

Llama-2

Llama 2 is a new version of the Llama 1 model, which was made available previously for research. The new pretrained and fine-tuned versions of the model have been updated for commercial release. In addition to performing a variety of pretraining data-level investigations to help understand the potential capabilities and limitations of our models, we applied considerable safety mitigations to the fine-tuned versions of the model through supervised fine-tuning, reinforcement learning from human feedback (RLHF), and iterative red teaming (these steps are covered further in the section - Fine-tune for product).

The responsible fine-tuning flow:

Here are the general steps needed to responsibly fine-tune an LLM for alignment, guided at a high level by Meta's Responsible AI framework:

- 1. Define content policies & mitigations.**
- 2. Prepare data.**
- 3. Train the model.**
- 4. Evaluate and improve performance.**

Llama 2 SETUP:

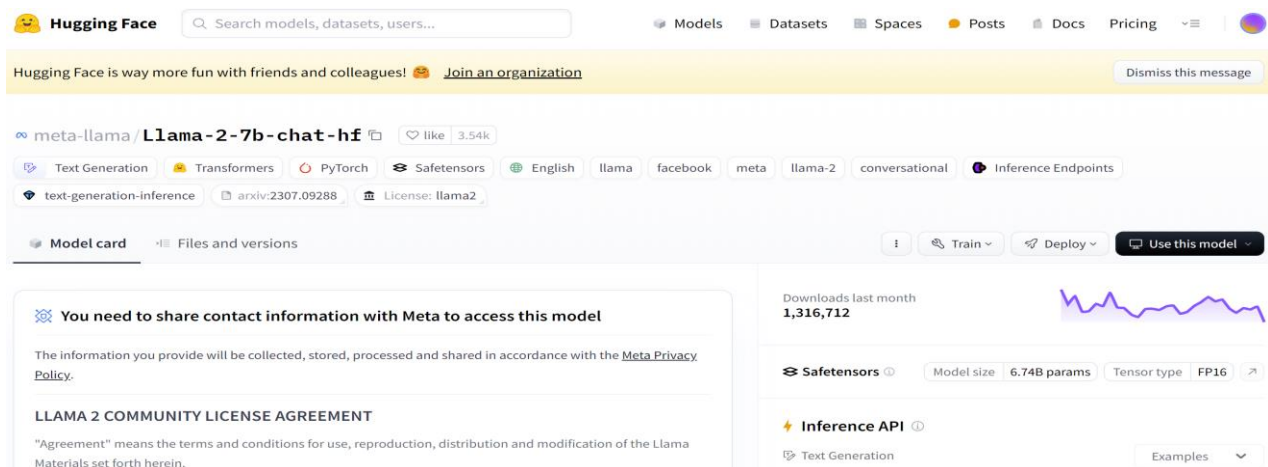
Setting up Llama 2 involves several steps, including obtaining access to the model, setting up the necessary environment, and writing code to interact with the model. Here's a detailed guide to help you set up Llama 2:

1. For access to Llama 2:

If Llama 2 is publicly available, you can download it from the repository or model hub (e.g., Hugging Face). If it's under restricted access, you may need to request access from Meta AI (formerly Facebook AI).

Website link: <https://huggingface.co/>

Model Version: Llama-2-7b-chat-hf.



Hugging Face is way more fun with friends and colleagues! [Join an organization](#) Dismiss this message

meta-llama / **Llama-2-7b-chat-hf** like 3.54k

Text Generation Transformers PyTorch Safetensors English llama facebook meta llama-2 conversational Inference Endpoints

