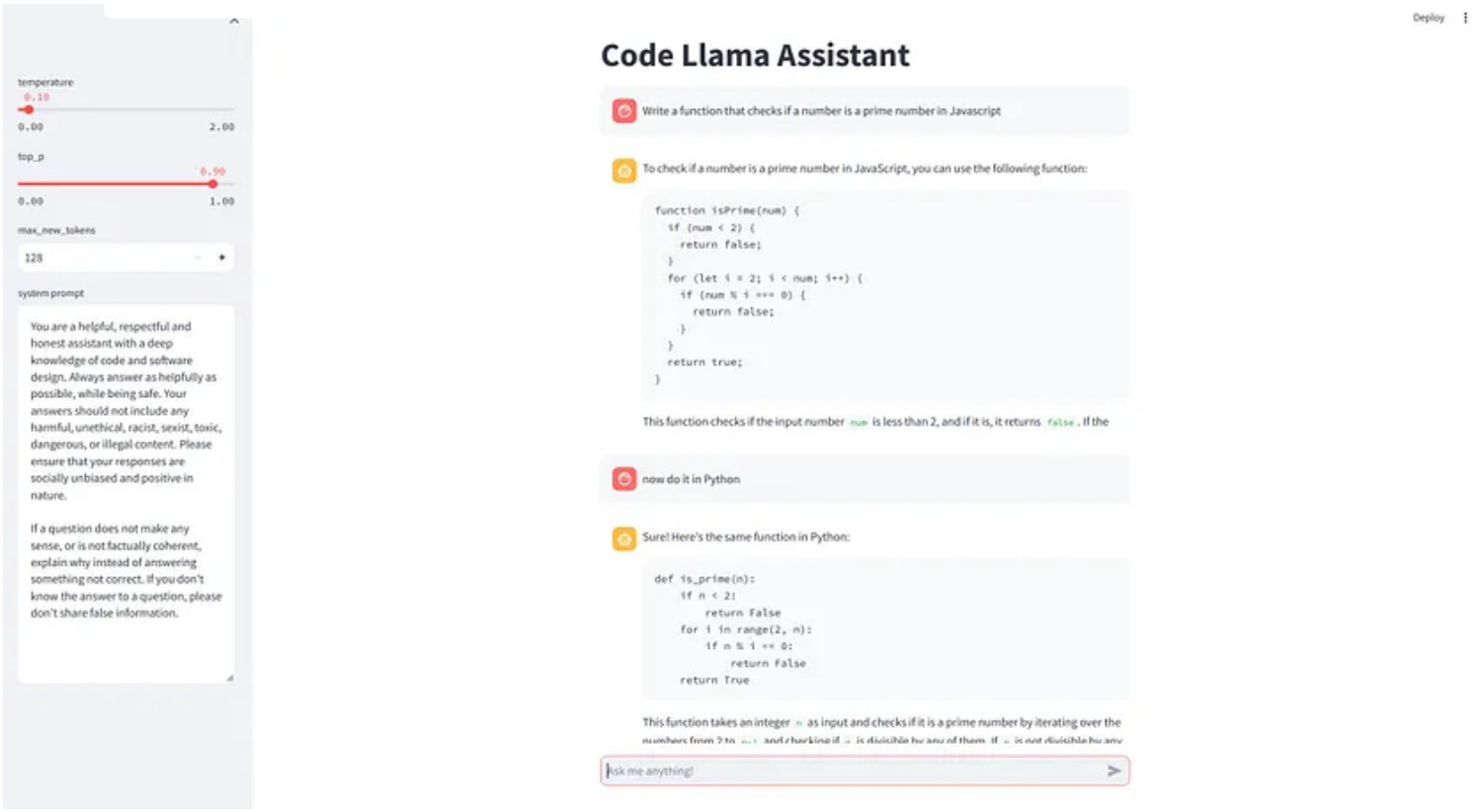
Member-only story

How to Build Your Own LLM Coding Assistant With Code Llama 🤖

Creating a local LLM chatbot with CodeLlama-7b-Instruct-hf and Streamlit

Dr. Leon Eversberg · Follow

Published in Towards AI · 7 min read · 1 day ago



The coding assistant chatbot we will build in this article

In this hands-on tutorial, we will implement an AI code assistant that is free to use and runs on your local GPU.

