Llama2 and Text Summarization

Tushit Dave · Follow
9 min read · Sep 9

Listen Share More

Unlocking the Power of Llama2 for Local Multi-Document Summarization

This marks my third article exploring the realm of "**Text Summarization**", where I've employed a variety of methodologies to achieve effective abstract Summarization across multiple documents. In a previous article, I delved into the application of

Llama-Index in conjunction with **GPT3.5 Turbo**, which you can find through the following link:

Multiple Document Summary and LLM Powered QA-System

In this blog post, we will discuss how we can summarize multiple documents and develop a summary using Llama-Index and...

medium.com

Introduction to Text Summarization:

As We all know, Text summarization is a crucial task in natural language processing that helps extract the most important information from a given document or text while retaining its core significance. In recent years, various techniques and models have been developed to automate this process, making it easier for individuals and businesses to digest large volumes of textual data. Here I am proposing a solution using Llama2 locally without using any cloud services, and you can deploy the same onto your local server or machine without exposing your documents to any third-party applications or OpenAI's Models. We will explore the capabilities of Llama2 and demonstrate how it can streamline your multiple document summarization needs.

Text Summarization using Llama2

Now, let’s go over how to use Llama2 for text summarization on several documents locally:

> *Installation and Code:*

To begin with, we need the following pre-requisites:

```
Natural Language Processing
!pip install langchain==0.0.191
!pip install llama-cpp-python==0.1.66
!pip install sentence-transformers
!pip install huggingface_hub
!pip install auto-gptq==0.2.2
# !pip install termcolor
```

In order to execute the Llama2 model on your local system, you will require llama-cpp (Llama C++), which can be easily installed via pip.

Additionally, you need to have huggingface_hub installed to access the Hugging Face repository and download the necessary model.

Another essential component is Auto-GPTQ, which serves as a crucial framework to run a quantized model. It can be viewed as a foundational framework that provides essential support for this purpose.

These are the frameworks I’ve successfully imported to build tokenization: AutoGPTQ for Causal LLM, Pipelines, Transformer’s Auto and Longformer Tokenizer, and, most significantly, Langchain and its essential modules for future use.

```
            ("logging", logging),
            ("click", click),
            ("torch", torch),
            ("transformers", transformers),
            ("os", os),
            ("re", re),
            ("shutil", shutil),
            ("subprocess", subprocess),
```

```
        ("requests", requests),
        ("pathlib", Path),
        ("auto_gptq", AutoGPTQForCausalLM),
        ("huggingface_hub", hf_hub_download),
        ("huggingface_instruct_embeddings", HuggingFaceInstructEmbeddings),
        ("langchain_pipeline", HuggingFacePipeline),
        ("llama_cpp", LlamaCpp),
        ("prompt_template", PromptTemplate),
        ("llm_chain", LLMChain),
        ("transformers_auto_tokenizer", AutoTokenizer),
        ("transformers_auto_model", AutoModelForCausalLM),
        ("transformers_generation_config", GenerationConfig),
        ("transformers_llm_model", LlamaForCausalLM),
        ("transformers_llm_tokenizer", LlamaTokenizer),
        ("transformers_longformer_tokenizer", LongformerTokenizer),
        ("transformers_pipeline", pipeline),
        ("rouge", Rouge),
        ("text_splitter", RecursiveCharacterTextSplitter),
        ("tqdm", tqdm),
        ("termcolor_colored", colored),
```

Let's begin to understand each framework we imported above and its significance and usage:

1. ***Logging:*** The logging module is responsible for producing log messages. It is used in this situation to record information about the loading process, such as model details and progress updates.

2. ***hf_hub_download***:The Hugging Face Hub library has this function. It is used to download model files from a Hugging Face model repository, depending on the repository ID and file name that are provided.

3. ***AutoTokenizer***: This is a Hugging Face Transformers library class. It is used to tokenize input text, which implies breaking it down into smaller units that the model can comprehend, such as words or subwords.

4. ***AutoGPTQForCausalLM, AutoModelForCausalLM***: These are Transformer library classes that represent pre-trained language models. For *quantized models*, the "*AutoGPTQForCausalLM* "class is used, whereas the "*AutoModelForCausalLM*" class is used for *complete models*.

5. *GenerationConfig*: This class is from Transformers and is used to configure text generation settings for the model.

