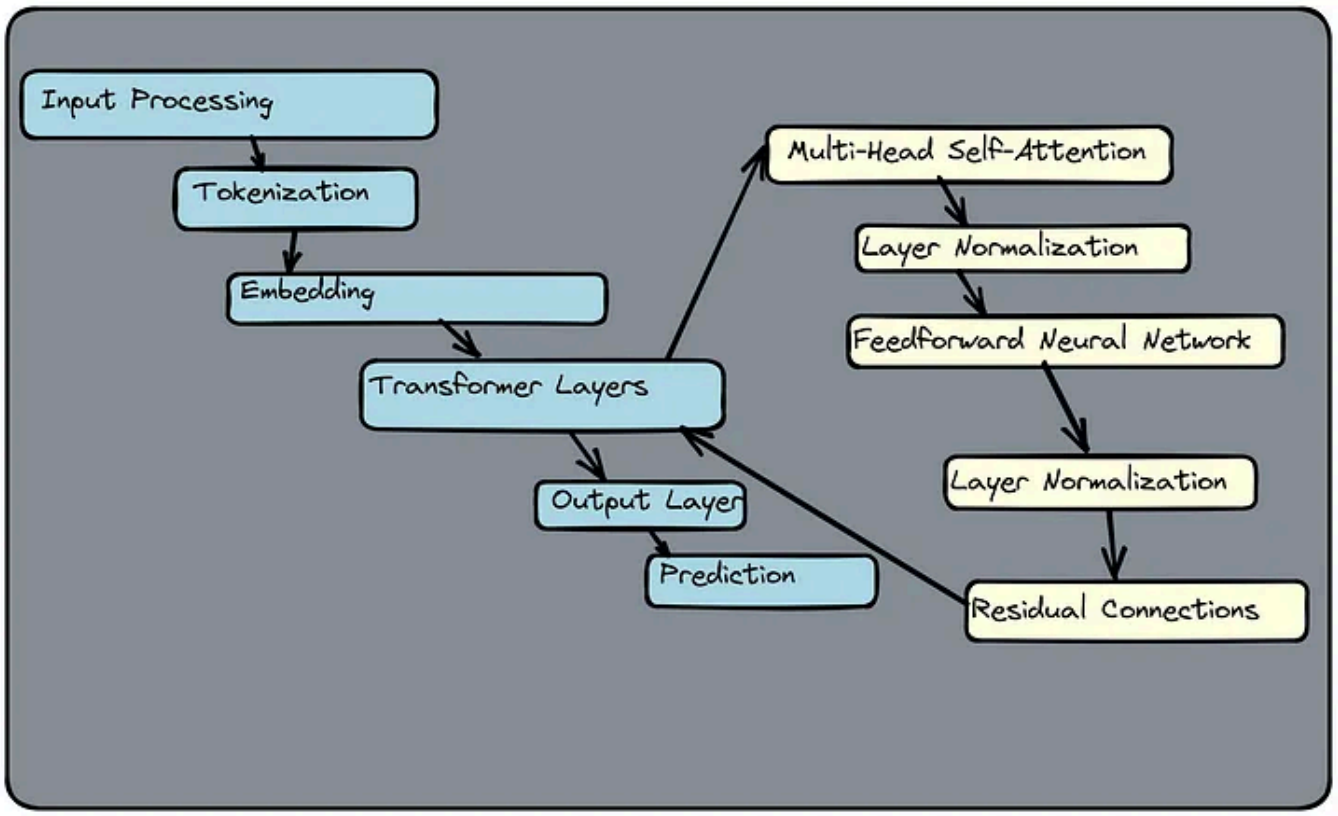


Image by the Author

Input Processing:

- Receive input text sequence
- Preprocess input (e.g., remove special characters, lowercasing)

Tokenization:

- Tokenize input sequence into individual tokens (words, subwords, or characters)

Embedding:

- Convert tokens into continuous vector representations (embeddings)
- Add positional encoding to embeddings

Transformer Layers

- For each layer:

- **a. Multi-Head Self-Attention** — Compute attention scores for each token based on its relationship with all other tokens in the sequence — Repeat the process in parallel multiple times (heads)
- **b. Layer Normalization** — Apply normalization after the self-attention layer
- **c. Feedforward Neural Network** — Pass the outputs of the multi-head attention through a feedforward network
- **d. Layer Normalization** — Apply normalization after the feedforward layer e. Residual Connections — Combine the original input with the output of the self-attention and feedforward layers

Output Layer

- Pass the final transformer layer's output through an output layer to produce logits or probabilities for each token in the vocabulary

Prediction

- Select the token with the highest probability for the task (e.g., next word or token in the sequence, classification label)

Let's see the high-level code for implementing an LLM using Pytorch:

1. **Tokenziation: Using pretrained model GPT2**

```
#Install transformers, pytorch and tensorflow-I am using google colab
pip install transformers

pip install pytorch

#or

pip install tensorflow
```

```
from transformers import GPT2Tokenizer

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
text = "Hello, world!"
tokens = tokenizer.encode(text)
print(tokens)
```

```
Downloading (…)olve/main/vocab.json: 100%
1.04M/1.04M [00:00<00:00, 4.96MB/s]
Downloading (…)olve/main/merges.txt: 100%
456k/456k [00:00<00:00, 15.1MB/s]
Downloading (…)lve/main/config.json: 100%
665/665 [00:00<00:00, 35.5kB/s]
#output [15496, 11, 995, 0]
```

2. Model Configuration and Creation :

```
from transformers import GPT2LMHeadModel, GPT2Config

config = GPT2Config(vocab_size=50257, n_positions=1024, n_ctx=1024,
                    n_embd=768, n_layer=12, n_head=12)

model = GPT2LMHeadModel(config)
```

3. Forward Pass and Language Modelling:

```
import torch

# Input tokens as tensor
input_ids = torch.tensor([tokens])

# Forward pass
outputs = model(input_ids)

# Predicted logits for next token
logits = outputs.logits

# Choose the token with the highest probability as the next token
predicted_token = torch.argmax(logits, dim=-1)[:, -1].item()
print(predicted_token)

#output-29606
```

4. Convert the predicted token index to the corresponding token

```
# Convert the predicted token index to the corresponding token
predicted_token_text = tokenizer.decode([predicted_token])

# Print the predicted token
print(predicted_token_text)

# Alive --> 29606 ---> Alive
```

Image by the Author

```
pip install transformers
pip install pytorch
#or
pip install tensorflow
pip install sentencepiece
```