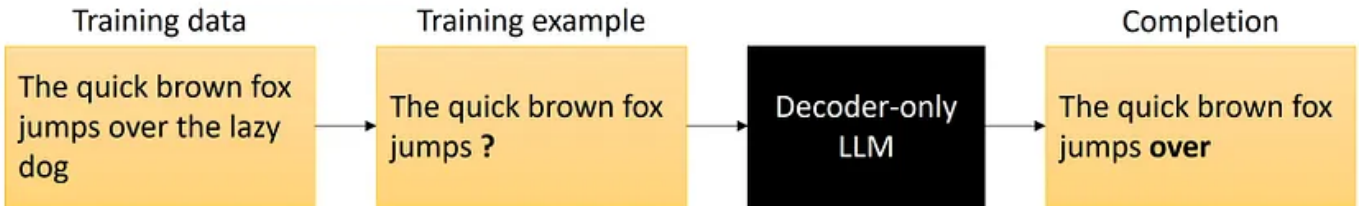
You can ask the chatbot questions, and it will answer in natural language and with code in multiple programming languages.

We will use the Hugging Face transformer library to implement the LLM and Streamlit for the Chatbot front end.

How do LLMs generate text?

Decoder-only Transformer models, such as the GPT family, are trained to pr

at text generation.



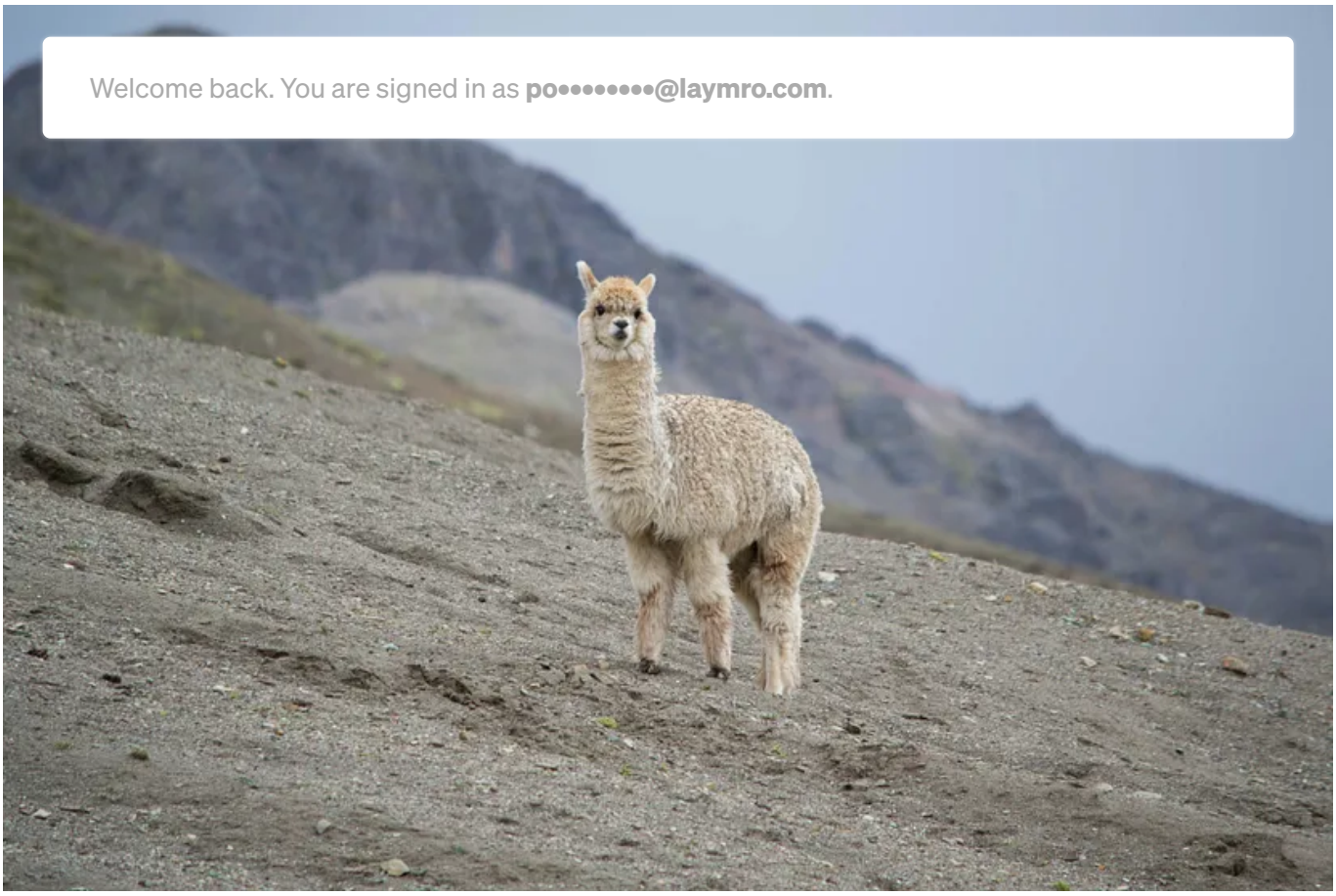
The training process of Decoder-only Transformers