6. ***pipeline: To create a text-generating pipeline, use this Transformers function.*** It handles tokenization, model inference, and other parameters to simplify the process of creating text from a language model.

7. ***LlamaCpp, LlamaTokenizer***: These classes, which are part of the Llama library, are used to work with quantized language models.***The LlamaCpp*** class is for quantized models, and tokenization is handled by ***LlamaTokenizer.***

The logical flow within the *load_model* function:

- The function begins by *logging* information about the loaded model and the target device (e.g., CPU or GPU).
- It determines if a ***model_basename*** is given. If so, it decides whether the model is quantized (e.g., with a ".ggml" extension) or full.
- It downloads the model's file from the Hugging Face model repository using the *hf_hub_download* function for quantized models. The model's parameters, such as context size and token limitations, are then configured, and an instance of the *LlamaCpp* class is returned.
- It uses *AutoTokenizer* and *AutoModelForCausalLM* to initialize a tokenizer and a language model for complete models (non-quantized). It also configures the various model settings.
- If no model_basename is specified, it examines the device_type and either initializes a quantized model using *LlamaTokenizer* and *LlamaForCausalLM* or a complete model using *AutoTokenizer* and *AutoModelForCausalLM*.
- Using the pipeline function, the function creates a text generation *pipeline* that encompasses the model, tokenizer, and generation parameters.
- In the end, it returns the configured text generation pipeline.

```
def load_model(device_type, model_id, model_basename=None):

    logging.info(f"Loading Model: {model_id}, on: {device_type}")
    logging.info("This action can take a few minutes!")

if model_basename is not None:
        if ".ggml" in model_basename:
logging.info (Using Llamacpp for GGML quantized models")
            model_path = hf_hub_download(repo_id=model_id, filename=model_basename
            max_ctx_size = 4096
            kwargs = {
                "model_path": model_path,
                "n_ctx": max_ctx_size,
                "max_tokens": max_ctx_size,
            }
            if device_type.lower() == "mps":
                kwargs["n_gpu_layers"] = 1000
            if device_type.lower() == "cuda":
                kwargs["n_gpu_layers"] = 1000
                kwargs["n_batch"] = max_ctx_size
            return LlamaCpp(**kwargs)

        else:
            logging.info("Using AutoGPTQForCausalLM for quantized models")

            if ".safetensors" in model_basename:
                # Remove the ".safetensors" ending if present
                model_basename = model_basename.replace(".safetensors", "")

            tokenizer = AutoTokenizer.from_pretrained(model_id, use_fast=True)
            logging.info("Tokenizer loaded")

            model = AutoGPTQForCausalLM.from_quantized(
                model_id,
                model_basename=model_basename,
                use_safetensors=True,
                trust_remote_code=True,
                device="cuda:0",
                use_triton=False,
                quantize_config=None,
            )
    elif (
        device_type.lower() == "cuda"
    ):
logging.info (Using AutoModelForCausalLM for full models")
        tokenizer = AutoTokenizer.from_pretrained(model_id)
        logging.info("Tokenizer loaded")
```

```
            model = AutoModelForCausalLM.from_pretrained(
                model_id,
                device_map="auto",
                torch_dtype=torch.float16,
                low_cpu_mem_usage=True,
                trust_remote_code=True,
                # max_memory={0: "15GB"} # Uncomment this line with you encounter CUD/
            )
            model.tie_weights()
        else:
            logging.info("Using LlamaTokenizer")
            tokenizer = LlamaTokenizer.from_pretrained(model_id)
            model = LlamaForCausalLM.from_pretrained(model_id)

        generation_config = GenerationConfig.from_pretrained(model_id)

    Create a pipeline for text generation
        pipe = pipeline(
            "text-generation",
            model=model,
            tokenizer=tokenizer,
            max_length=2048,
            temperature=0,
            top_p=0.95,
            repetition_penalty=1.15,
            generation_config=generation_config,
        )

        local_llm = HuggingFacePipeline(pipeline=pipe)
    logging.info (Local LLM Loaded")

        return local_llm
```

In brief, this function loads language models, either quantized or complete, configures them, and sets up a text generation pipeline to generate text based on the loaded model. It accomplishes these tasks efficiently by utilizing numerous modules from the Transformers and Llama libraries.