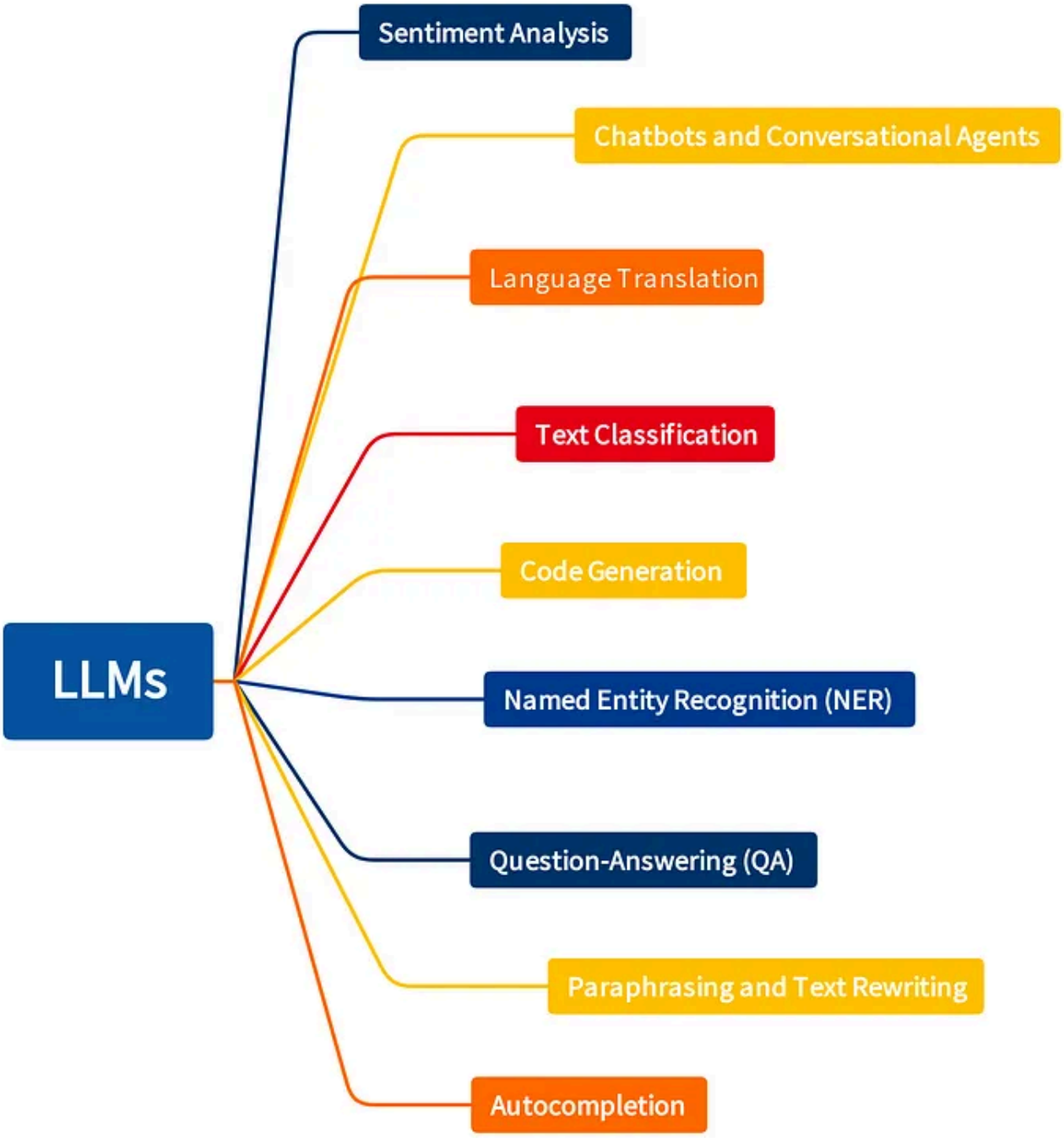


Image by the Author

Sentiment Analysis: LLMs can analyze the sentiment expressed in a text, determining whether it is positive, negative, or neutral. Sentiment analysis is commonly used for analyzing product reviews, social media posts, and customer feedback.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from transformers import pipeline

# Load pre-trained sentiment analysis model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased-finetuned-ss
```

```
model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-unc

# Create sentiment analysis pipeline
sentiment_pipeline = pipeline("sentiment-analysis", model=model, tokenizer=toke

# Perform sentiment analysis on sample text
text = "I love this restaurant! The food is amazing."
sentiment = sentiment_pipeline(text)[0]
print(sentiment)
```

```
Downloading (…)okenizer_config.json: 100%
48.0/48.0 [00:00<00:00, 2.56kB/s]
Downloading (…)lve/main/config.json: 100%
629/629 [00:00<00:00, 34.4kB/s]
Downloading (…)solve/main/vocab.txt: 100%
232k/232k [00:00<00:00, 9.96MB/s]
Downloading pytorch_model.bin: 100%
268M/268M [00:02<00:00, 92.3MB/s]
Output--> {'label': 'POSITIVE', 'score': 0.9998877048492432}
```

Language Translation: LLMs can be used for machine translation, where they translate text from one language to another. For example, translating a sentence from English to French or from Chinese to Spanish.

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
from transformers import pipeline

# Load pre-trained translation model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("Helsinki-NLP/opus-mt-en-fr")
model = AutoModelForSeq2SeqLM.from_pretrained("Helsinki-NLP/opus-mt-en-fr")

# Create translation pipeline
translation_pipeline = pipeline("translation_en_to_fr", model=model, tokenizer=

# Perform translation on sample text
text = "The weather is beautiful today."
translation = translation_pipeline(text, max_length=40)[0]['translation_text']
print(translation)
```

```
Downloading (…)olve/main/source.spm: 100%
778k/778k [00:00<00:00, 938kB/s]
Downloading (…)olve/main/target.spm: 100%
802k/802k [00:00<00:00, 956kB/s]
Downloading (…)olve/main/vocab.json: 100%
1.34M/1.34M [00:01<00:00, 1.26MB/s]
/usr/local/lib/python3.9/dist-packages/transformers/models/marian/tokenization_
  warnings.warn("Recommended: pip install sacremoses.")
Downloading pytorch_model.bin: 100%
301M/301M [00:17<00:00, 18.4MB/s]
Downloading (…)neration_config.json: 100%
293/293 [00:00<00:00, 15.6kB/s]
output---> Le temps est beau aujourd'hui.
```

Named Entity Recognition (NER): LLMs can be used to identify and classify named entities (e.g., people, organizations, locations) mentioned in a text. For example, given the sentence "Barack Obama was born in Hawaii," the model would identify "Barack Obama" as a person and "Hawaii" as a location.

```
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline

# Load pre-trained NER model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("dbmdz/bert-large-cased-finetuned-cor
model = AutoModelForTokenClassification.from_pretrained("dbmdz/bert-large-casec

# Create NER pipeline
ner_pipeline = pipeline("ner", model=model, tokenizer=tokenizer)

# Perform NER on sample text
text = "Barack Obama was born in Hawaii."
entities = ner_pipeline(text)
print(entities)

#[{'entity': 'I-PER', 'score': 0.9990103, 'index': 1, 'word': 'Barack',
#'start': 0, 'end': 6}, {'entity': 'I-PER', 'score': 0.999342, 'index': 2,
#'word': 'Obama', 'start': 7, 'end': 12}, {'entity': 'I-LOC', 'score': 0.99945,
# 'index': 6, 'word': 'Hawaii', 'start': 25, 'end': 31}]
```

Question-Answering (QA): LLMs can be trained to answer questions based on a given context or passage. For example, given the context "Mount Everest is the

tallest mountain in the world," and the question "What is the tallest mountain?", the model would generate the answer "Mount Everest."

```
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
from transformers import pipeline

# Load pre-trained QA model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-large-uncased-whole-word-maskir
model = AutoModelForQuestionAnswering.from_pretrained("bert-large-uncased-whole

# Create QA pipeline
qa_pipeline = pipeline("question-answering", model=model, tokenizer=tokenizer)

# Perform QA on sample text
context = "Mount Everest is the tallest mountain in the world."
question = "What is the tallest mountain?"
answer = qa_pipeline(question=question, context=context)
print(answer)
```

```
#{'score': 0.9668874144554138, 'start': 0, 'end': 13, 'answer': 'Mount Everest'
```

Code Generation: LLMs can be trained to generate code based on natural language descriptions. For example, given the description "Create a function that adds two numbers and returns the result," the model could generate the corresponding Python code.