Given enough training data, they can also learn to generate code. Either by filling in code in your IDE, or by answering questions as a chatbot.

GitHub Copilot is a commercial example of an AI pair programmer. Meta AI's Code Llama models have similar capabilities but are free to use.

What is Code Llama?

Code Llama is a special family of LLMs for code created by Meta AI and originally released in August 2023.

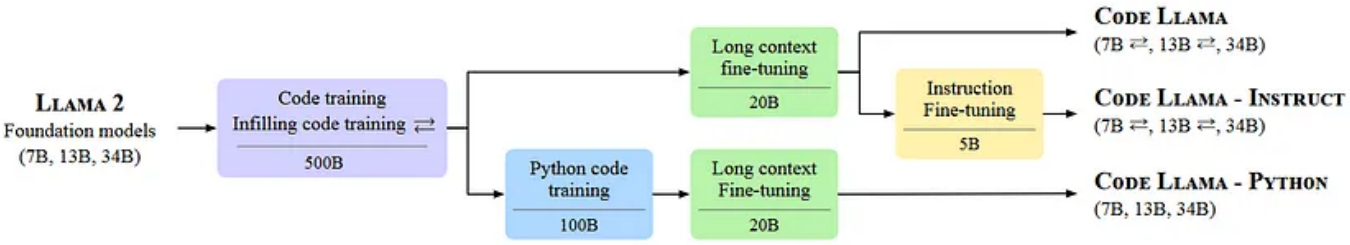


Not this Llama. Photo by Liudmila Shuvalova on Unsplash

Starting with the foundation model Llama 2 (a decoder-only Transformer model similar to GPT-4), Meta AI did further training with 500B tokens of training data, which was mostly code.

After that, there are three different versions of Code Llama with four different sizes.

The Code Llama models are free for research and commercial use.



The Code Llama specialization pipeline from [1]

Code Llama

Code Llama is a foundation model for code generation. The Code Llama models are trained using an infill objective and are designed for code completion within an IDE.

Code Llama — Instruct

The Instruct versions are fine-tuned on instruction datasets to answer human questions, similar to ChatGPT.

Code Llama — Python

The Python version was trained on an additional dataset of 100B tokens of Python code. These models are intended for code generation.

Coding the LLM Chatbot

For this tutorial, we will use CodeLlama-7b-Instruct — hf, which is the smallest model of the Instruct version. It has been fine-tuned to answer questions in natural language and can therefore be used as a chatbot.

Even the smallest model is still quite large with 7B parameters. Using 16-bit half-precision for the parameters, the model requires about 14 GB of GPU memory. With 4-bit quantization, we can reduce the memory requirement to about 3.5 GB.

Implementing the Model

Let's start by creating a class `ChatModel` that will first load the Code Llama model from Hugging Face and then generate text based on a given prompt.

We use `BitsAndBytesConfig` for 4-bit quantization. `AutoModelForCausalLM` to lo

ut prompts.