text-generation-inference arxiv:2307.09288 License: llama2

Model card Files and versions Train Deploy Use this model

You need to share contact information with Meta to access this model

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

LLAMA 2 COMMUNITY LICENSE AGREEMENT

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

Downloads last month: **1,316,712**

Safetensors Model size: 6.74B params Tensor type: FP16

Inference API Text Generation Examples

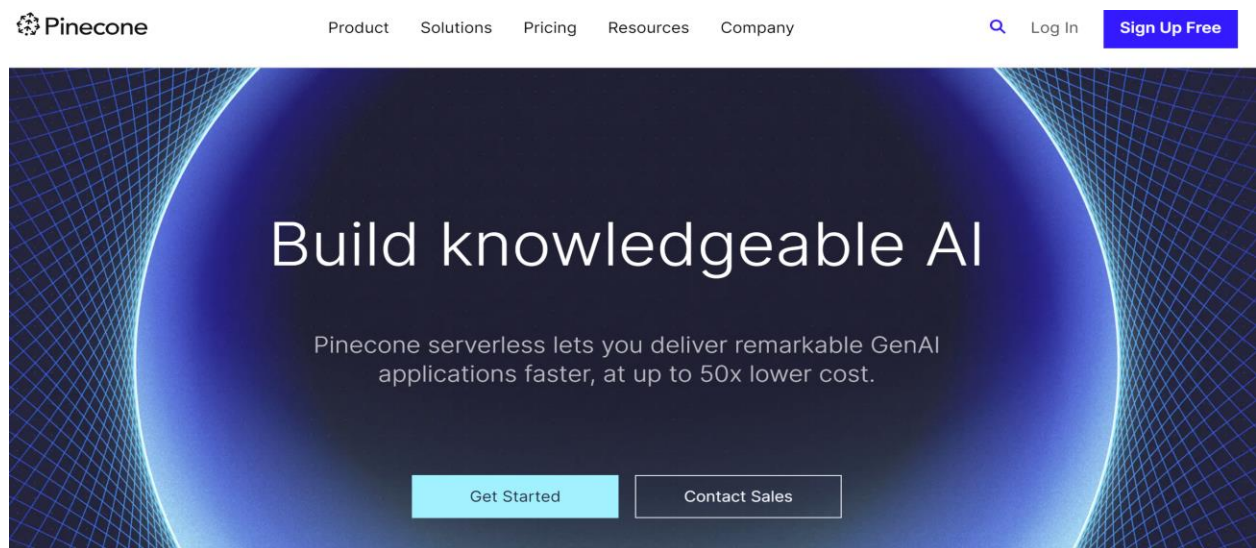
Link for Llama website: <https://llama.meta.com/>

Login and create account.

2. Access to pinecone:

Link for reference: <https://www.pinecone.io/>

Create your account.

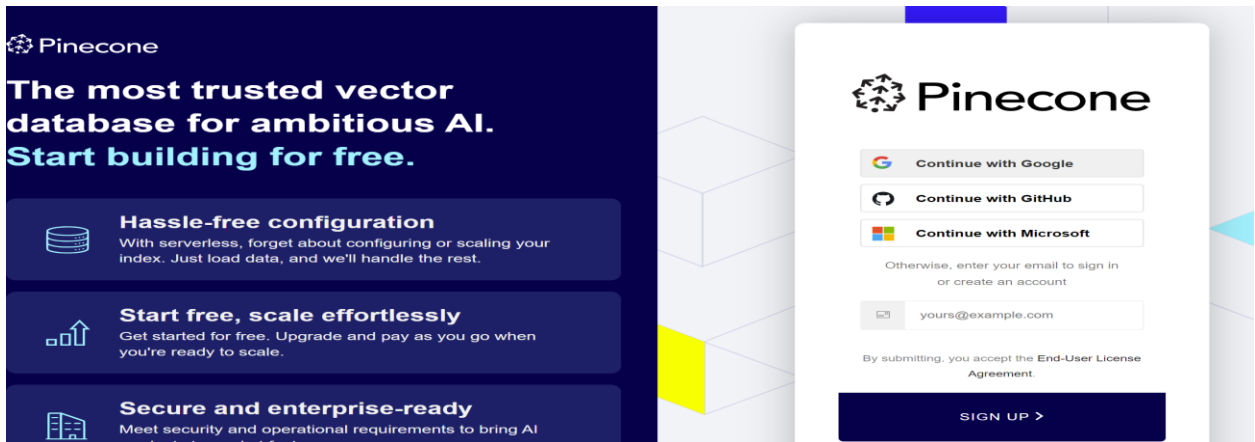


Pinecone Product Solutions Pricing Resources Company Log In [Sign Up Free](#)

Build knowledgeable AI

Pinecone serverless lets you deliver remarkable GenAI applications faster, at up to 50x lower cost.