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load pre-trained code generation model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("microsoft/CodeGPT-small-py")
model = GPT2LMHeadModel.from_pretrained("microsoft/CodeGPT-small-py")

# Perform code generation on sample text
text = "Create a function that adds two numbers and returns the result."
input_ids = tokenizer.encode(text, return_tensors='pt')

# Generate code
output = model.generate(input_ids, max_length=50, num_return_sequences=1, no_re
generated_code = tokenizer.decode(output[0], skip_special_tokens=True)
```

```
# Print the generated code
print(generated_code)
```

```
#output-->Create a function that adds two numbers and returns the result. :
param func: The function to add :type func : int :return: A function with
two values :rtype: function """ def func_wrapper(func): @functools.wraps(
```

Paraphrasing and Text Rewriting: LLMs can rephrase or rewrite sentences while preserving the original meaning. This capability can be used for tasks such as paraphrase detection, data augmentation, and content generation.

```
from transformers import PegasusTokenizer, PegasusForConditionalGeneration

# Load pre-trained paraphrasing model and tokenizer
tokenizer = PegasusTokenizer.from_pretrained("tuner007/pegasus_paraphrase")
model = PegasusForConditionalGeneration.from_pretrained("tuner007/pegasus_parap

# Original text to be paraphrased
original_text = "Students are practising hard in order to participate in the st

# Tokenize and encode the original text
input_ids = tokenizer.encode(original_text, return_tensors="pt")

# Generate paraphrased text
paraphrased_ids = model.generate(input_ids, max_length=50, num_return_sequences
paraphrased_text = tokenizer.decode(paraphrased_ids[0], skip_special_tokens=Tru

# Print the paraphrased text
print(paraphrased_text)
```

```
#Output-->Students are practicing for the state tournament.
```

Text Completion and Autocompletion: LLMs can predict and complete sentences or phrases based on partial input. This capability is useful for applications such as predictive typing, autocompletion, and writing assistance.

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load pre-trained GPT-2 tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")

# Partial text to be completed
partial_text = "Once upon a time, in a faraway land, there lived a"

# Tokenize and encode the partial text
input_ids = tokenizer.encode(partial_text, return_tensors="pt")

# Generate text completions
output = model.generate(input_ids, max_length=50, num_return_sequences=1, no_re
completed_text = tokenizer.decode(output[0], skip_special_tokens=True)

# Print the completed text
print(completed_text)
```

```
#Once upon a time, in a faraway land, there lived a man who was a great man,
#and he was called the Lord of the Rings. He was the son of a nobleman,
#a son who had been born in the land of
```

Chatbots and Conversational Agents: LLMs can be used to build chatbots and conversational agents that understand and respond to natural language queries from users. These agents can be used for customer support, virtual assistance, and interactive applications.

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load pre-trained GPT-2 tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")

def generate_response(input_text):
    # Encode the input text and add the end-of-sentence token
    input_ids = tokenizer.encode(input_text + tokenizer.eos_token, return_tensc

    # Generate a response
    response_ids = model.generate(input_ids, max_length=100, num_return_sequenc
    response_text = tokenizer.decode(response_ids[0], skip_special_tokens=True)
```

```
    return response_text

# Chatbot loop
print("Chatbot: Hello! How can I help you?")
while True:
    # Get input from the user
    user_input = input("You: ")

    # Exit if the user types "exit" or "quit"
    if user_input.lower() in ["exit", "quit"]:
        break

    # Generate a response from the chatbot
    chatbot_response = generate_response(user_input)
    print(f"Chatbot: {chatbot_response}")

print("Chatbot: Goodbye!")
```

GPT2 is not that great. If you want to use the GPT 3 then you need to use the API and it is not open-sourced and available in huggingface like GPT2.

You can create an app using

- Streamlit
- Gradio
- Flask

For example, create a Streamlit app for a few of the above NLP activities:

```
pip install streamlit
pip install transformers
```

```
import streamlit as st
from transformers import pipeline

# Define functions for each task
def named_entity_recognition(text):
    ner_pipeline = pipeline("ner", grouped_entities=True)
    return ner_pipeline(text)
```

```
def text_classification(text):
    sentiment_pipeline = pipeline("sentiment-analysis")
    return sentiment_pipeline(text)

def text_completion(text):
    completion_pipeline = pipeline("text-generation")
    return completion_pipeline(text, max_length=50, do_sample=True)[0]["generat

def paraphrasing(text):
    paraphrase_pipeline = pipeline("text2text-generation", model="tuner007/pega
    return paraphrase_pipeline(text, max_length=60)[0]["generated_text"]

# Streamlit app
st.title("NLP Demo")
st.write("Select a task and provide input text:")

# User input
input_text = st.text_input("Input Text:")

# Task selection
task = st.selectbox("Select a Task:", ("Named Entity Recognition", "Text Classi

# Perform task and display result
if st.button("Submit"):
    if task == "Named Entity Recognition":
        result = named_entity_recognition(input_text)
    elif task == "Text Classification":
        result = text_classification(input_text)
    elif task == "Text Completion":
        result = text_completion(input_text)
    elif task == "Paraphrasing":
        result = paraphrasing(input_text)
    st.write("Result:")
    st.write(result)
```

```
streamlit run nlp_demo.py
```

```
http://localhost:8501/
```

You can refer to my previous project on Streamlit

How to Build an NLP Machine Learning App-End to End

NLP App using Streamlit and Python NLP Libraries

medium.com

Image by the Author

Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM

Two weeks ago, we released Dolly, a large language model (LLM) trained for less than $30 to exhibit ChatGPT-like human...

www.databricks.com

Weights: https://huggingface.co/databricks

Dataset: https://github.com/databrickslabs/dolly/tree/master/data

If you want to try it yourself check out the dolly model published in hugging face:

I used google colab to test it and the code is

```
%pip install accelerate>=0.12.0 transformers[torch]==4.25.1
```

```
import torch
from transformers import pipeline
```

```
generate_text = pipeline(model="databricks/dolly-v2-12b",
torch_dtype=torch.bfloat16, trust_remote_code=True,
device_map="auto")
```

```
generate_text("Can you compare Data Warehouse Vs Data Lake?")
```

```
#output
Both Data Warehouse and Data Lake are non-transient stores that store data
in a cloud-native format. They both provide a centralized location to store
and process data. However, the major difference is that Data Lake offers
both horizontal scalability and low-latency processing. Additionally, it
supports large data volume storage with the ability to process data in
milliseconds when necessary. Whereas, Data Warehouse provides security &
robust architecture but it is difficult to scale and process data in
milliseconds as it is usually run on a transaction server which is a
separate system from the production cluster.
```

Image by the Author

```
generate_text("Can you provide the skill sets needed for a Machine Learning Eng
```

```
ML Engineer should have skills in one or more of the following areas:
\n1. Statistics and Mathematics\n2. Python\n3. C++\n4.
System administration\n5. Good communication skills
```

```
generate_text("Can you provide the python code for reversing the linkedlist?")
```