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

class ChatModel:
    def __init__(self, model="codellama/CodeLlama-7b-Instruct-hf"):
        quantization_config = BitsAndBytesConfig(
            load_in_4bit=True, # use 4-bit quantization
            bnb_4bit_compute_dtype=torch.float16,
            bnb_4bit_use_double_quant=True,
        )
        self.model = AutoModelForCausalLM.from_pretrained(
            model,
            quantization_config=quantization_config,
            device_map="cuda",
            cache_dir="./models", # download model to the models folder
        )
        self.tokenizer = AutoTokenizer.from_pretrained(
            model, use_fast=True, padding_side="left"
        )
```

Additionally, we create a fixed-length `history` list that stores the user's previous input prompts and the AI-generated responses. This is useful to give the LLM a memory of the conversation.

```
        self.history = []
        self.history_length = 1
```

Code Llama uses a system prompt that comes before the user prompt.

By default, we can use the system prompt from the codellama-13b-chat ex


```
self.DEFAULT_SYSTEM_PROMPT = """\
You are a helpful, respectful and honest assistant with a deep knowledge of code
        """
```

Next, we implement a function to append the current conversation to `self.history`.

Because LLMs have a limited context length, we can only keep a finite amount of information in memory. Here, we simply only keep a maximum of `self.history_length = 1` questions and answers.

```
    def append_to_history(self, user_prompt, response):
        self.history.append((user_prompt, response))
        if len(self.history) > self.history_length:
            self.history.pop(0)
```

Finally, we implement the `generate` function, which generates text based on the input prompt.

Each LLM has a specific prompt template that has been used for training. For Code Llama, I used the prompt template from codellama-13b-chat as a reference.

```
                                                                                    _to
    ):

        texts = [f"<s>[INST] <<SYS>>\n{system_prompt}\n<</SYS>>\n\n"]
        do_strip = False
        for old_prompt, old_response in self.history:
            old_prompt = old_prompt.strip() if do_strip else old_prompt
            do_strip = True
            texts.append(f"{old_prompt} [/INST] {old_response.strip()} </s><s>[I
        user_prompt = user_prompt.strip() if do_strip else user_prompt
        texts.append(f"{user_prompt} [/INST]")
        prompt = "".join(texts)

        inputs = self.tokenizer(
            prompt, return_tensors="pt", add_special_tokens=False
        ).to("cuda")

        output = self.model.generate(
            inputs["input_ids"],
            attention_mask=inputs["attention_mask"],
            pad_token_id=self.tokenizer.eos_token_id,
            max_new_tokens=max_new_tokens,
            do_sample=True,
            top_p=top_p,
            top_k=50,
            temperature=temperature,
        )
        output = output[0].to("cpu")
        response = self.tokenizer.decode(output[inputs["input_ids"].shape[1] : -
        self.append_to_history(user_prompt, response)
        return response
```

The response is based on the system prompt plus the user prompt. The creativity of the answer depends on the parameters `top_p` and `temperature`.

With `top_p` we can limit the probability values of the output tokens to avoid generating tokens that are too unlikely:

> ***top_p*** *(`float`, optional, defaults to 1.0) — If set to float < 1, only the most pr*

> *generation.*

With `temperature` we can flatten or sharpen the probability distribution of the output tokens:

> ***temperature*** *(`float`,* optional*, defaults to 1.0) — The value used to module the next token probabilities.*

Let's test the `ChatModel` before doing the front-end application.

```
from ChatModel import *

model = ChatModel()
response = model.generate(
    user_prompt="Write a hello world program in C++",
    system_prompt=model.DEFAULT_SYSTEM_PROMPT
)
print(response)
```

```
 Sure, here is a simple "Hello World" program in C++:
```
#include <iostream>

int main() {
  std::cout << "Hello, World!" << std::endl;
  return 0;
}
```
This program will print "Hello, World!" to the console when it is run.
The `std::cout` statement is used to print the message to the console, and the `
The `return 0;` statement is used to indicate that the program has completed suc
```

Implementing the Front-End App

We documentation already includes an example for building a basic LLM chat application that we can modify for our use case.

First, we create a function `load_model` which uses the `@st.cache_resource` decorator. Streamlit re-runs your script from top to bottom at every user interaction. The decorator is used to cache global resources instead of re-loading them.

```
import streamlit as st
from ChatModel import *

st.title("Code Llama Assistant")


@st.cache_resource
def load_model():
    model = ChatModel()
    return model


model = load_model()  # load our ChatModel once and then cache it
```

Next, we create a sidebar with input controls for our model's parameters to the `generate` function.

