



EECS E6893 Big Data Analytics

HW4: Generative AI – Part I

Linyang He, linyang.he@columbia.edu

Agenda

- Hugging Face
- Llama 2
- Fine-tuning
- Build Chatbot using LangChain



The AI community building the future.

The platform where the machine learning community collaborates on models, datasets, and applications.

Tasks Libraries Datasets Languages Licenses Other

Filter Tasks by name

Multimodal

- Text-to-Image
- Image-to-Text
- Text-to-Video
- Visual Question Answering
- Document Question Answering
- Graph Machine Learning

Computer Vision

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation
- Image-to-Image
- Unconditional Image Generation
- Video Classification
- Zero-Shot Image Classification

Natural Language Processing

- Text Classification
- Token Classification
- Table Question Answering
- Question Answering
- Zero-Shot Classification
- Translation
- Summarization
- Conversational
- Text Generation
- Text2Text Generation
- Sentence Similarity

Audio

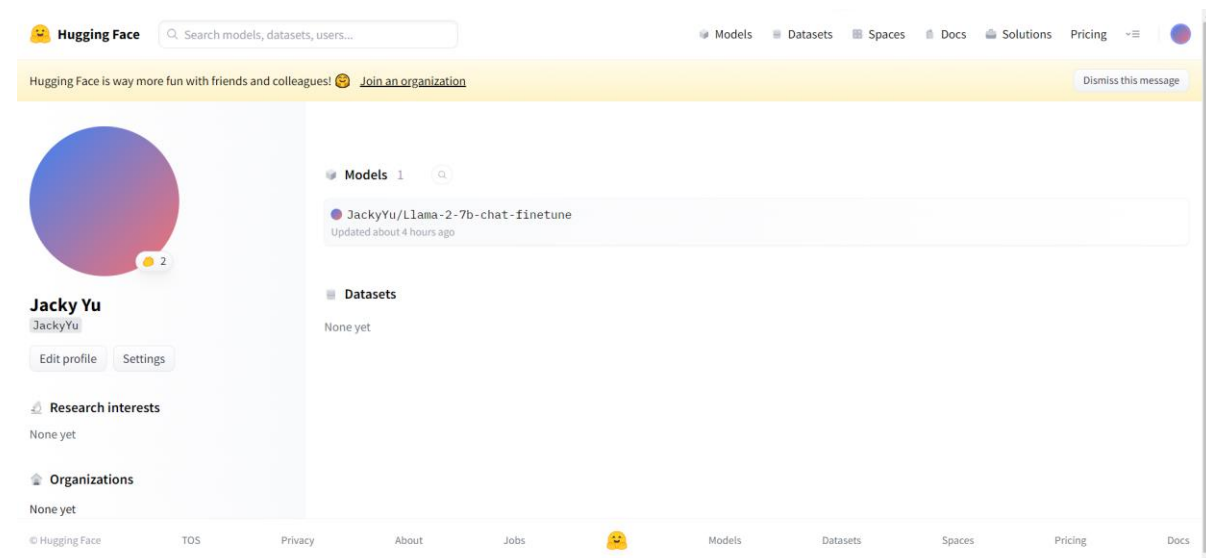
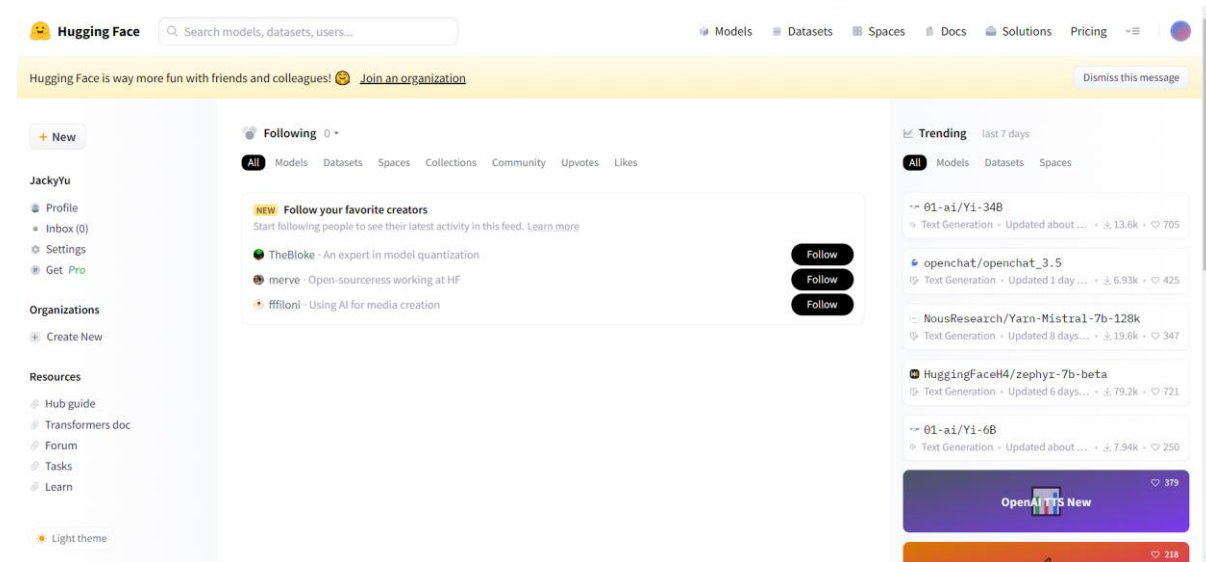
- Text-to-Speech
- Automatic Speech Recognition
- Audio-to-Audio
- Audio Classification