```
A Python implementation of reversing a linked list is provided below.
\n\ndef reverse_linked_list(head):\n    """\n    Reverses a linked list
by prepending the node before the \n    current node to the list.\n
"""\n    new_list = [head]\n    last = head\n    while last is not
None:\n        new_list.append(last)\n        last = last.next\n
\n    new_last = last\n    new_last.next = None\n    \n    return new_list\n\n
```

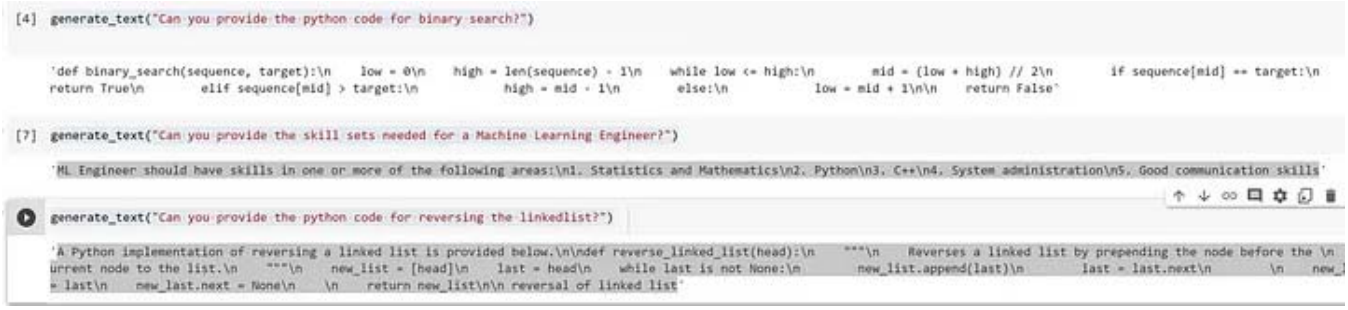


Image by the Author

Image by the Author

Huggingface:

If you want to more models check out

The following pipelines are available and you can play with them.

Task	Description	Modality	Pipeline identifier
Text classification	assign a label to a given sequence of text	NLP	pipeline(task="sentiment-analysis")
Text generation	generate text given a prompt	NLP	pipeline(task="text-generation")
Summarization	generate a summary of a sequence of text or document	NLP	pipeline(task="summarization")
Image classification	assign a label to an image	Computer vision	pipeline(task="image-classification")
Image segmentation	assign a label to each individual pixel of an image (supports semantic, panoptic, and instance segmentation)	Computer vision	pipeline(task="image-segmentation")
Object detection	predict the bounding boxes and classes of objects in an image	Computer vision	pipeline(task="object-detection")
Audio classification	assign a label to some audio data	Audio	pipeline(task="audio-classification")
Automatic speech recognition	transcribe speech into text	Audio	pipeline(task="automatic-speech-recognition")
Visual question answering	answer a question about the image, given an image and a question	Multimodal	pipeline(task="vqa")
Document question answering	answer a question about a document, given an image and a question	Multimodal	pipeline(task="document-question-answering")
Image captioning	generate a caption for a given image	Multimodal	pipeline(task="image-to-text")

Source huggingface documentation

For example, if you want to want to use sentiment analysis:

```
from transformers import pipeline
```

```
classifier = pipeline("sentiment-analysis")
```

```
results = classifier(["The economy is doing great and inflation is under contro
for result in results:
    print(f"label: {result['label']}, with score: {round(result['score'], 4)}")
```

```
# label: POSITIVE, with score: 0.9988
```

```
results = classifier(["The inflation is very high and it is affecting the econc
for result in results:
    print(f"label: {result['label']}, with score: {round(result['score'], 4)}")
```

```
#label: NEGATIVE, with score: 0.9997
```

Use AutoModelForSequenceClassification and AutoTokenizer to load the pretrained model and its associated tokenizer.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
```

```
model =
AutoModelForSequenceClassification.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Check out huggingface documentation for more pipelines and models.

Image by the Author-Adobe Firefly

You can try openai models API including chatgpt 4 and 3.5. It is not open-sourced like Dolly 2.0 or other open-sourced models. You can use the API and there is a fee for it.

MODELS	DESCRIPTION
GPT-4 Limited beta	A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code
GPT-3.5	A set of models that improve on GPT-3 and can understand as well as generate natural language or code
DALL·E Beta	A model that can generate and edit images given a natural language prompt
Whisper Beta	A model that can convert audio into text
Embeddings	A set of models that can convert text into a numerical form
Moderation	A fine-tuned model that can detect whether text may be sensitive or unsafe
GPT-3	A set of models that can understand and generate natural language
Codex Deprecated	A set of models that can understand and generate code, including translating natural language to code

Source -openai

Check out the tutorial: How to build an AI that can answer questions about your website.

Image by the Author

Langchain:

LangChain is a framework for developing applications powered by language models. Check out their Github for more info. Let's see a few examples based on their documentation.

```
pip install langchain
# or
conda install langchain -c conda-forge
```

```
pip install openai
```

```
export OPENAI_API_KEY="..."#Get the API key from OpenAI site

or

import os
os.environ["OPENAI_API_KEY"] = "..."
```

```
from langchain.llms import OpenAI
llm = OpenAI(temperature=0.9)
text = "Can you explain Big O notation"
print(llm(text))
#Answer
#Big O notation is a way to analyze the complexity or performance of an
algorithm relative to its input size. It is commonly used to measure the
time and/or space complexity of an algorithm in order to compare the
efficiency of different solutions. Big O notation is expressed using an
upper-case O followed by a function in parentheses. The function describes
the rate of growth of the algorithm's resource usage asymptotically as the
size of the inputs approaches infinity.
```

```
text = "Can you provide the python code for binary tree BFS traversal"
print(llm(text))
```

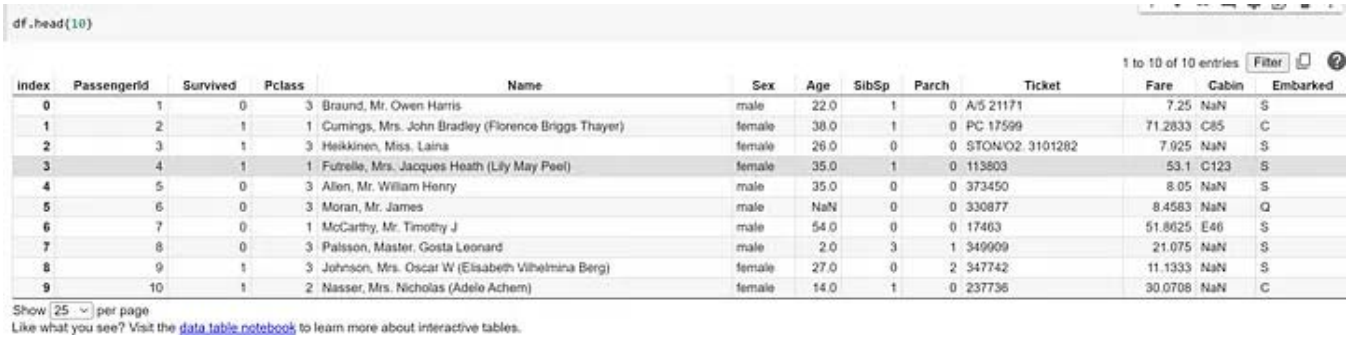
```
# BFS traversal of a binary tree
def bfs(root):
    if root is None:
        return

    queue = []
    queue.append(root)

    while(len(queue) > 0):
        print(queue[0].data, end = " ")
        node = queue.pop(0)
```

```
            if node.left is not None:
                queue.append(node.left)

            if node.right is not None:
                queue.append(node.right)
```

There are a lot of Agents. For example, we can work with the Pandas Dataframe agent.