[Get Started](#) [Contact Sales](#)



->Create Index

test-tutor

METRIC	DIMENSIONS	HOST
cosine	384	https://test-tutor-e0wyt6w.svc.aped-4627-b74a.pinecone.io

CLOUD	REGION	TYPE	VECTOR COUNT
aws AWS	us-east-1	Serverless	0

No Records Yet

A record is an object you add to an index containing a vector and, optionally, its metadata

[+ Add a Record](#)

->Create API key:

PORJ1 API Keys

Name	Value	Actions
default	*****	Copy Refresh Delete
tutorpi	*****	Copy Refresh Delete

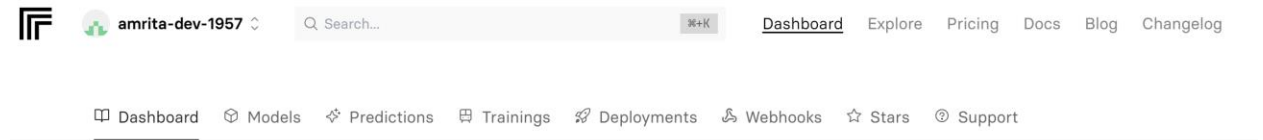
[+ Create API Key](#)

3. Access to Replicate:

To use the replicate.com service for deploying models like Llama 2, you'll need to adapt your setup to interact with their API. This involves setting up an account, deploying your

model on Replicate, and then writing the code to make API calls to interact with the deployed model.

Website link: <https://replicate.com/>



amrita-dev-1957

Search...

Dashboard Explore Pricing Docs Blog Changelog

Dashboard Models Predictions Trainings Deployments Webhooks Stars Support

Your usage so far this month has a total cost of \$0.03. View your invoices.

Recent predictions

ID	Model	Source	Status	Run time	Created
kgvp1b0fxdrq80cfmmj8v0ckj0	meta/llama-2-13b-chat	API	Succeeded	0.8 seconds	19 hours ago
9dy5phkdv1rg80cfmmctqesd9w	meta/llama-2-13b-chat	API	Succeeded	1.0 seconds	20 hours ago
hrgh811125rgc0cfmkxb49m75w	meta/llama-2-13b-chat	API	Succeeded	2.0 seconds	20 hours ago
0hg19jgk1rge0cfmkxbvmj80c	meta/llama-2-13b-chat	API	Succeeded	2.1 seconds	20 hours ago
j62v6j08kdrq0cfmkxbsen21w	meta/llama-2-13b-chat	API	Succeeded	2.1 seconds	20 hours ago
e9bfq5rj61rga0cfmkrsrxn0c	meta/llama-2-13b-chat	API	Succeeded	2.4 seconds	20 hours ago
9nhqz6wvh1rga0cfmjwbtb5krqc	meta/llama-2-13b-chat	API	Succeeded	7.3 seconds	21 hours ago
04v5katp2xrqe0cfmjwrsb3698	meta/llama-2-13b-chat	API	Succeeded	0.1 seconds	21 hours ago

4. Setup Environment:

Install necessary dependencies and libraries. For running large language models like Llama 2, you typically need Python, PyTorch or TensorFlow, and other required packages.