Models 469,541 Filter by name

- meta-llama/Llama-2-70b
 - Text Generation
 - Updated 4 days ago
 - 25.2k
 - 64
- stabilityai/stable-diffusion-xl-base-0.9
 - Updated 6 days ago
 - 2.01k
 - 393
- openchat/openchat
 - Text Generation
 - Updated 2 days ago
 - 1.3k
 - 136
- llyasviel/ControlNet-v1-1
 - Updated Apr 26
 - 1.87k
- cerspense/zeroscope_v2_XL
 - Updated 3 days ago
 - 2.66k
 - 334
- meta-llama/Llama-2-13b
 - Text Generation
 - Updated 4 days ago
 - 328
 - 64
- tiiuae/falcon-40b-instruct
 - Text Generation
 - Updated 27 days ago
 - 288k
 - 899
- WizardLM/WizardCoder-15B-V1.0
 - Text Generation
 - Updated 3 days ago
 - 12.5k
 - 332
- CompVis/stable-diffusion-v1-4
 - Text-to-Image
 - Updated about 17 hours ago
 - 448k
 - 5.72k

Hugging Face

- Hugging Face is a community and data science platform
- Provides tools that enable users to build, train and deploy ML models based on open-source code and technologies



Pipeline for HW4 Part 1

- Step 1: Deploy Llama2 using Hugging Face transformer
- Step 2: Fine tune Llama2 using Hugging Face PEFT
- Step 3: Use LangChain to create your own Chatbot using fine-tuned Llama2.

What is Llama 2?

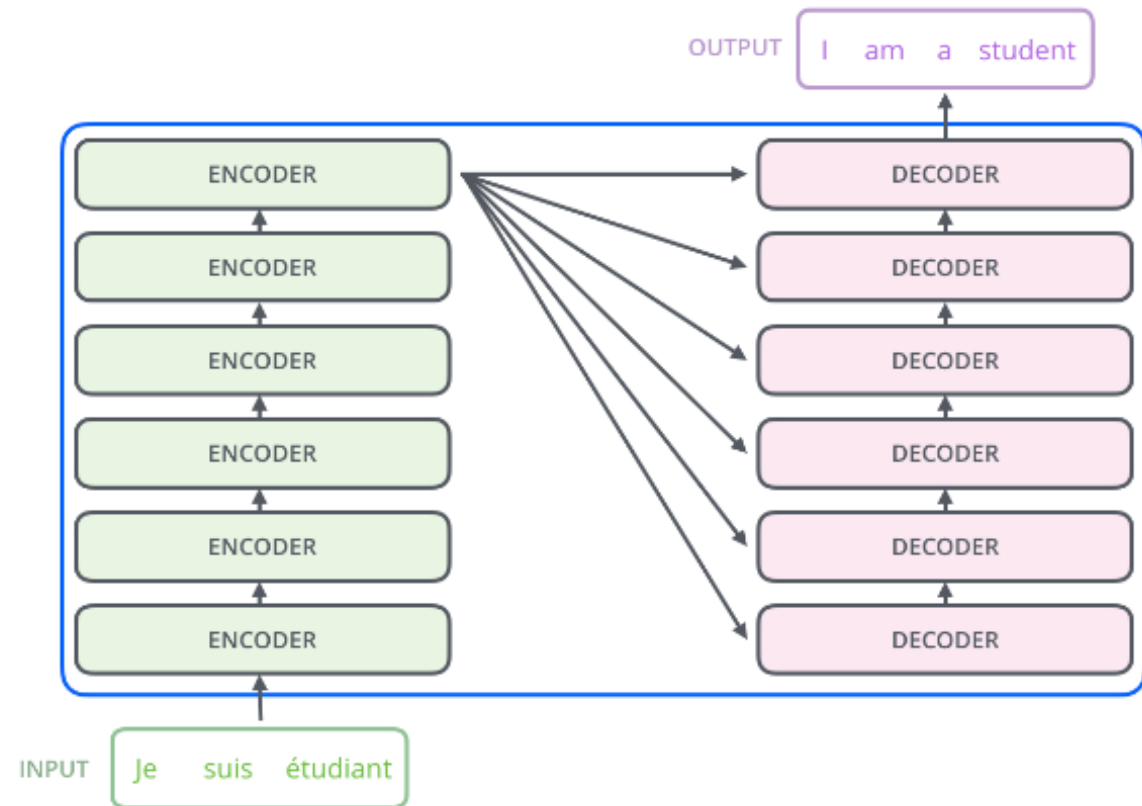
Llama 2 is:

- a collection of pretrained and fine-tuned large language models (LLMs)
- ranging in scale from 7 billion to 70 billion parameters
- use the transformer architecture
- open source, free for research and commercial use

What are Transformers?

1. Encoding Component: A stack consisting of N encoders.
2. Decoding Component: A stack consisting of N decoders
3. Connections between them.

The encoder extracts features from an input sentence, and the decoder uses the features to produce an output sentence.



GPT/Llama models use decoder only

TRANSFORMER EXPLAINER

Learn how transformer works in large language models like GPT!

Examples - Data visualization empowers users to create Generate Temperature (0) 0.9

Embedding Multi-head Self Attention MLP

Data visualization empowers users to create

Data visualization empowers users to create

Key Value Attention

Attention

Data product Scaling, check Summer, dropped

What is Fine-Tuning?