```
from langchain.agents import create_pandas_dataframe_agent
```

```
from langchain.llms import OpenAI
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/ma
```



df.head(10)

1 to 10 of 10 entries Filter

index	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.075	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

Image by the Author

```
agent = create_pandas_dataframe_agent(OpenAI(temperature=0), df, verbose=True)

agent.run("how many rows are there?")

> Entering new AgentExecutor chain...
Thought: I need to count the number of rows
Action: python_repl_ast
Action Input: len(df)
Observation: 891
Thought: I now know the final answer
Final Answer: There are 891 rows in the dataframe.
```

```
> Finished chain.
There are 891 rows in the dataframe.
```

Image by the Author

```
agent.run("how many people have more than 3 sibligngs")

output:
> Entering new AgentExecutor chain...
Thought: I need to count the number of people with more than 3 siblings
Action: python_repl_ast
Action Input: df[df['SibSp'] > 3].shape[0]
Observation: 30
Thought: I now know the final answer
Final Answer: 30 people have more than 3 siblings.

> Finished chain.
30 people have more than 3 siblings.
```

```
agent.run("whats the square root of the average age?")

> Entering new AgentExecutor chain...
Thought: I need to calculate the average age first
Action: python_repl_ast
Action Input: df['Age'].mean()
Observation: 29.69911764705882
Thought: I can now calculate the square root
Action: python_repl_ast
Action Input: math.sqrt(df['Age'].mean())
Observation: name 'math' is not defined
Thought: I need to import the math library
Action: python_repl_ast
Action Input: import math
Observation:
Thought: I can now calculate the square root
Action: python_repl_ast
```

```
Action Input: math.sqrt(df['Age'].mean())
Observation: 5.449689683556195
Thought: I now know the final answer
Final Answer: 5.449689683556195

> Finished chain.
5.449689683556195
```

This agent calls the Python agent under the hood, which executes LLM-generated Python code. There are a lot of agents. Check it out. The Langchain documentation covers all the topics in detail.

Check Analysis of Twitter the-algorithm source code with LangChain, GPT4 and Deep Lake.

LLM Training Times:

Here is a table that shows the approximate training time for different models and hardware configurations:

Model Size	Parameters	Training Time (Days)	Hardware
1 billion	1B	4	16 GPUs
10 billion	10B	1	1024 GPUs
100 billion	100B	1 month	10240 GPUs
1 trillion	1T	1 year	102400 GPUs

Image credit

LLM DataSets:

Here are some of the publicly available datasets commonly used for training large language models, along with their URLs:

1. **Common Crawl:** A dataset that includes text from publicly available web pages. It is regularly updated and provides monthly snapshots. URL: https://commoncrawl.org/
2. **Wikipedia:** The entire text of Wikipedia, covering a diverse range of topics. Wikipedia data can be downloaded in various formats. URL: https://dumps.wikimedia.org/
3. **BooksCorpus:** A dataset containing the text of over 11,000 books from various genres. URL: http://yknzhu.wixsite.com/mbweb (Note: This dataset may no

longer be publicly available due to copyright concerns.)

4. **OpenWebText:** A dataset similar to the original GPT-3 training set, created from publicly available web pages. URL: https://github.com/jcpeterson/openwebtext
5. **Gutenberg Dataset:** A dataset containing the text of over 25,000 books from Project Gutenberg. URL: http://www.gutenberg.org/
6. **WikiText:** A dataset created by Salesforce Research that contains text from Wikipedia articles. Available in different sizes (e.g., WikiText-2, WikiText-103). URL: https://www.salesforce.com/products/einstein/ai-research/the-wikitext-dependency-language-modeling-dataset/
7. **Enron Email Dataset:** A dataset containing a large collection of email messages from the Enron Corporation. URL: https://www.cs.cmu.edu/~./enron/
8. **Reddit Comments:** A dataset containing comments from the social media platform Reddit. URL: https://files.pushshift.io/reddit/comments/
9. **The Toronto Book Corpus:** A dataset containing the text of over 7,000 books from a wide range of genres. URL: https://www.cs.toronto.edu/~mbweb/ (Note: This dataset may no longer be publicly available due to copyright concerns.)
10. **The Billion Word Benchmark:** A dataset containing approximately one billion words from web pages. URL: http://www.statmt.org/lm-benchmark/

For example, using the Reddit dataset you can develop a custom model. Again I used google colab with GPU.

```
from torch.utils.data import Dataset

# Create a custom dataset class to wrap tokenized inputs
class RedditDataset(Dataset):
    def __init__(self, encodings):
        self.encodings = encodings

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.item
        item['labels'] = item['input_ids'].clone()
        return item

    def __len__(self):
        return len(self.encodings.input_ids)
```

```
# Convert the tokenized inputs into a custom dataset
reddit_dataset = RedditDataset(inputs)

# Create the Trainer with the custom dataset
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=reddit_dataset,
)

# Fine-tune the model
trainer.train()

# Save the fine-tuned model
model.save_pretrained("./fine-tuned-model")
```

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load the pre-trained tokenizer (same tokenizer used during fine-tuning)
tokenizer = GPT2Tokenizer.from_pretrained("distilgpt2")

# Load the fine-tuned model
model_name = "/content/fine-tuned-model"
model = GPT2LMHeadModel.from_pretrained(model_name)

# Define a prompt for text generation
prompt = "Once upon a time, in a land far away,"

# Encode the prompt and prepare it for the model
input_ids = tokenizer.encode(prompt, return_tensors="pt")

# Generate text based on the prompt
output = model.generate(
    input_ids,
    max_length=100,  # Maximum length of the generated text
    num_return_sequences=1,  # Number of generated sequences
    no_repeat_ngram_size=2,  # Avoid repeating n-grams
    temperature=0.7,  # Higher values result in more random text, lower values
    top_k=50,  # Top K tokens considered for each step
)

# Decode and print the generated text
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(generated_text)

#The attention mask and the pad token id were not set. As a consequence, you ma
#Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
#Once upon a time, in a land far away, I was in the middle of a long road,
```

```
#and I had a friend who was a little older than me. He was about to go to
#the bathroom, so I grabbed his hand and started to walk. I didn't know what
#to do, but I thought I'd just walk in and walk out.
#I was walking in, my friend was standing there, looking at me, as if to say,
#"Oh, you're
```

Image by the Author-Adobe Firefly

Prompt engineering: The process of designing effective prompts to guide a large language model's responses, thereby improving its performance on specific tasks. This may involve careful phrasing of questions, providing examples, or setting specific constraints.

Transformer Architecture: The underlying neural network architecture used in LLMs, is characterized by its self-attention mechanism. Transformers are highly effective in capturing long-range dependencies in sequences of text.

Bidirectional Encoder Representations from Transformers (BERT): A pre-trained language model developed by Google that uses bidirectional transformers to learn context-aware word embeddings. BERT has achieved state-of-the-art results on a variety of natural language processing tasks and can be fine-tuned for specific tasks with limited labeled data.

Self-Attention: A mechanism in transformers that allows each input token to attend to all other tokens in the input sequence. It helps the model understand the context and relationships between words in a sentence.

Tokenization: The process of splitting a text input into smaller units, called tokens, which can be individual words, subwords, or characters. Tokenization is a crucial preprocessing step in LLMs.

Pre-training: The initial training phase of LLMs where the model is trained on a large, unlabeled text corpus. During pre-training, the model learns language representations, grammar, and contextual relationships between words.

Fine-tuning: The second training phase of LLMs where the pre-trained model is further trained on a smaller, task-specific labeled dataset. Fine-tuning helps the model specialize in a specific task, such as text classification or question-answering.