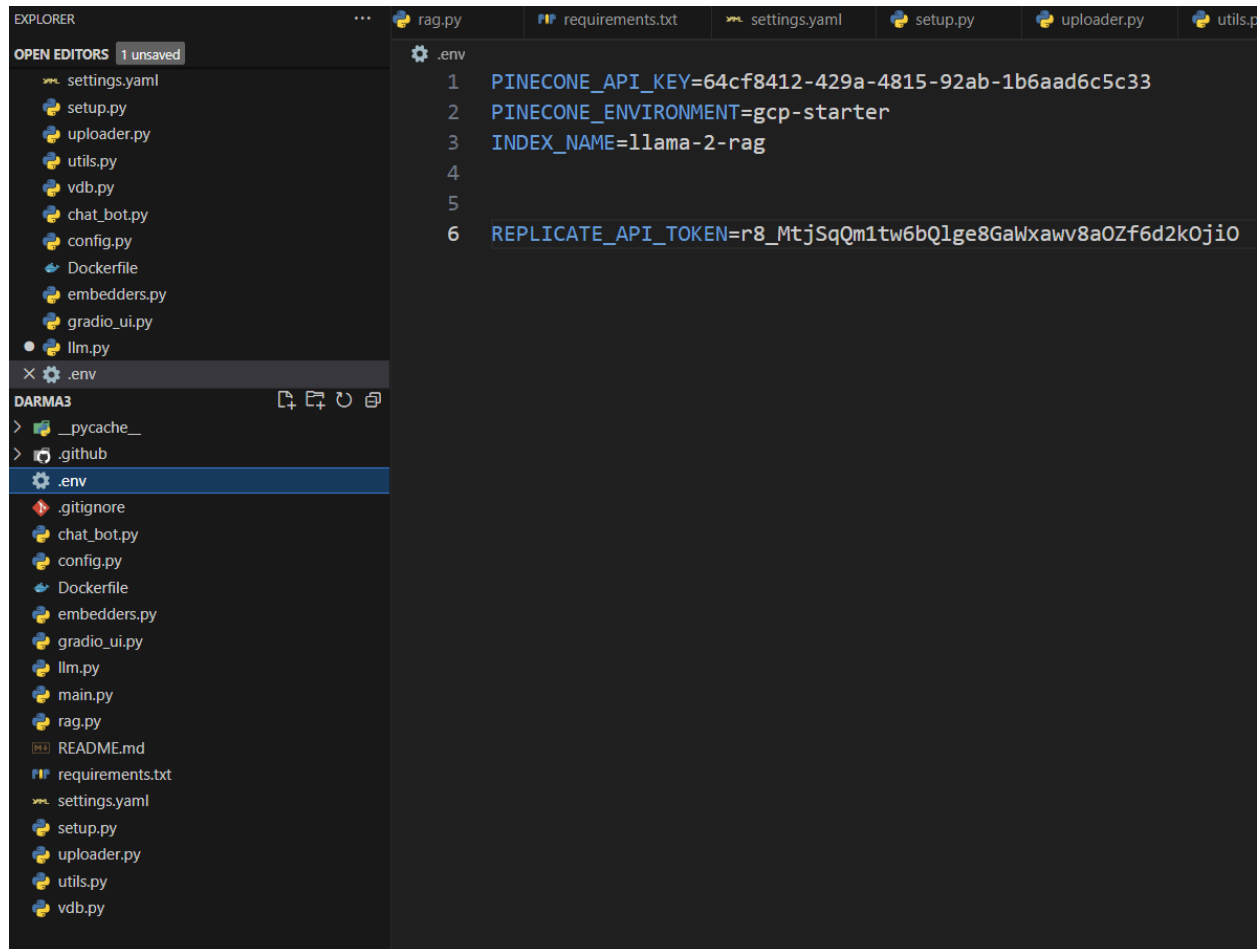
1.Ensure you have Python installed. Then, create a virtual environment and install the required packages.

```
# Create a virtual environment
python -m venv llama2_env

# Activate the virtual environment
# On Windows
.\llama2_env\Scripts\activate
# On macOS/Linux
source llama2_env/bin/activate

# Install necessary packages
pip install torch transformers
```

2. Create a .env file which consist of API

A screenshot of a code editor interface. The top bar shows several open files: rag.py, requirements.txt, settings.yaml, setup.py, uploader.py, and utils.p. The left sidebar has two panels. The 'EXPLORER' panel shows a file tree with folders like 'DARMA3' and 'utils', and various Python files. The 'OPEN EDITORS' panel shows a list of open files, with '.env' selected. The main editor area displays the content of the '.env' file, which contains six lines of environment variables. The first five lines are commented out, and the sixth line is active.

```
1 PINECONE_API_KEY=64cf8412-429a-4815-92ab-1b6aad6c5c33
2 PINECONE_ENVIRONMENT=gcp-starter
3 INDEX_NAME=llama-2-rag
4
5
6 REPLICATE_API_TOKEN=r8_MtjSqQm1tw6bQlge8Gawxawv8a0Zf6d2k0ji0
```