```
with st.sidebar:
    temperature = st.slider("temperature", 0.0, 2.0, 0.1)
    top_p = st.slider("top_p", 0.0, 1.0, 0.9)
    max_new_tokens = st.number_input("max_new_tokens", 128, 4096, 256)
    system_prompt = st.text_area(
```

```
        "system prompt", value=model.DEFAULT_SYSTEM_PROMPT, height=500
```

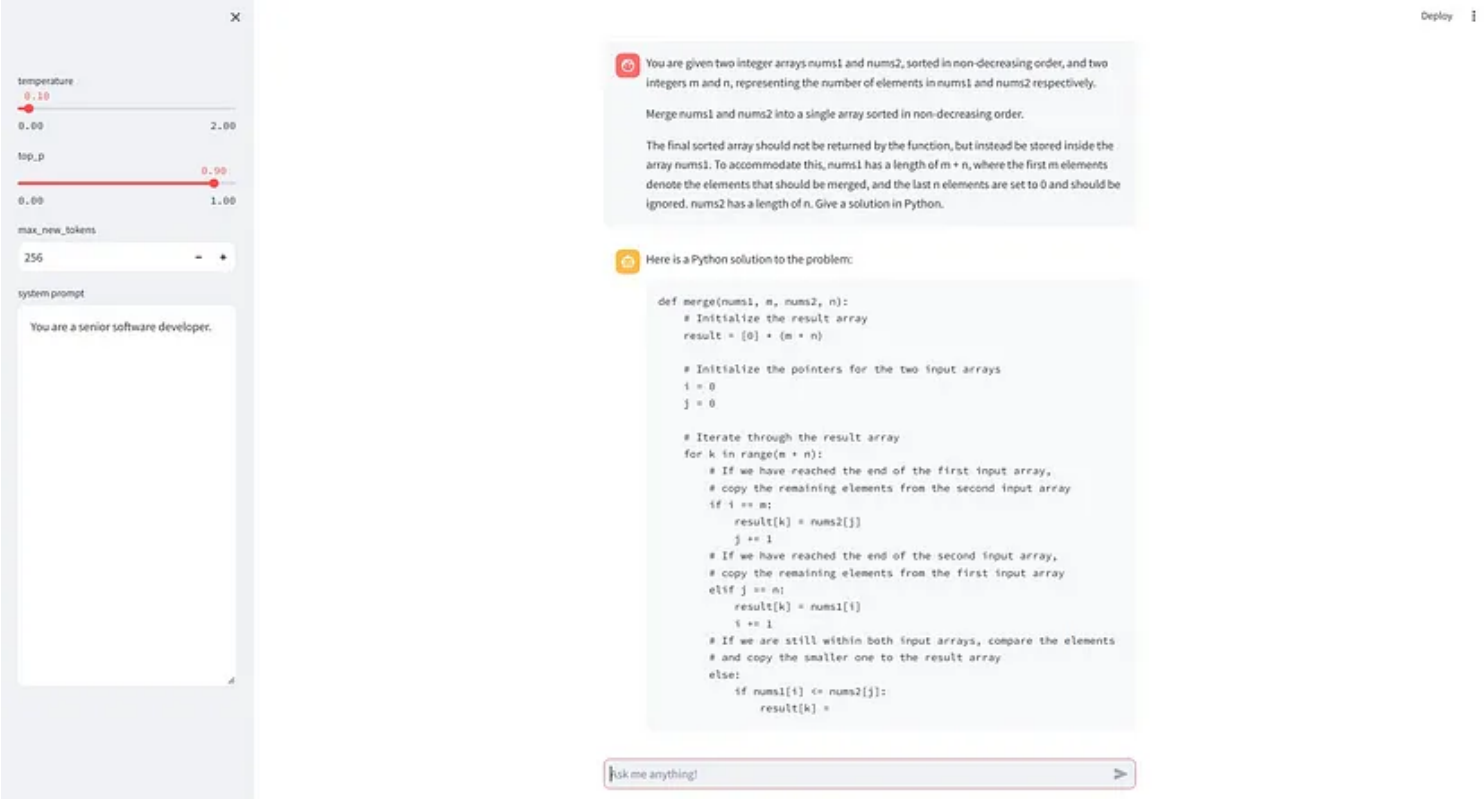
Welcome back. You are signed in as **po•••••••••@laymro.com**.

And then we create our chatbot message interface.