Language Modeling: The task of predicting the next word or token in a sequence, given the preceding words or tokens. Language modeling is a common objective during the pre-training phase of LLMs.

Masked Language Modeling: A variation of language modeling where some tokens in the input sequence are randomly masked, and the model is trained to predict the masked tokens based on the context. This is the training objective used in BERT and similar models.

Vocabulary: The set of all unique tokens that the LLM is capable of understanding and generating. The vocabulary size affects the model's ability to represent a diverse set of words and phrases.

Embedding Layer: A layer in the neural network that converts input tokens into continuous vectors or embeddings. These embeddings capture the semantic information of words and are used as input to the transformer layers.

Perplexity: A measure of how well a language model predicts a sample of text. Lower perplexity indicates a better fit of the model to the data.

Generative Model: A type of model that is capable of generating new data samples that resemble the training data. LLMs are generative models as they can generate text based on a given input.

Autoregressive Model: A type of language model that predicts each token in a sequence based on the tokens that came before it. GPT (Generative Pre-trained Transformer) is an example of an autoregressive language model.

Transfer learning: A technique where a pre-trained model, typically trained on a large dataset, is fine-tuned on a smaller, task-specific dataset. The pre-trained model acts as a starting point, enabling the model to learn faster and achieve better performance on the target task, even with limited labeled data.

Curriculum learning: A training strategy that gradually increases the training examples' difficulty to improve the model's learning efficiency and generalization.

By starting with more straightforward examples, the model can learn fundamental concepts before progressing to more complex ones.

Model distillation: A technique for training smaller, less complex models (students) to mimic the behavior of larger, more complex models (teachers). Model distillation can reduce the computational complexity and resource requirements of a model while maintaining a similar performance to the teacher model.

Model pruning: A technique for reducing the size and complexity of a neural network by removing less important neurons, weights, or connections. Model pruning can help reduce the memory and computational requirements of a model, making it more suitable for deployment on resource-constrained devices.

Image by the Author

Frameworks and Libraries Used for LLMs:

Some of the most popular ones include:

Hugging Face Transformers: A widely-used Python library that provides pre-trained LLMs (e.g., BERT, GPT, RoBERTa) and tools for fine-tuning and deploying these models. It supports both PyTorch and TensorFlow backends.

- Website: https://huggingface.co/transformers/

TensorFlow: An open-source machine learning framework developed by Google, which can be used to build, train, and deploy LLMs from scratch or using pre-built models.

- Website: https://www.tensorflow.org/

PyTorch: An open-source deep learning library developed by Facebook, which provides tensor computation and deep neural network capabilities for building and training LLMs.

- Website: https://pytorch.org/

Keras: A high-level neural networks API that can run on top of TensorFlow, Microsoft Cognitive Toolkit, or Theano. It provides an easy-to-use interface for building and training LLMs.

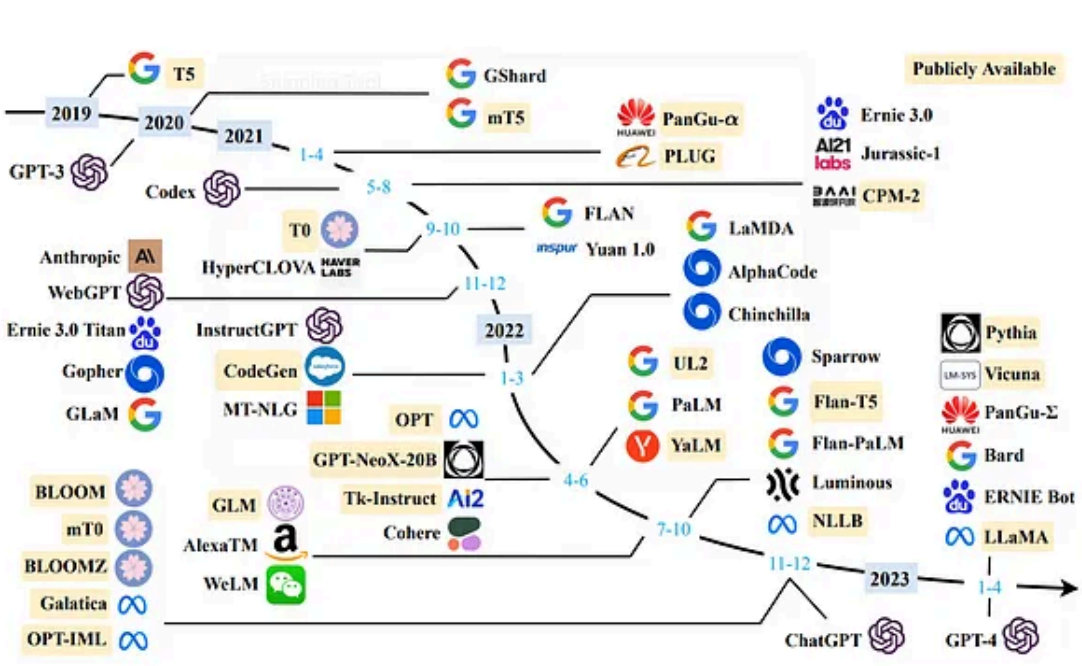
- Website: https://keras.io/

OpenAI: The organization behind some of the most popular LLMs like GPT-3 and GPT-4. OpenAI provides access to their models through the OpenAI API, which allows developers to integrate LLMs into their applications without directly handling the models.

- Website: https://www.openai.com/

Image by the Author

List of LLMs available :



Source:A Survey of Large Language Models

1. GPT4: From OpenAi

GPT-4

We've created GPT-4, the latest milestone in OpenAI's effort in scaling up deep learning. GPT-4 is a large multimodal...

openai.com

2. Alpaca: From Stanford

Stanford CRFM

We introduce Alpaca 7B , a model fine-tuned from the LLaMA 7B model on 52K instruction-following demonstrations. On our...

crfm.stanford.edu

Github:

GitHub - tatsu-lab/stanford_alpaca: Code and documentation to train Stanford's Alpaca models, and...

This is the repo for the Stanford Alpaca project, which aims to build and share an instruction-following LLaMA model...

github.com

3.LLaMA: From Meta

Introducing LLaMA: A foundational, 65-billion-parameter language model

As part of Meta's commitment to open science, today we are publicly releasing LLaMA (Large Language Model Meta AI), a...

ai.facebook.com

4. Dolly: From Databricks:

Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM

Two weeks ago, we released Dolly, a large language model (LLM) trained for less than $30 to exhibit ChatGPT-like human...

www.databricks.com

GitHub - databrickslabs/dolly: Databricks' Dolly, a large language model trained on the Databricks...

Databricks' Dolly is an instruction-following large language model trained on the Databricks machine learning platform...

github.com

5. GPT Finance — Bloomberg Model:

BloombergGPT: A Large Language Model for Finance

The use of NLP in the realm of financial technology is broad and complex, with applications ranging from sentiment...

arxiv.org

6. Bard — From Google

Bard

Bard is your creative and helpful collaborator to supercharge your information, boost productivity, and bring ideas to...

bard.google.com

7. Bloom -From Huggingface

bigscience/bloom · Hugging Face

BigScience Large Open-science Open-access Multilingual Language ModelVersion 1.3 / 6 July 2022 Current Checkpoint...

huggingface.co

Check out for more on LLMs

1. A list of 1 billion LLMs
2. List of Open Sourced Fine-Tuned Large Language Models (LLM)

Image generated by Adobe Firefly

Image by the Author

Additional Resources:

1. A table of contents for reviews on research papers within AI.

2. Building LLM applications for production — by Chip Huyen

3. Huggingface Transformers

4. Stanford Alpaca

5. Shreyas take on LLMs in production

6. Eight Things to Know about Large Language Models

7. Experimenting with LLMs to Research, Reflect, and Plan.

8. LLM Twitter thread

9. Check out this thread and notebook.

10.Running Python micro-benchmarks using the ChatGPT Code Interpreter alpha.