We will introduce the model to the local device now that we have seen it above. The code below will determine whether the GPU or CPU is available. To run the loaded model further, Device_Type will be assigned:

```
DEVICE_TYPE = "cuda" if torch.cuda.is_available() else "cpu"
SHOW_SOURCES = True
logging.info(f"Running on: {DEVICE_TYPE}")
logging.info(f"Display Source Documents set to: {SHOW_SOURCES}")
```

Now, we will opt for the 7B-Chat model for our application, as I have limited GPU resources and cannot accommodate larger models like the 13B or 70B variants.

```
model_id = "TheBloke/Llama-2-7B-Chat-GGML"
model_basename = "llama-2-7b-chat.ggmlv3.q4_0.bin"
```

I'd like to express my gratitude to TheBloke for their efforts in converting all ".HF" formats to ".GGML." You are welcome to explore the repository at your convenience by visiting https://huggingface.co/TheBloke.

Now call the load_model function:

```
LLM = load_model(device_type=DEVICE_TYPE, model_id=model_id, model_basename=model_
```

So far, so good.

To begin, we need numerous documents, each with over 10,000 tokens. These documents will provide the foundation for creating summaries. To begin the process, we will use the Wikipedia API to retrieve Wonder City-related data.This large collection of lengthy documents will allow us to investigate robust summarizing strategies. Using

	wonder_city	information	num_tokens
0	Beirut	Beirut is the capital and largest city of Le...	12695
1	Doha	Doha (Arabic: الدوحة, romanized: ad-Dawḥa [ad'...	9016
2	Durban	Durban (DUR-bən) (Zulu: eThekwini, from ithek...	7992
3	Havana	Havana (; Spanish: La Habana [la a'βana] ; Luc...	30000
4	Kuala Lumpur	Kuala Lumpur (Malaysian pronunciation: ['kualə...	12584

Wonder_city data can be downloaded from this link.

On this Raw data, We will apply a few basic pre-processing steps:

- Removal of Special Characters
- Removing Extra White Spaces
- Converting all text to lower case

We shall be creating a new column named clean_information and storing it back in our dataframe:

	wonder_city	information	num_tokens	cleaned_information	token_count
0	Beirut	Beirut is the capital and largest city of Le...	12695	beirut is the capital and largest city of leba...	12121
1	Doha	Doha (Arabic: الدوحة, romanized: ad-Dawḥa [ad'...	9016	doha (arabic romanized addawa [addua] or adda)...	8252
2	Durban	Durban (DUR-bən) (Zulu: eThekwini, from ithek...	7992	durban (durbn) (zulu ethekwini from itheku me...	7395
3	Havana	Havana (; Spanish: La Habana [la a'βana] ; Luc...	30000	havana (spanish la habana [la aana] lucumi il...	28979
4	Kuala Lumpur	Kuala Lumpur (Malaysian pronunciation: ['kualə...	12584	kuala lumpur (malaysian pronunciation [kual a ...	12397

Now that we have cleaned the information and determined the token count for each document, named "wonder_city," it becomes evident that we cannot input more than 4096 tokens into our Llama algorithm to generate a summary. However, before proceeding, we must first create a template for our text. First, we will define a template string. This template serves as a structured format for generating the summary and incorporates a placeholder, *{text}*, where the actual text content will be inserted.

Now we'll make a prompt template object, which will use the previously established template and expect an input variable called "text."

We shall make an LLMChain Object. This object is in charge of connecting the prompt template and the language model (LLM) for text generation. It basically creates the pipeline for creating the summary.

```
def generate_summary(text_chunk):
    # Defining the template to generate summary
    template = """
    Write a concise summary of the text, return your responses with 5 lines that c
    ```{text}```
    SUMMARY:
    """
    prompt = PromptTemplate(template=template, input_variables=["text"])
    llm_chain = LLMChain(prompt=prompt, llm=LLM)

    summary = llm_chain.run(text_chunk)
    return summary
```

As we are aware, the Llama2 model has a limitation of processing up to 4096 tokens. Therefore, it is essential to divide our documents (referred to as "wonder_city") into manageable chunks. There are several methods for chunking, and you can explore various techniques in my notebook dedicated to this topic. In our specific use case, we will employ Langchain's "***RecursiveCharacterTextSplitter"*** module. This module not only assists in chunking but also facilitates token overlap, enabling us to capture context for the subsequent chunking process.