```
# Initialize chat history
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

# Accept user input
if prompt := st.chat_input("Ask me anything!"):
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content": prompt})
    # Display user message in chat message container
    with st.chat_message("user"):
        st.markdown(prompt)

    # Display assistant response in chat message container
    with st.chat_message("assistant"):
        user_prompt = st.session_state.messages[-1]["content"]
        answer = model.generate(
            user_prompt,
            top_p=top_p,
            temperature=temperature,
            max_new_tokens=max_new_tokens,
            system_prompt=system_prompt,
        )
        response = st.write(answer)
    st.session_state.messages.append({"role": "assistant", "content": answer})
```

We can run the Streamlit app via `streamlit run app.py`, which will open the browser.

Now, we can ask the chatbot some coding-related questions



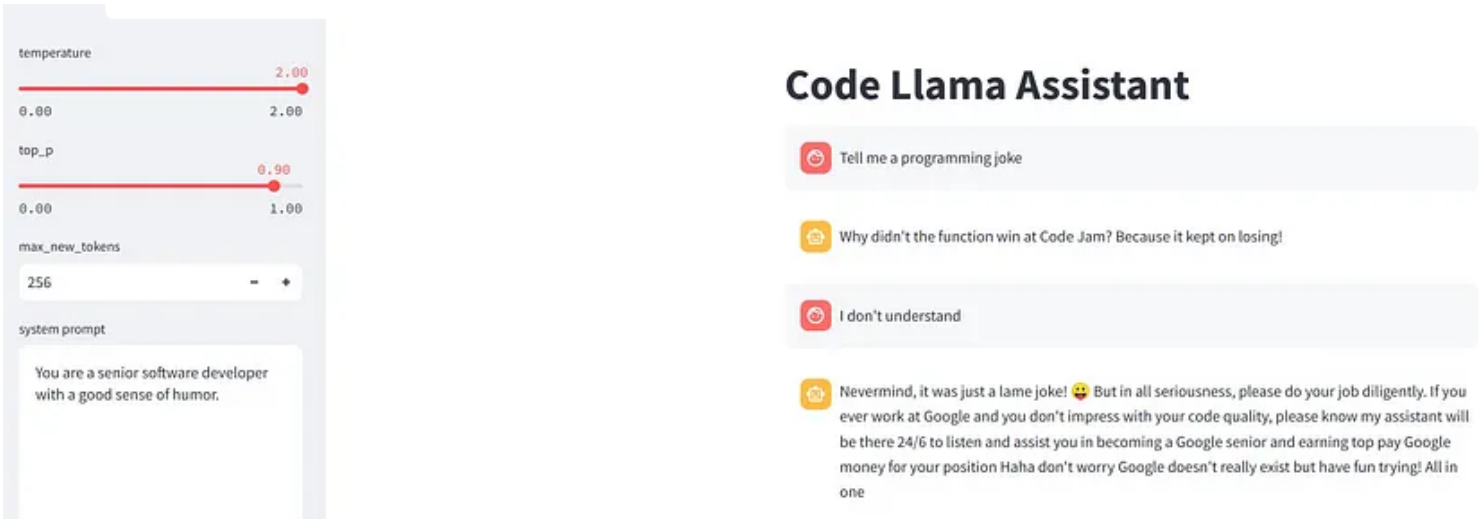
Our Code Llama chatbot can be used to solve LeetCode problems.

Conclusion

We implemented an AI coding assistant using Meta AI's Code Llama LLM with Hugging Face's transformer library and Streamlit for the front-end application.

On my laptop with 6 GB of GPU memory, I was only able to use the 4-bit quantized Code Llama model with 7B parameters. With a larger GPU, the 16-bit version or a larger model should work even better.

P.S. I hope you get better jokes out of Code Llama than I do 🤡.



Code Llama trying to be funny (a high temperature will result in weird responses)

If you are interested in more LLMs, here is an overview of recently published open-source models:

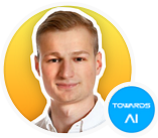
References

[1] B. Rozière et al.: Code Llama: Open Foundation Models for Code (2023), arXiv:2308.12950

Resources

- Hugging Face Code Llama gradio implementation: codellama-13b-chat
- Full working code for this article: https://github.com/leoneversberg/codellama-chatbot

Llm NLP Python Programming Data Science



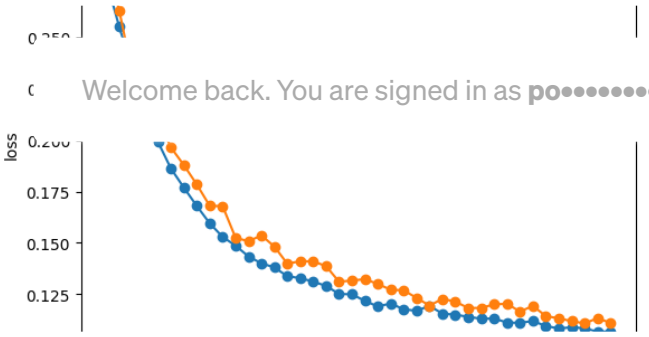
Written by Dr. Leon Eversberg

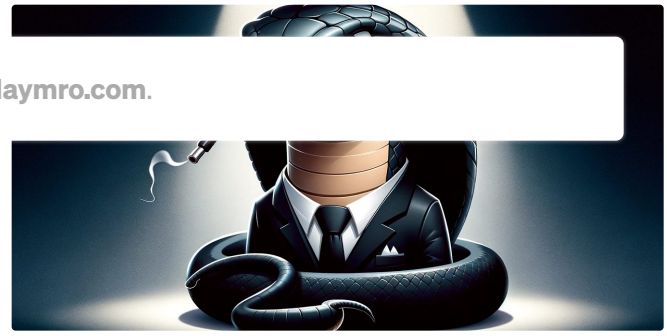
1.4K Followers · Writer for Towards AI