3. Download and Load Llama 2

Use the Hugging Face transformers library to download and load the model.

```
from transformers import AutoModelForCausalLM, AutoTokenizer

# Replace 'model-name' with the actual model name if available on Hugging Face
model_name = "meta-llama/LLaMA-2-7B"

# Load the tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
```

4. Generate Text

Write a script to generate text using the model.

```
import torch

class LLaMA2:

    def __init__(self, model_name):

        self.model_name = model_name

        self.tokenizer = AutoTokenizer.from_pretrained(model_name)

        self.model = AutoModelForCausalLM.from_pretrained(model_name)

    def generate(self, prompt, max_length=100, temperature=0.7):

        inputs = self.tokenizer(prompt, return_tensors="pt")

        outputs = self.model.generate(

            inputs.input_ids,

            max_length=max_length,

            temperature=temperature,

            num_return_sequences=1

        )

        return self.tokenizer.decode(outputs[0], skip_special_tokens=True)

# Example usage

llama2 = LLaMA2("meta-llama/LLaMA-2-7B")

prompt = "What is the capital of France?"

response = llama2.generate(prompt)

print(response)
```

5. Integrate with Your Application

You can integrate the above logic into your Llama application. Here's how you can adapt your existing Replicate LLM class to work with Llama 2:

```
class LLaMA2LLM:

    def __init__(self, model_name, system_prompt, **kwargs):

        self.messages = []

        self.system_prompt = system_prompt

        self.prompt = ""

        self.kwargs = kwargs
```

```

self.model_name = model_name

self.tokenizer = AutoTokenizer.from_pretrained(model_name)

self.model = AutoModelForCausalLM.from_pretrained(model_name)

def add_message(self, msg):
    if len(self.messages) > 9:
        self.messages = self.messages[2:]
    self.messages.append(msg)
    self.prompt = self.get_prompt_from_messages()

def get_prompt_from_messages(self):
    prompt = ""
    for msg in self.messages:
        if msg["isUser"]:
            prompt += f"[INST] {msg['message']} [/INST] \n"
        else:
            prompt += msg["message"] + "\n"
    return prompt

def respond(self, query):
    self.add_message({"isUser": True, "message": query})

    input_prompt = f"{self.system_prompt}\n{self.prompt}"
    inputs = self.tokenizer(input_prompt, return_tensors="pt")
    outputs = self.model.generate(
        inputs.input_ids,
        max_length=self.kwarg.get("max_new_tokens", 500),
        temperature=self.kwarg.get("temperature", 0.75),
        top_p=self.kwarg.get("top_p", 1),
        repetition_penalty=self.kwarg.get("repetition_penalty", 1),
        num_return_sequences=1
    )
    model_response = self.tokenizer.decode(outputs[0], skip_special_tokens=True)
    self.add_message({"isUser": False, "message": model_response})
    return model_response

# Example usage
llm = LLaMA2LLM("meta-llama/LLaMA-2-7B", "You are a helpful tutor assistant.", temperature=0.7, top_p=0.9, max_new_tokens=300)

```

```
response = llm.respond("What is the capital of France?")  
print(response)
```

Gradio: Integrating Gradio for the frontend. Gradio provides an easy-to-use interface for building interactive web applications for machine learning models. Here's how you can integrate Gradio with your ReplicateLLM backend.

Steps to Integrate Gradio

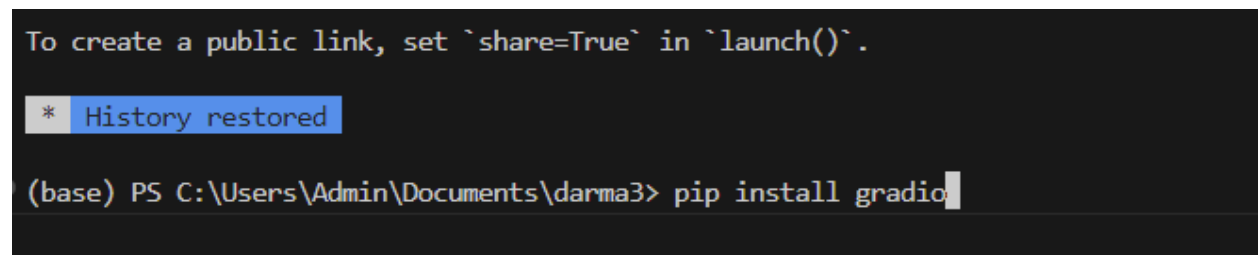
1. Set Up Gradio:

- Install Gradio using pip if you haven't already.
- Create a Gradio interface that interacts with your ReplicateLLM class.
- Create Gradio Interface:

2. Define a function that will take user input, use your ReplicateLLM class to generate a response, and return that response.