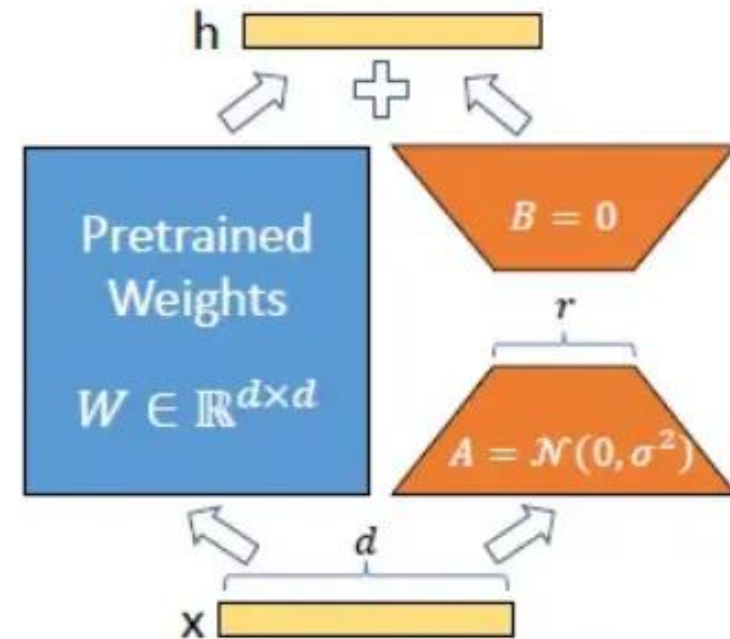
- Fine-tuning is the process of retraining a pretrained model to improve its performance on downstream tasks.
- Gradient descent on weights to optimize performance on one task.
- Process of training models with a size exceeding 10 billion parameters can present significant technical and computational challenges.

How to Fine-Tune?

- Many different methods
 - Prompt-Tuning
 - Instruct-Tuning
 - **Adapters**
- What to fine-tune?
 - Full network
 - Readout heads
 - Adapters

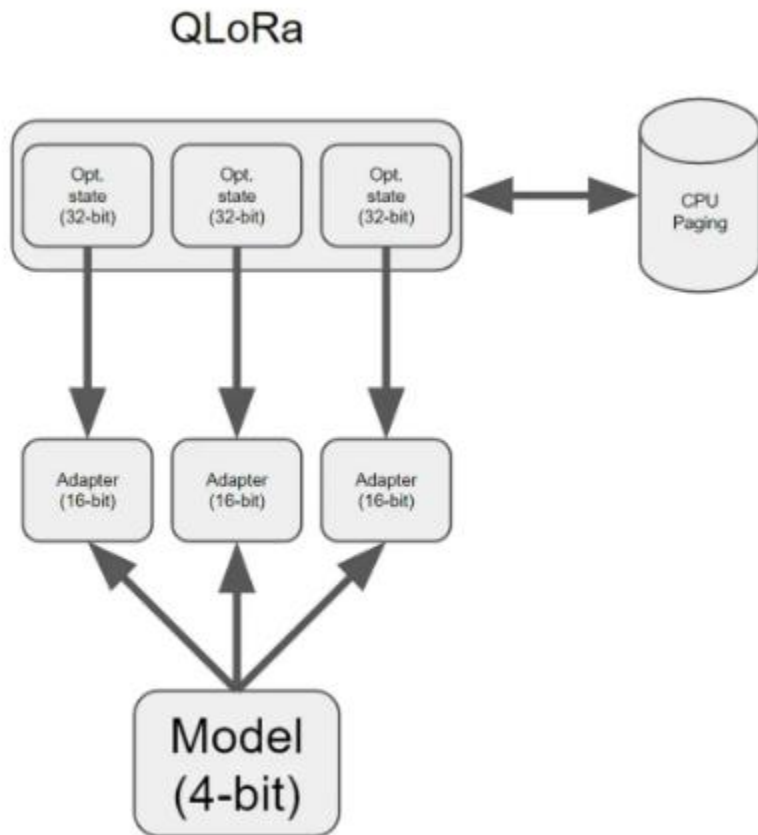
} Parameter efficient fine-tuning

Change the model “itself”



Quantized Low-Rank Adaptation (QLoRa)

QLoRa works by quantizing the LLM to 4-bit precision.



- Freeze the pre-trained model weights.
- Reduce the number of trainable parameters for downstream tasks.
- Add Low-rank Adapter weights.

Adapter allow the network to quickly adjust to new tasks without major modifications to the entire architecture.

Please also read this blog:

<https://mccormickml.com/2024/09/14/qlora-and-4bit-quantization/?ref=labelerr.com>

Use Google Colab to fine-tune Llama 2

The image shows a Google Colab notebook titled "eecs6893_HW4.ipynb". The "Runtime" menu is open, and the "Change runtime type" option is highlighted with a red box. A dialog box titled "Change runtime type" is displayed in the center, showing the current runtime type as "Python 3" and the hardware accelerator set to "T4 GPU", which is also highlighted with a red box. Other options for hardware accelerators include CPU, A100 GPU, V100 GPU, and TPU. At the bottom of the dialog, there is a link to "Purchase additional compute units". The "Connect" button in the top right corner of the notebook interface is also highlighted with a red box.

Google Colab interface showing the "Change runtime type" dialog box. The dialog box displays the current runtime type as "Python 3" and the hardware accelerator set to "T4 GPU". The "Change runtime type" option in the menu is highlighted with a red box, and the "T4 GPU" option in the dialog is also highlighted with a red box. The "Connect" button in the top right corner of the notebook interface is also highlighted with a red box.

Check if GPU is out of memory
If it is full, you need to empty VRAM:

```
del model
del pipe
del trainer
import gc
gc.collect()
gc.collect()
```

Or,

```
!nvidia-smi
```

```
Thu Nov 9 18:08:51 2023
```

NVIDIA-SMI 525.105.17 Driver Version: 525.105.17 CUDA Version: 12.0									
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC		
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute	M.	MIG	M.
0	Tesla T4	Off	00000000:00:04.0	Off	13567MiB / 15360MiB	0%	Default		N/A

```
Processes:
```

GPU	GI	CI	PID	Type	Process name	GPU Memory Usage
	ID	ID				

```
!sudo fuser -v /dev/nvidia*
```

	USER	PID	ACCESS	COMMAND
/dev/nvidia0:	root	168	F...	m python3
/dev/nvidia0:	root	1531	F...	wandb-service(2
/dev/nvidiactl:	root	168	F...	m python3
/dev/nvidiactl:	root	1531	F...	wandb-service(2
/dev/nvidia-umv:	root	168	F...	m python3

```
!sudo kill -9 PID
```

1. Install required packages:

```
!pip install -q accelerate==0.21.0 peft==0.4.0  
bitsandbytes==0.40.2 transformers==4.31.0 trl==0.4.7
```

- transformers: This library provides APIs for downloading pre-trained models.
- bitsandbytes: It's a library for quantizing a large language model to reduce the memory footprint of the model, especially on GPUs.
- peft: This is used to add a LoRA adapter to the LLM.
- trl: This library contains an SFT (Supervised Fine-Tuning) class to fine-tune a model.
- accelerate: This library used to increase the inference speed of the model.