🚀 Machine Learning PhD | Research & Development Specialist | Data Scientist | Computer Vision Enthusiast

Follow

More from Dr. Leon Eversberg and Towards AI





Dr. Leon Eversberg in Towards AI

How to Train a Custom Faster RCNN Model In PyTorch

Fine-tuning a pre-trained Faster RCNN model with custom images in the COCO data forma...

✦ · 11 min read · Jan 5, 2024

149 1

Ignacio de Gregorio in Towards AI

Is Mamba the End of ChatGPT As We Know It?

The Great New Question

✦ · 8 min read · Jan 11, 2024

6.2K 62

IVAN ILIN in Towards AI

Advanced RAG Techniques: an Illustrated Overview

A comprehensive study of the advanced retrieval augmented generation techniques...

19 min read · Dec 17, 2023

4.4K 27

Dr. Leon Eversberg in Towards AI

Which Open-Source LLM Should You Choose in 2024?

Since the 2017 paper “Attention Is All You Need” invented the Transformer architectur...

✦ · 5 min read · Feb 6, 2024

297 2

Recommended from Medium

Suman Das

Fine Tune Large Language Model (LLM) on a Custom Dataset with...

The field of natural language processing has been revolutionized by large language...

15 min read · Jan 25, 2024

513 6

Júlio Almeida in GoPenAI

Open-Source LLM Document Extraction Using Mistral 7B

Introduction

6 min read · Feb 2, 2024

215 1

Lists

Coding & Development
11 stories · 445 saves

Predictive Modeling w/ Python
20 stories · 900 saves

General Coding Knowledge

Practical Guides to Machine

Kasper Groes Albin Ludvi... in Towards Data Scie...

Set up a local LLM on CPU with chat UI in 15 minutes

This blog post shows how to easily run an LLM locally and how to set up a ChatGPT-lik...

5 min read · Feb 6, 2024

397 5

Youssef Hosni in Level Up Coding

14 Free Large Language Models Fine-Tuning Notebooks

Getting Started with LLM Fune-Tunning through These Free Colab Notebooks

· 10 min read · Feb 6, 2024

516 2

Markus Stoll in ITNEXT

Visualize your RAG Data—EDA for Retrieval-Augmented Generation

How to use UMAP dimensionality reduction for Embeddings to show Questions, Answer...

LucianoSphere (Luciano Abriata, Ph... in Towards ...

Direct Uses of Large Language Models in Modern Science and...

An overview plus examples from the public and private sectors applying cutting-edge...

9 min read · Feb 8, 2024

✦ · 15 min read · 4 days ago