In the code below, We are chunking text and using those chunks to generate summaries. Once we have generated summaries for all the chunks using the Llama2 model, we will consolidate them into a single summary by concatenating them with newline characters. These resulting "summaries" will then be stored in our DataFrame's "summary" column.
```

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=4096, chunk_overlap=50,

df["summary"] = ""

for index, row in tqdm(df.iterrows(), total=len(df), desc="Generating Summaries")
    wonder_city = row["wonder_city"]
    text_chunk = row["cleaned_information"]
    chunks = text_splitter.split_text(text_chunk)
    chunk_summaries = []

    for chunk in chunks:
        summary = generate_summary(chunk)
        chunk_summaries.append(summary)


    combined_summary = "\n".join(chunk_summaries)
    df.at[index, "summary"] = combined_summary
```

This code will take a few hours to run due to the large number of tokens being processed. Therefore, it's a good idea to grab a cup of coffee, sit back, relax, and enjoy some other tasks or music while it runs. :)

Once the summaries have been generated, you can calculate the number of tokens and view the results as shown below:

	wonder_city	information	num_tokens	cleaned_information	token_count	summary	summary_token_count
0	Beirut	Beirut is the capital and largest city of Le...	12695	beirut is the capital and largest city of leba...	12121	1. Beirut is the capital and largest city of L...	2806
1	Doha	Doha (Arabic: الدوحة, romanized: ad-Dawḥa [ad'...	9016	doha (arabic romanized addawa [addua] or adda)...	8252	1. Doha is the capital city and financial hub ...	1645
2	Durban	Durban (DUR-bən) (Zulu: eThekwini, from ithek...	7992	durban (durbn) (zulu ethekwini from itheku me...	7395	1. Durban is located in KwaZulu-Natal, South A...	2151
3	Havana	Havana (; Spanish: La Habana [la a 'βana] ; Luc...	30000	havana (spanish la habana [la aana] lucumi il...	28979	1) Havana is the capital and largest city of C...	6909
4	Kuala Lumpur	Kuala Lumpur (Malaysian pronunciation: ['kualə...	12584	kuala lumpur (malaysian pronunciation [kual a ...	12397	1. Kuala Lumpur is the capital city of Malaysi...	2721

Now you can check your summarized column as follows:

```
selected_columns = df[["wonder_city", "summary"]]

for index, row in selected_columns.iterrows():
    wonder_city = row["wonder_city"]
    summary = row["summary"]

    formatted_wonder_city = colored(wonder_city, "green", attrs=["bold", "underli

    formatted_summary = colored(f"Summary: {summary}", "black")

    print(formatted_wonder_city)

    print()

    print(formatted_summary)

    print("\n---------------------------------------------\n")
```

I have also calculated ROUGE scores, primarily for the purpose of evaluating the quality of my summaries. That concludes my explanation.

You can visit my GitHub notebook link to gain a deeper understanding of the code. I hope this article will assist you in developing your own text summarization solution for multiple documents.

Summary

According to research and practical implementation, LLM (Large Language Models) still have a considerable journey ahead, demanding substantial computational resources to be available locally on your system. To effectively process extensive volumes of text data, the presence of a GPU is essential.

— — — — — — — — — — — — — — — -|| — — — — — — — — — — — — — — —

JUST !! Do not add stories to your list; please upvote the stories and reach out to me for questions and follow-ups. I will be happy to help.

Next.. I am working on big stuff to present. Wait for some time I will come back with a **BOOM !! :)**.

Till than ***Buy me a coffee***. **Bon Voyage !!**

Feel free to reach out to me on linkedin: https://www.linkedin.com/in/tushitdave/

Llama 2 | Artificial Intelligence | Text Summarization | Llm | Large Language Models