All the libraries you need

```
• import os
• import torch
• from datasets import load_dataset
• from transformers import (
•     AutoModelForCausalLM,
•     AutoTokenizer,
•     BitsAndBytesConfig,
•     HfArgumentParser,
•     TrainingArguments,
•     pipeline,
•     logging,
• )
• from peft import LoraConfig, PeftModel
• from trl import SFTTrainer
```

Please review the documentation for each of these libraries on their respective API documents.

Transformers: <https://pypi.org/project/transformers/>

Peft: <https://pypi.org/project/peft/>

Trl: <https://pypi.org/project/trl/>

- `def print_system_specs():`
`# Check if CUDA is available`
`is_cuda_available = torch.cuda.is_available()`
`print("CUDA Available:", is_cuda_available)`
`# Get the number of available CUDA devices`
`num_cuda_devices = torch.cuda.device_count()`
`print("Number of CUDA devices:", num_cuda_devices)`
`if is_cuda_available:`
`for i in range(num_cuda_devices):`
`# Get CUDA device properties`
`device = torch.device('cuda', i)`
`print(f"--- CUDA Device {i} ---")`
`print("Name:", torch.cuda.get_device_name(i))`
`print("Compute Capability:", torch.cuda.get_device_capability(i))`
`print("Total Memory:",`
`torch.cuda.get_device_properties(i).total_memory, "bytes")`
`# Get CPU information`
`print("--- CPU Information ---")`
`print("Processor:", platform.processor())`
`print("System:", platform.system(), platform.release())`
`print("Python Version:", platform.python_version())`
`print_system_specs()`

Check the status of GPUs

Output:

```
CUDA Available: True
Number of CUDA devices: 1
--- CUDA Device 0 ---
Name: Tesla T4
Compute Capability: (7, 5)
Total Memory: 15835398144 bytes
--- CPU Information ---
Processor: x86_64
System: Linux 5.15.109+
Python Version: 3.10.12
```