11. LLM's ranking.

12. How does Langchain work? Check this thread.

13. BabyAGI User Guide.

14. Performance of ChatGPT, GPT-4, and Google Bard on a Neurosurgery Oral Boards Preparation Question Bank

15. This project brings large-language model and LLM-based chatbot to web browsers. Everything runs inside the browser with no server support and accelerated with WebGPU.

16. Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models.

17 . Stanford 2023 AI Report and AI in 14 charts.

18. Stanford University — Large Language Model Course-CS324

19. COS 597G (Fall 2022): Understanding Large Language Models-Princeton University-Free course

20. Huggingface Transformers course

21. Let's build GPT: from scratch, in code, spelled out-Andrej Karpathy

22. Large Language Models, Spring 2023

Image by the Author

1. Instruct-tune LLaMA on consumer hardware
2. Building applications with LLMs
3. Code and documentation to train Stanford's Alpaca models, and generate the data.
4. A gradio web UI for running Large Language Models like LLaMA, llama.cpp, GPT-J, Pythia, OPT, and GALACTICA.
5. The simplest, fastest repository for training/finetuning medium-sized GPTs.LLMs Vs Classical ML Models.
6. An English-language shell for any OS, powered by LLMs.
7. Foundation Models

Image by the Author

In conclusion, Large Language Models (LLMs) have emerged as powerful and versatile tools that are revolutionizing the field of natural language processing. As state-of-the-art models continue to grow in size and sophistication, they unlock new

possibilities and applications across numerous domains, including healthcare, finance, and customer service.

Image by the Author

1. All the GitHub repos mentioned above.
2. Huggingface-https://huggingface.co/
3. Openai — https://openai.com/
4. Databricks Dolly2.0 -https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm
5. All the Twitter feeds and articles mentioned above.
6. Attention is all you need-https://arxiv.org/abs/1706.03762
7. List of Large Language MOdels-https://en.wikipedia.org/wiki/Large_language_model
8. Available LLMs-https://slashdot.org/software/large-language-models/

Machine Learning | Data Science | Programming | Python | NLP

Following

Written by Senthil E

2.7K Followers · Writer for Analytics Vidhya

ML/DS - Certified GCP Professional Machine Learning Engineer, Certified AWS Professional Machine learning Speciality,Certified GCP Professional Data Engineer .

More from Senthil E and Analytics Vidhya

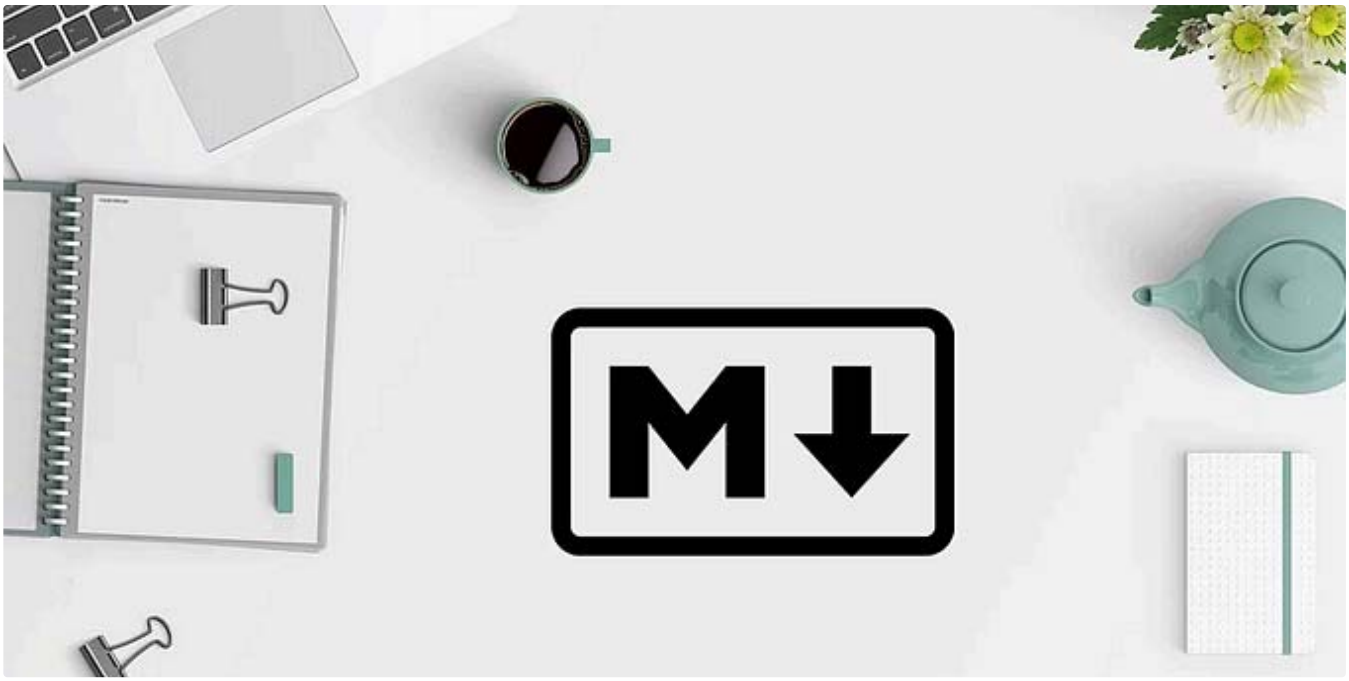
Senthil E in Level Up Coding

Navigating the World of LLMs: A Beginner's Guide to Prompt Engineering-Part 2

From Basics To Advanced Techniques

32 min read · Mar 17, 2024

302



Hannan Satopay in Analytics Vidhya

The Ultimate Markdown Guide (for Jupyter Notebook)

An in-depth guide for Markdown syntax usage for Jupyter Notebook

10 min read · Nov 18, 2019

2.2K 13

Harikrishnan N B in Analytics Vidhya

Confusion Matrix, Accuracy, Precision, Recall, F1 Score

Binary Classification Metric

6 min read · Dec 10, 2019

922 6

Senthil E in Level Up Coding

Unleashing the Potential of LLMs: How Enterprises are Leveraging AI for Enhanced Services

From Chatbots to Automation: Exploring the Versatile Use Cases of LLMs in Enterprises