Follow

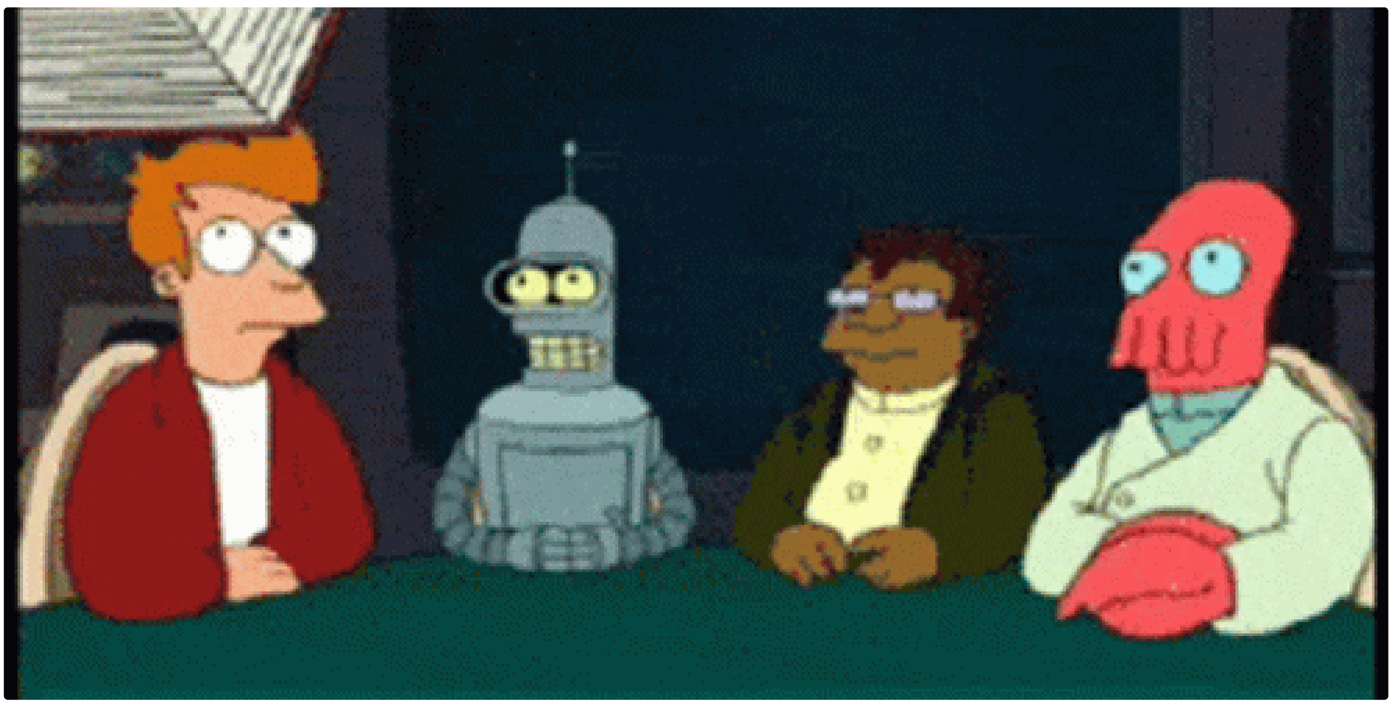
Written by Tushit Dave

15 Followers

A Tale Weaver and Visionary Data Scientist , Linkedin: https://www.linkedin.com/in/tushitdave/ ,Buy me a coffee : https://ko-fi.com/tushitdave