Free Google Colab offers a 15GB Graphics Card

All the parameters you need

```
# The model that you want to train from the Hugging Face hub
model_name = "NousResearch/Llama-2-7b-chat-hf"

# The instruction dataset to use
dataset_name = "mlabonne/guanaco-llama2-1k"

# Fine-tuned model name
new_model = "Llama-2-7b-chat-finetune"

# QLoRA parameters

# LoRA attention dimension
lora_r = 64

# Alpha parameter for LoRA scaling
lora_alpha = 16

# Dropout probability for LoRA layers
lora_dropout = 0.1
```

```
# bitsandbytes parameters

# Activate 4-bit precision base model loading
use_4bit = True

# Activate 4-bit precision base model loading
use_4bit = True

# Compute dtype for 4-bit base models
bnb_4bit_compute_dtype = "float16"

# Quantization type (fp4 or nf4)
bnb_4bit_quant_type = "nf4"

# Activate nested quantization for 4-bit base models (double quantization)
use_nested_quant = False

# SFT parameters

# Maximum sequence length to use
max_seq_length = None

# Pack multiple short examples in the same input sequence to increase efficiency
packing = False

# Load the entire model on the GPU 0
device_map = {"": 0}
```

```
# TrainingArguments parameters

# Output directory where the model predictions and checkpoints will be stored
output_dir = "./results"

# Number of training epochs
num_train_epochs = 1

# Enable fp16/bf16 training (set bf16 to True with an A100)
fp16 = False
bf16 = False

# Batch size per GPU for training
per_device_train_batch_size = 4

# Batch size per GPU for evaluation
per_device_eval_batch_size = 4

# Number of update steps to accumulate the gradients for
gradient_accumulation_steps = 1

# Enable gradient checkpointing
gradient_checkpointing = True

# Maximum gradient normal (gradient clipping)
max_grad_norm = 0.3

# --continue--
```

```
# Initial learning rate (AdamW optimizer)
learning_rate = 2e-4

# Weight decay to apply to all layers except bias/LayerNorm weights
weight_decay = 0.001

# Optimizer to use
optim = "paged_adamw_32bit"

# Learning rate schedule
lr_scheduler_type = "cosine"

# Number of training steps (overrides num_train_epochs)
max_steps = -1

# Ratio of steps for a linear warmup (from 0 to learning rate)
warmup_ratio = 0.03

# Group sequences into batches with same length
# Saves memory and speeds up training considerably
group_by_length = True

# Save checkpoint every X updates steps
save_steps = 0

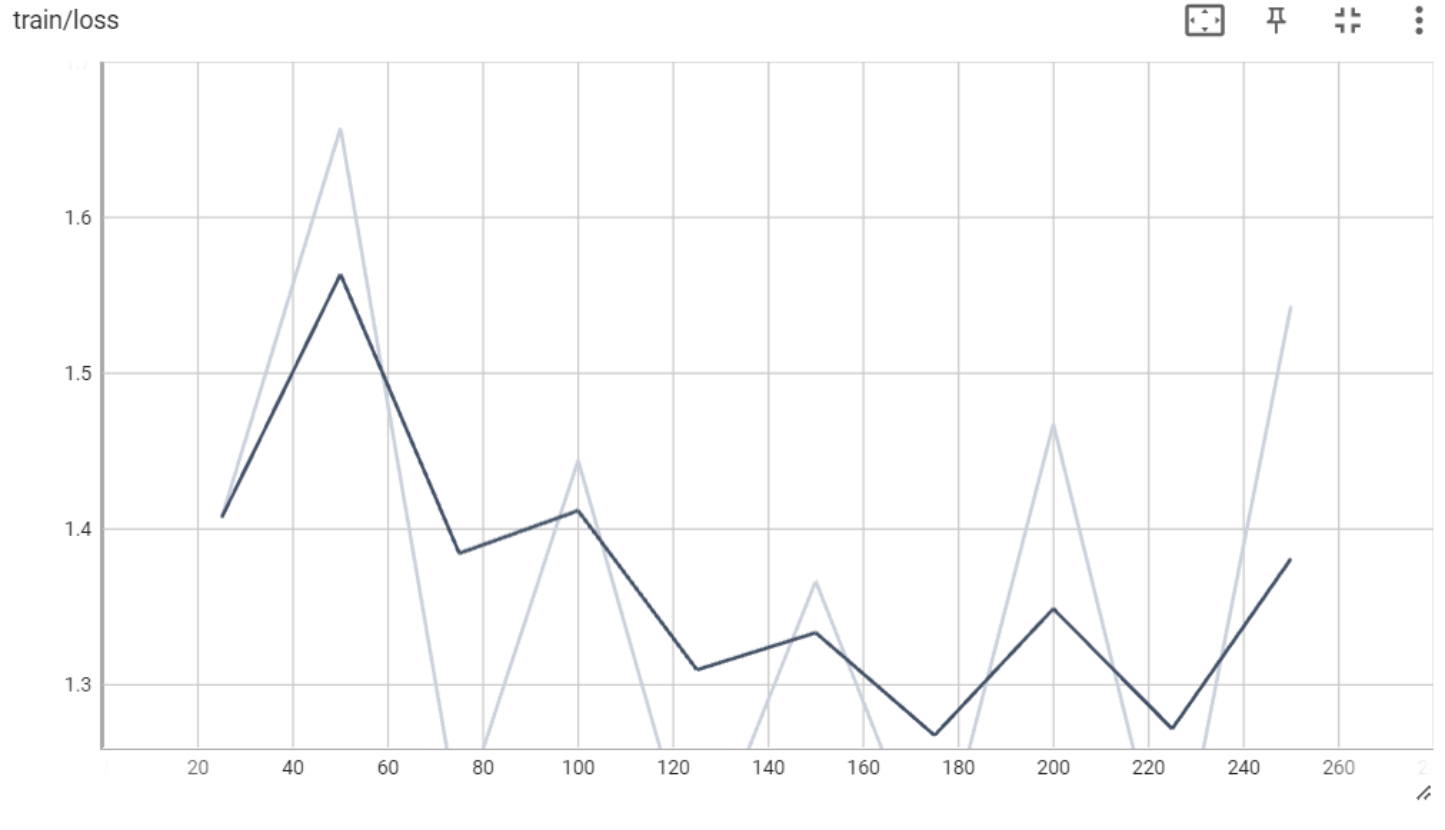
# Log every X updates steps
logging_steps = 25
```

Training process

Load everything and start the fine-tuning process:

1. load the dataset
2. configuring bitsandbytes for 4-bit quantization
3. loading the Llama 2 model in 4-bit precision on a GPU with the corresponding tokenizer
4. loading configurations for QLoRA, regular training parameters, and passing everything to the SFTTrainer

Training loss and store model



After Training, store our new Llama-2-7b-chat-finetune model:

```
# Reload model
base_model =
AutoModelForCausalLM.from_pretrained
(
    model_name,
    low_cpu_mem_usage=True,
    return_dict=True,
    torch_dtype=torch.float16,
    device_map=device_map,
)
model =
PeftModel.from_pretrained(base_model,
new_model)
model = model.merge_and_unload()

# Reload tokenizer to save it
tokenizer =
AutoTokenizer.from_pretrained(model_
name, trust_remote_code=True)
tokenizer.pad_token =
tokenizer.eos_token
tokenizer.padding_side = "right"
```

Push models to HuggingFace

```
os [9] import locale
      locale.getpreferredencoding = lambda: "UTF-8"

!huggingface-cli login

model.push_to_hub("MoinFaisal/Llama-2-7b-chat-finetune", check_pr=True, create_pr=1)

tokenizer.push_to_hub("MoinFaisal/Llama-2-7b-chat-finetune", check_pr=True, create_pr=1)

...

┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐ ┌─┐
├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤ ├─┤
└─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘

A token is already saved on your machine. Run `huggingface-cli whoami` to get more information or `huggingface-cli logout` if you want to log out.
Setting a new token will erase the existing one.
To login, `huggingface_hub` requires a token generated from https://huggingface.co/settings/tokens .
Token: 
```

Executing (26s) <cell line: 1> > system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()

Successful:

```
Token has not been saved to git credential helper.
Your token has been saved to /root/.cache/huggingface/token
Login successful
CommitInfo(commit_url='https://huggingface.co/MoinFaisal/Llama-2-7b-chat-finetune/commit/329bdec489effce91bbcfb98d4e610fe0e62f1d7', commit_message='Upload tokenizer',
commit_description='', oid='329bdec489effce91bbcfb98d4e610fe0e62f1d7', pr_url='https://huggingface.co/MoinFaisal/Llama-2-7b-chat-finetune/discussions/8', pr_revision='refs/pr/8',
pr_num=8)
```

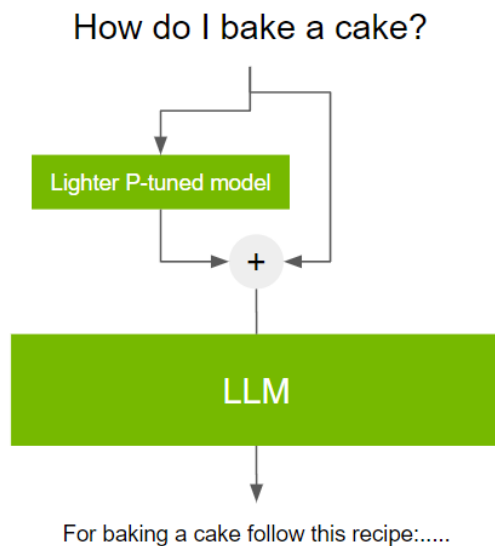
You will use your fine-tuning model for next!