3. Set up the Gradio interface to call this function.

Link to Gradio Website: <https://www.gradio.app/>

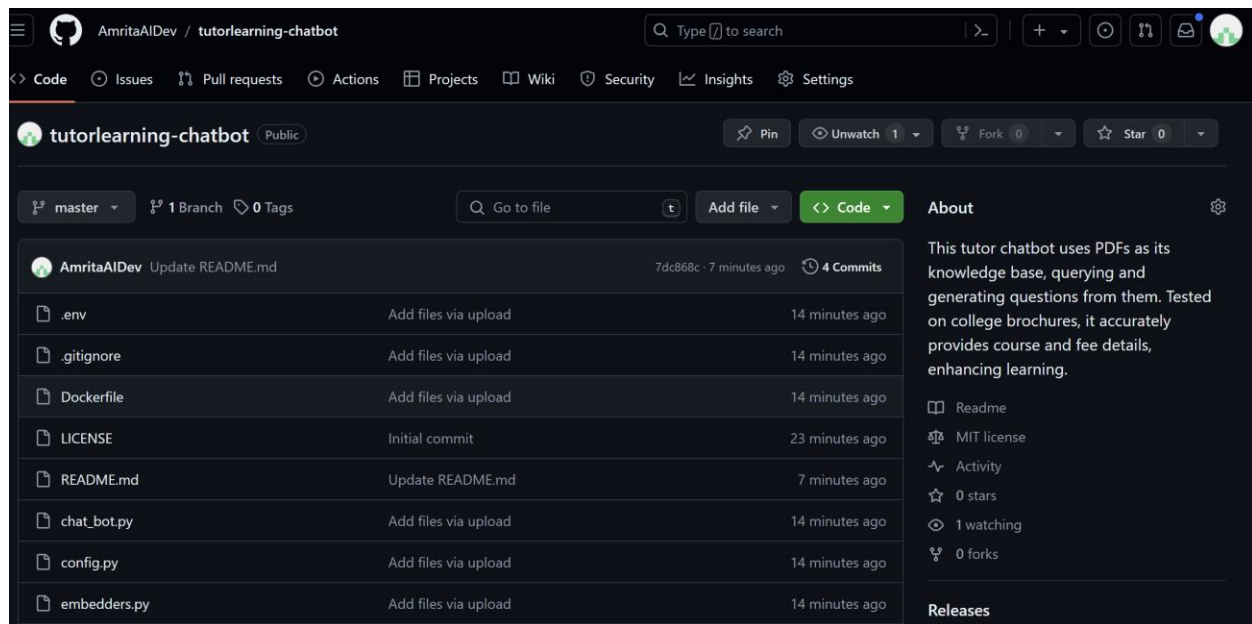


```
To create a public link, set `share=True` in `launch()`.  
* History restored  
(base) PS C:\Users\Admin\Documents\darma3> pip install gradio
```

Key Points:

- Environment Setup: Ensure all necessary dependencies are installed.
- Model Loading: Load the Llama 2 model and tokenizer using transformers.
- Text Generation: Use the model to generate text based on the provided prompt.
- Integration: Integrate the model into your application logic for generating questions and providing responses.

GitHub Link: <https://github.com/AmritaAIDev/tutorlearning-chatbot>



Task performed:

- Asking queries based on uploaded PDFs as knowledge source.

Result:

- We experimented with various college brochures, each representing different institutions, and queried information about courses, fees, and related details. In the majority of cases, we obtained satisfactory and precise responses.

Task performed:

- Generating questions based on certain Course Materials

Results:

- We uploaded supplementary materials for certain courses such as ASPICE, SCS 9001, etc. The model is generating questions beyond the scope of the given topic, and in some instances, it fails to generate questions altogether.