---

More from Tushit Dave



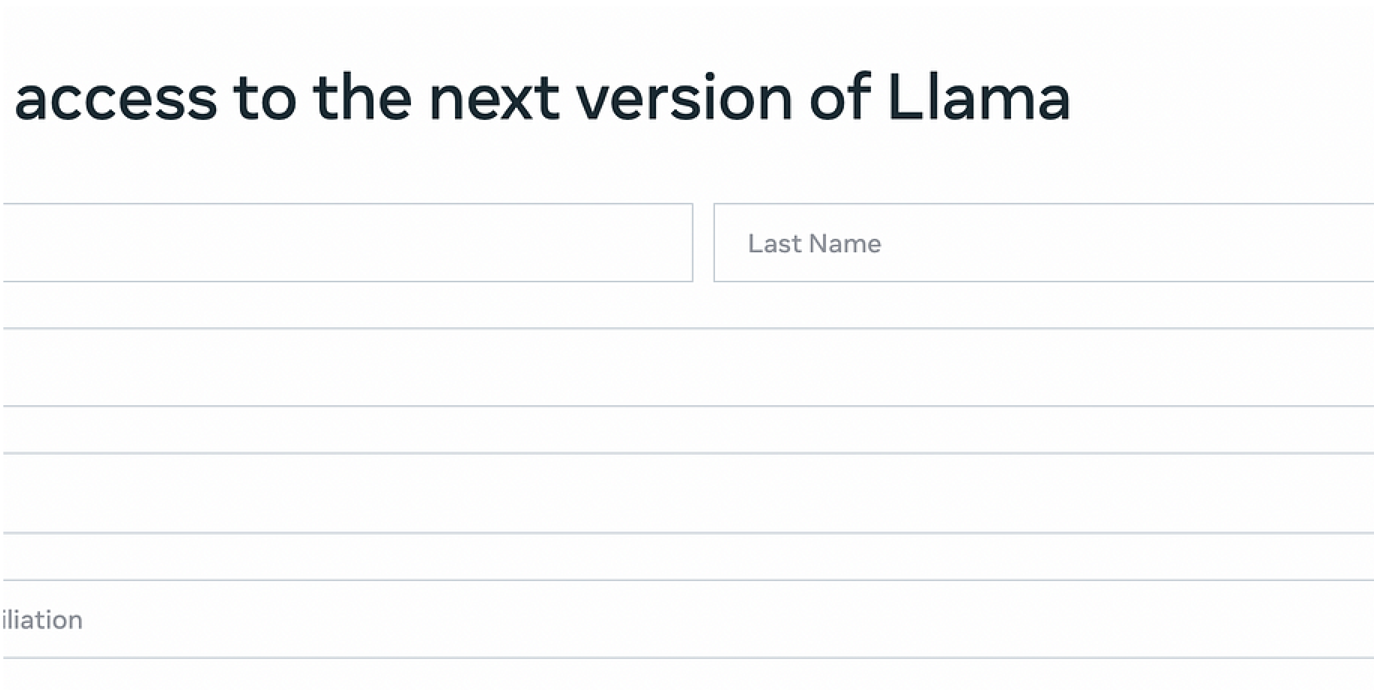
 Tushit Dave

Multiple Document Summary and LLM Powered QA-System

In this blog post, we will discuss how we can summarize multiple documents and develop a summary using Llama-Index and also develop a QA...

3 min read · Aug 28

81 1



Tushit Dave

How to Install Llama 2 Locally

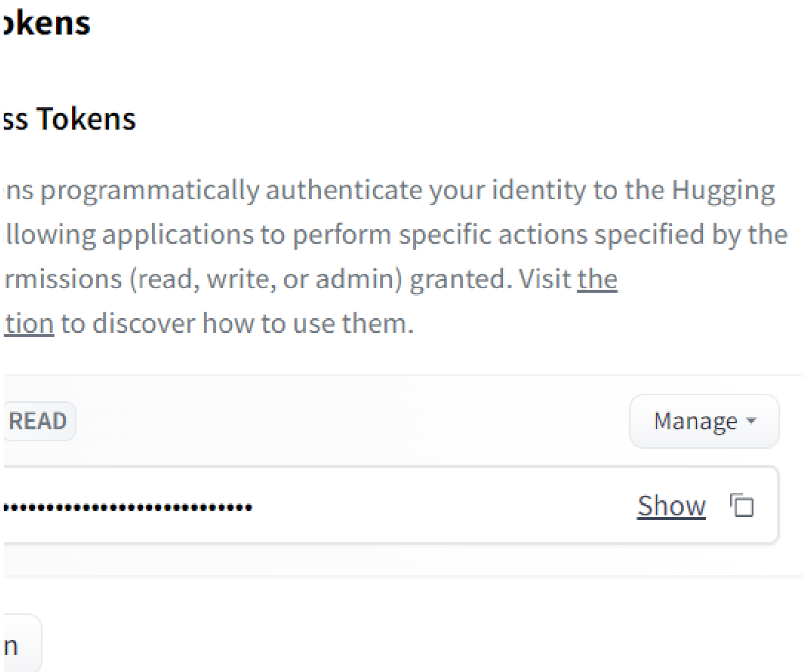
After the major release from Meta, you might be wondering how to download models such as 7B, 13B, 7B-chat, and 13B-chat locally in order to…