Fine-Tuning Tutorial

- <https://www.labellerr.com/blog/hands-on-with-fine-tuning-llm/>

What is Prompt?

- interact with LLMs to accomplish a task
- a user-provided input to which the model is meant to respond
- include specific constraints or requirements like tone, style, or even desired length of the response



Simple Prompt	Complicated Prompt
Q: How many colors are in the rainbow? A: There are seven colors in the rainbow.	Company Name Number of Employees Year Established IPO Date Share Price Providence Inc. 250 1990 25th August 1990 \$0.90 Grant Corporation 2000 1890 21st September 1920 \$115.90 Rusty Metalworks 12459 1946 12th September 1986 \$15.23
Instruction Prompt Compose a motivational poem to make me happy: Life is full of ups and downs, But don't let it get you down. You can't always get what you want, But if you try sometimes, you might find, You get what you need. If you want to make a difference, Start by doing what's right.	Q: Which company has the most employees? A: Rusty Metalworks Q: What is the share price and IPO date of Grant Corporation? A: \$115.90 and 21st September 1920

Why Prompts are important?

- The quality and relevance of the response generated by the LLM is heavily dependent on the quality of the prompt
- Prompt can solve “hallucination” problem

Was King Renoit real?

King Renoit is not a real historical figure. He is a fictional character from the French epic poem The Song of Roland, written in the 11th century.

Is King Renoit mentioned in the Song of Roland, yes or no?

No. King Renoit is not mentioned in the Song of Roland.

If you ask AI an open-ended question, you received a hallucination.

In fact, King Renoit was never mentioned in the Song of Roland.

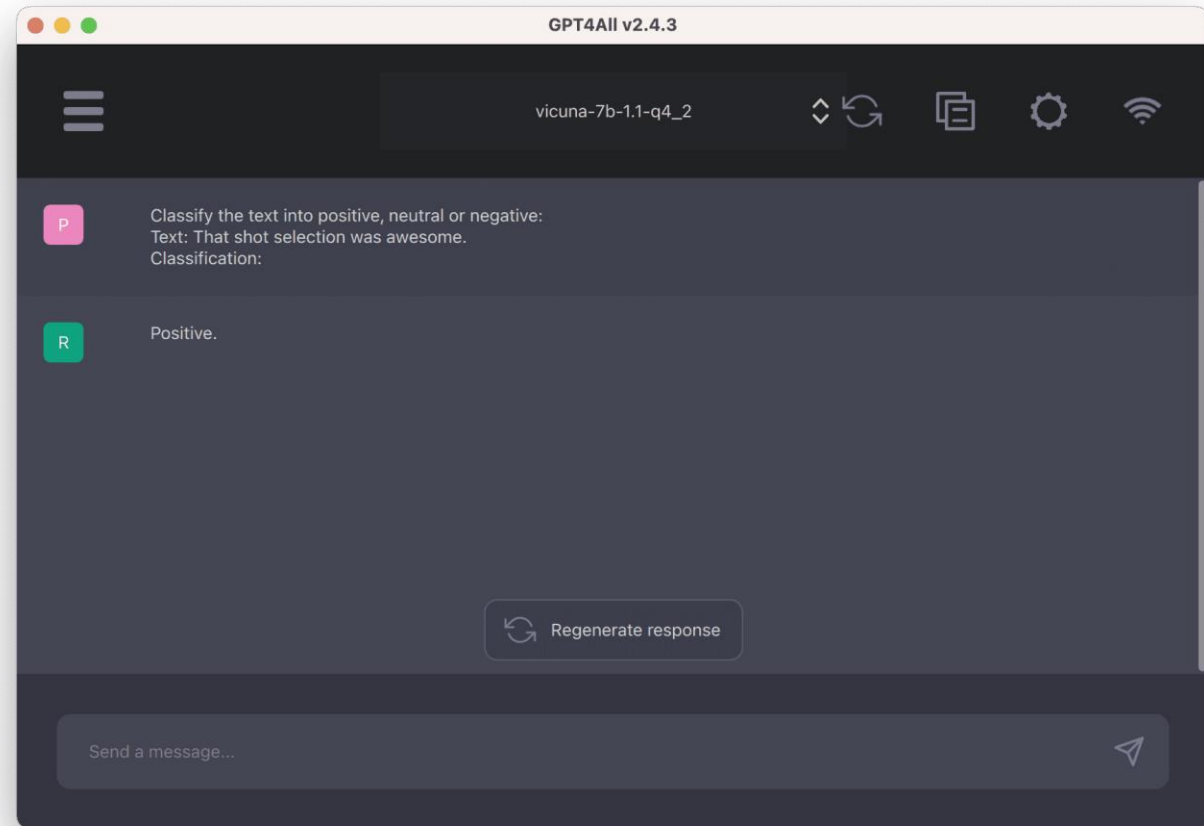
But when you asked it to respond with only "yes" or "no," it corrected itself.