58 min read · Mar 31, 2024

113

See all from Senthil E

See all from Analytics Vidhya

Recommended from Medium

Amogh Agastya

Decoding LLM Performance: A Guide to Evaluating LLM Applications

Exploring frameworks and strategies for evaluating LLM Applications

21 min read · Dec 30, 2023

157 2

Jane Huang in Data Science at Microsoft

Evaluating LLM systems: Metrics, challenges, and best practices

A detailed consideration of approaches to evaluation and selection

11 min read · Mar 5, 2024

867 10

Lists

Predictive Modeling w/ Python

20 stories · 1111 saves

Practical Guides to Machine Learning

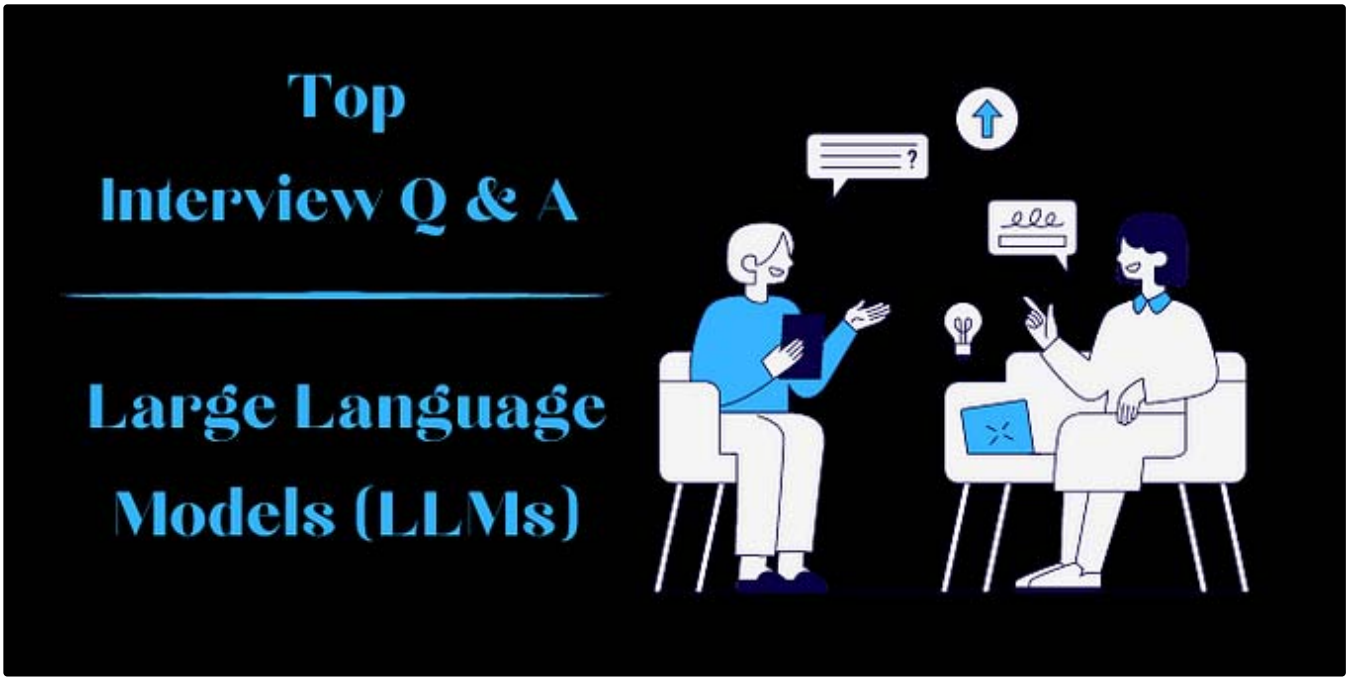
10 stories · 1325 saves

Coding & Development

11 stories · 567 saves

Natural Language Processing

1380 stories · 873 saves



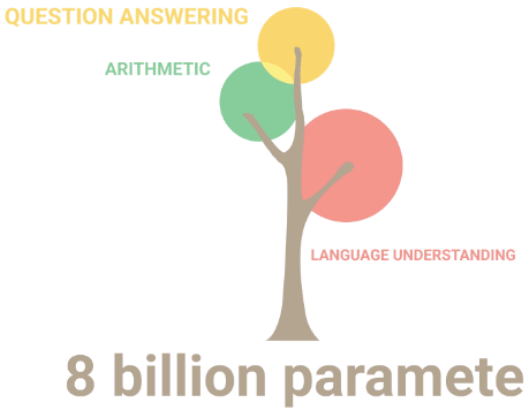
Youssef Hosni in Level Up Coding

Top Large Language Models (LLMs) Interview Questions & Answers

Demystifying Large Language Models (LLMs): Key Interview Questions and Expert Answers

· 42 min read · Sep 5, 2023

542 3



Amber Ivanna Trujillo

Top 10—Must Know GenAI with LLM [Large Language Model] Interview questions

With the rise of the LLM era, every company and organization is looking into resources to learn and implement the best features of this...

· 10 min read · Oct 20, 2023

63

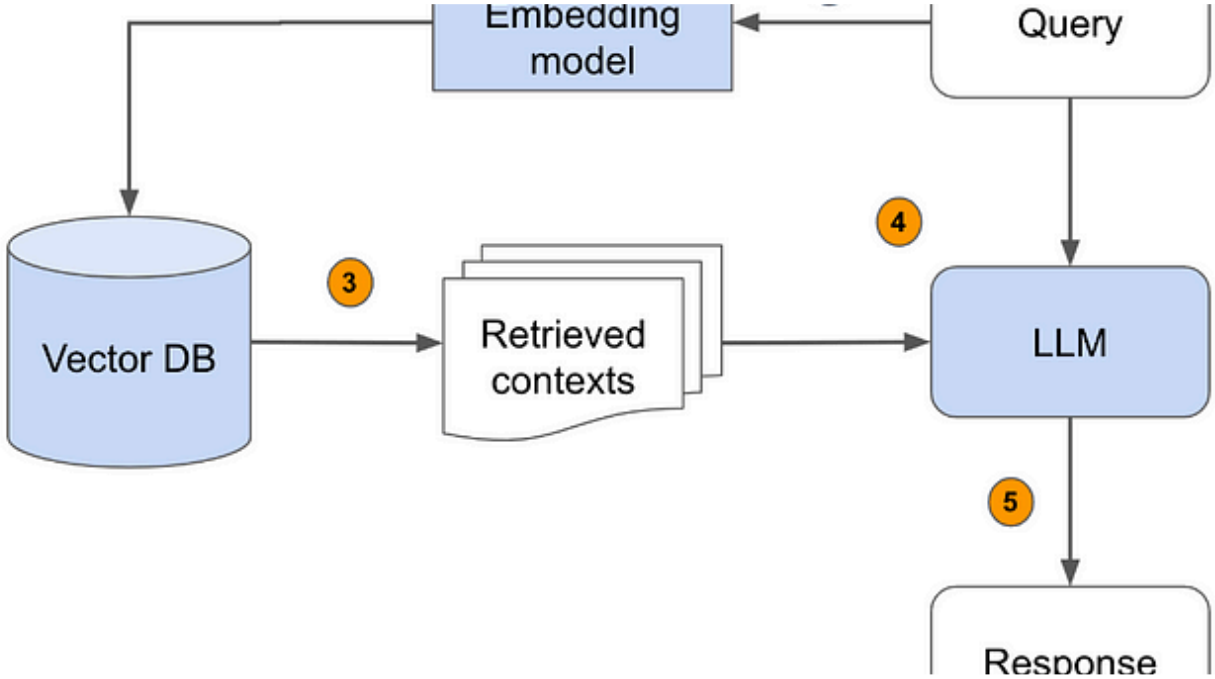
Ameydhote

Fine-Tuning Language Models for NER: A Hands-On Step-by-Step Guide

With its unparalleled ability to comprehend and produce text that resembles that of a person, large language models, or LLMs, have become...

6 min read · Feb 1, 2024

40 1



Bijit Ghosh

RAG Vs VectorDB

Introduction to RAG and VectorDB

14 min read · Jan 28, 2024

187 4

See more recommendations