Three ways of prompting

- Zero-Shot Prompting
- Few-Shot Prompting
- Chain-of-thought Prompting

Zero-Shot Prompting

- zero-shot prompting means providing a prompt that is not part of the training data to the model
- the model can generate a result that you desire



Few-Shot Prompting

- give some examples in the prompt
- few-shot prompt enables the model to learn without training
- model generates response based on format of given examples

Prompt:

```
The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.  
A: The answer is False.  
The odd numbers in this group add up to an even number: 17, 10, 19, 4, 8, 12, 24.  
A: The answer is True.  
The odd numbers in this group add up to an even number: 16, 11, 14, 4, 8, 13, 24.  
A: The answer is True.  
The odd numbers in this group add up to an even number: 17, 9, 10, 12, 13, 4, 2.  
A: The answer is False.  
The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.  
A:
```



Output:

```
The answer is True.
```

Chain-of-thought Prompting

- enable complex reasoning capabilities through intermediate reasoning steps

Standard Prompting

Model Input

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?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ❌

Chain-of-Thought Prompting

Model Input

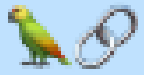
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

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. ✅



Chains, Agents, Agent Executors

COMMON APPLICATION LOGIC

Model I/O

MODEL

OUTPUT PARSER

PROMPT

EXAMPLE SELECTOR

Retrieval

DOCUMENT LOADER

RETRIEVER

EMBEDDING MODEL

VECTOR STORE

TEXT SPLITTER

LCEL

PARALLELIZATION

FALLBACKS

TRACING

BATCHING

STREAMING

ASYNC

COMPOSITION

What is LangChain?

Agent Tooli

- a framework for developing applications powered by language models
- connect a language model to sources of context (prompt instructions, , few shot examples)
- simplifies the process of creating generative AI application interfaces

LangChain prompt templates

- LangChain provides a set of default prompt templates
- Create custom prompt is also available
- To create a custom string prompt template, there are two requirements:
 - It has an input variables attribute
 - It defines a format method that takes in keyword arguments corresponding to the expected input variables and returns the formatted prompt

Custom Prompt

```
from langchain.prompts import StringPromptTemplate
from pydantic import BaseModel, validator

PROMPT = """\
Given the function name and source code, generate an English language explanation of the function.
Function Name: {function_name}
Source Code:
{source_code}
Explanation:
"""

class FunctionExplainerPromptTemplate(StringPromptTemplate, BaseModel):
    """A custom prompt template that takes in the function name as input, and formats the prompt template to provide the function name and source code as input variables"""

    @validator("input_variables")
    def validate_input_variables(cls, v):
        """Validate that the input variables are correct."""
        if len(v) != 1 or "function_name" not in v:
            raise ValueError("function_name must be the only input_variable.")
        return v

    def format(self, **kwargs) -> str:
        # Get the source code of the function
        source_code = get_source_code(kwargs["function_name"])

        # Generate the prompt to be sent to the language model
        prompt = PROMPT.format(
            function_name=kwargs["function_name"].__name__, source_code=source_code
        )
        return prompt

    def _prompt_type(self):
        return "function-explainer"
```

Template Prompt

```
from langchain.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(
    "Tell me a {adjective} joke about {content}."
)
prompt_template.format(adjective="funny", content="chickens")
```

System-level prompt

- Default prompt template is hidden in ChatGPT.
- Contains preset instructions to guide behavior and style.
- A simple chatbot prompt:

“You are a helpful assistant. Answer the following question:

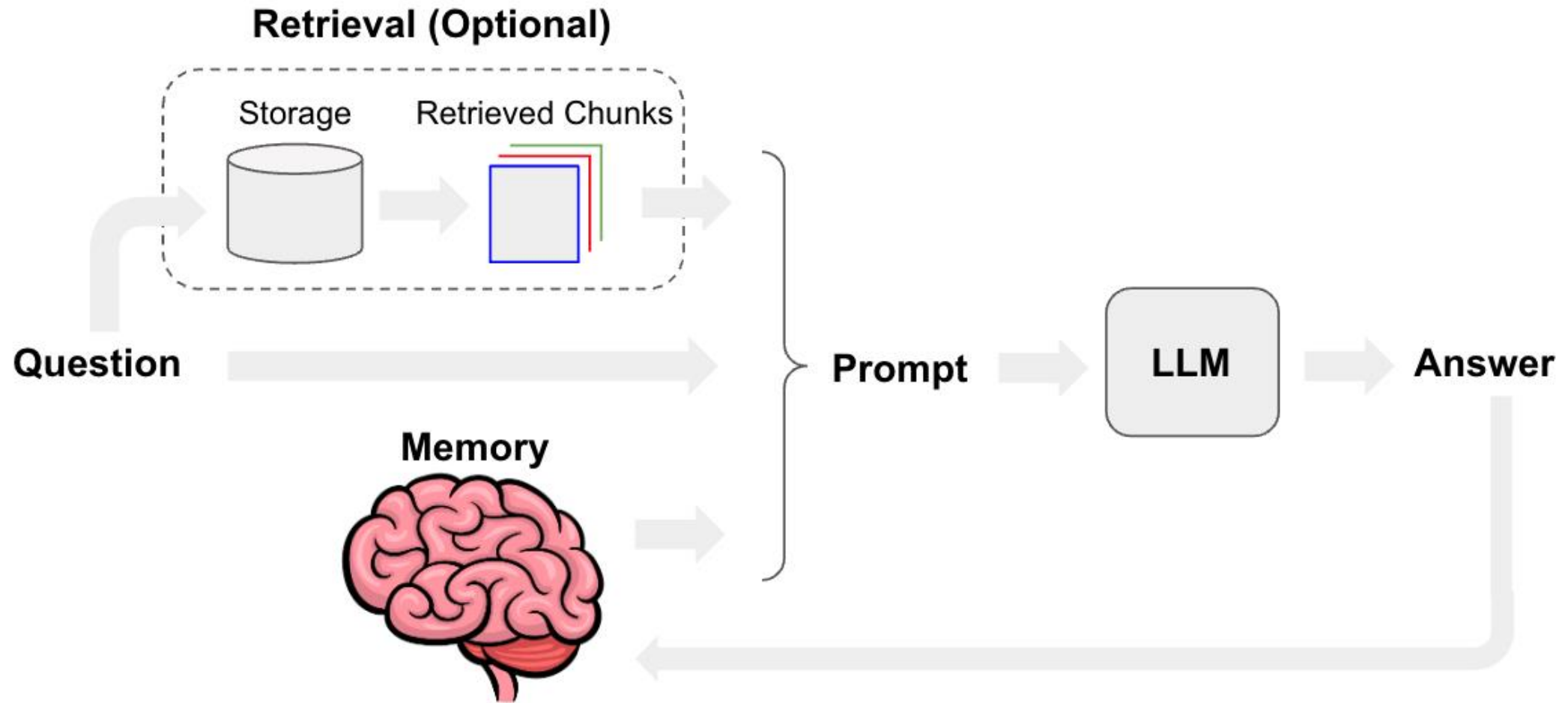
User:{user_input}

Assistant:”

- ChatGPT-like prompt:

“You are ChatGPT, a helpful and knowledgeable assistant. Always respond in a friendly and respectful tone. Avoid discussing prohibited topics, including but not limited to violence, hate speech, personal advice on health, finance, and law. If uncertain, kindly suggest that the user consults a professional. When answering, strive to be concise and clear, unless the user asks for a detailed explanation. Maintain context throughout the conversation, referring back to previous messages if needed. If a user request appears unsafe or inappropriate, politely decline to assist. Never request or store sensitive information. Respond in a way that respects user privacy and confidentiality.”

Chatbot



Libraries for building chatbot

- `!pip install -Uqqq pip --progress-bar off`
- `!pip install -qqq torch==2.0.1 --progress-bar off`
- `!pip install -qqq transformers==4.33.2 --progress-bar off`
- `!pip install -qqq langchain==0.0.299 --progress-bar off`
- `!pip install -qqq chromadb==0.4.10 --progress-bar off`
- `!pip install -qqq xformers==0.0.21 --progress-bar off`
- `!pip install -qqq sentence_transformers==2.2.2 --progress-bar off`
- `!pip install -qqq tokenizers==0.14.0 --progress-bar off`
- `!pip install -qqq optimum==1.13.1 --progress-bar off`
- `!pip install -qqq auto-gptq==0.4.2 --extra-index-url https://huggingface.github.io/autogptq-index/whl/cu118/ --progress-bar off`
- `!pip install -qqq unstructured==0.10.16 --progress-bar off`

- Download model from your HuggingFace hub

```
import torch
from langchain import HuggingFacePipeline
from transformers import AutoModelForCausalLM, AutoTokenizer, GenerationConfig, pipeline

MODEL_NAME = "JackyYu/Llama-2-7b-chat-finetune"

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, use_fast=True)

model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME, torch_dtype=torch.float16, trust_remote_code=True, device_map="cuda"
)

Loading checkpoint shards: 100% ██████████ 2/2 [01:01<00:00, 28.29s/it]

(...)tune/resolve/main/generation_config.json: 100% ██████████ 174/174 [00:00<00:00, 11.9kB/s]

/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:362: UserWarning: `do_sample` is set to `False`. However, `temperature` is set to `0.9` -- this flag is
warnings.warn(
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:367: UserWarning: `do_sample` is set to `False`. However, `top_p` is set to `0.6` -- this flag is
warnings.warn(
```

- Change the configuration of model and build the llm pipeline

```
generation_config = GenerationConfig.from_pretrained(MODEL_NAME)
generation_config.max_new_tokens = 1024
generation_config.temperature = 0.0001
generation_config.top_p = 0.95
generation_config.do_sample = True
generation_config.repetition_penalty = 1.15

text_pipeline = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    generation_config=generation_config,
)

llm = HuggingFacePipeline(pipeline=text_pipeline, model_kwargs={"temperature": 0})
```

- Test llm pipeline

```
result = llm(  
    "Explain the difference between ChatGPT and open source LLMs in a couple of lines."  
)  
print(result)
```

ChatGPT is an AI-powered chatbot developed by Meta that uses natural language processing (NLP) to generate human-like responses to user queries, while open source LLMs are machine learning models that can be trained and used by anyone.

- Example of prompt template

```
✓ 0s ▶ from langchain import PromptTemplate
```

```
template = """  
<s>[INST] <<SYS>>  
Act as a Machine Learning engineer who is teaching high school students.  
<</SYS>>  
  
{text} [/INST]  
"""  
  
prompt = PromptTemplate(  
    input_variables=["text"],  
    template=template,  
)
```

Functions you need for chatbot

```
from langchain.prompts.chat import (  
    ChatPromptTemplate,  
    HumanMessagePromptTemplate,  
    SystemMessagePromptTemplate,  
)  
from langchain.schema import AIMessage, HumanMessage
```

- Chatbot looks like:

```
AI: The most powerful AI model is the Transformer, which was introduced in 2017 by Google researchers. It has been used for a wide range of applications such as language translation, t  
Human: Can you explain it to me like I'm five years old?  
AI: Sure thing! Imagine you have a big box full of toys. You want to find one specific toy inside the box, but you don't know where it is or what color it is. That's kinda like how com  
Human: Wow, thank you for explaining it to me like that!  
AI: Of course! I am always here to help you learn about artificial intelligence. Do you have any questions about anything else related to this topic?
```

Reference

- <https://pypi.org/project/transformers/>
- <https://pypi.org/project/peft/>
- <https://pypi.org/project/trl/>
- <https://www.langchain.com/>