4 min read · Aug 31

2

See all from Tushit Dave

Recommended from Medium



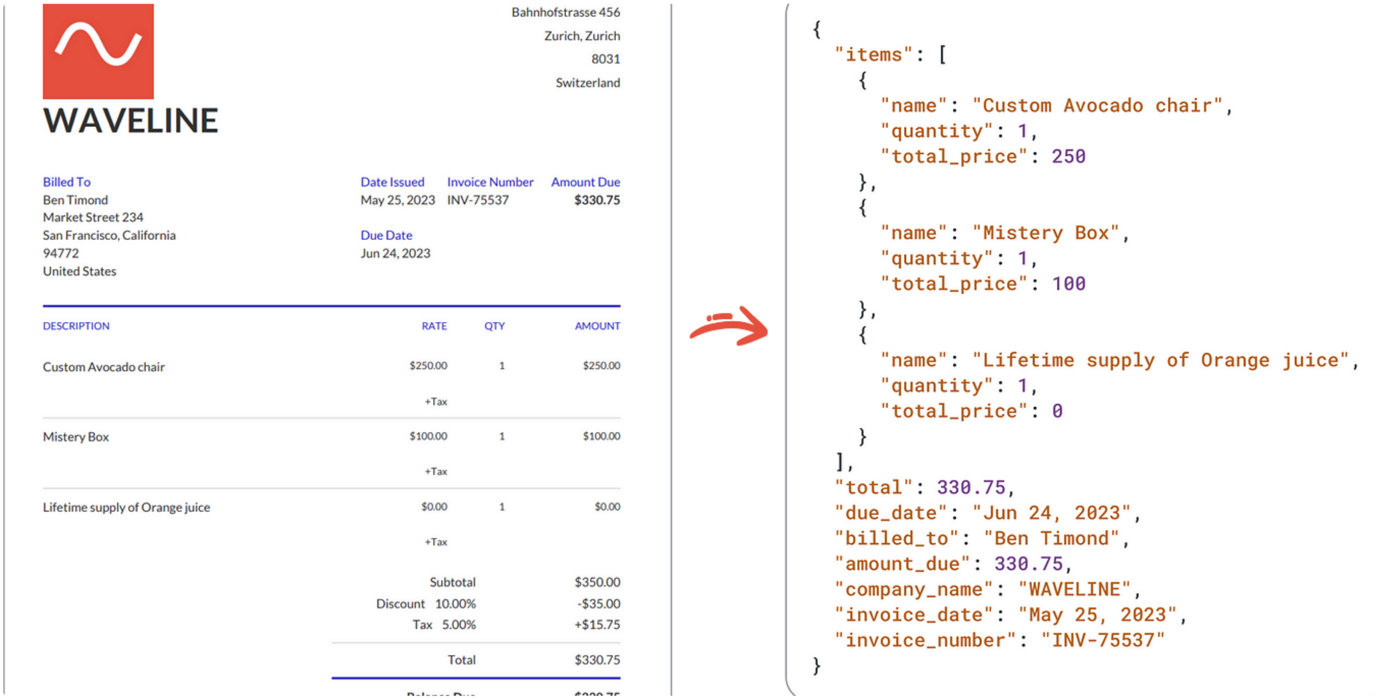
Ankit

Generating Summaries for Large Documents with Llama2 using Hugging Face and Langchain

Introduction

11 min read · Aug 28

99 3



WAVELINE

Bahnhofstrasse 456
Zurich, Zurich
8031
Switzerland

Billed To
Ben Timond
Market Street 234
San Francisco, California
94772
United States

Date Issued: May 25, 2023
Invoice Number: INV-75537
Amount Due: $330.75
Due Date: Jun 24, 2023

DESCRIPTION	RATE	QTY	AMOUNT
Custom Avocado chair	$250.00 +Tax	1	$250.00
Mistery Box	$100.00 +Tax	1	$100.00
Lifetime supply of Orange juice	$0.00 +Tax	1	$0.00

Subtotal		$350.00
Discount	10.00%	-$35.00
Tax	5.00%	+$15.75
Total		$330.75


```
{
  "items": [
    {
      "name": "Custom Avocado chair",
      "quantity": 1,
      "total_price": 250
    },
    {
      "name": "Mistery Box",
      "quantity": 1,
      "total_price": 100
    },
    {
      "name": "Lifetime supply of Orange juice",
      "quantity": 1,
      "total_price": 0
    }
  ],
  "total": 330.75,
  "due_date": "Jun 24, 2023",
  "billed_to": "Ben Timond",
  "amount_due": 330.75,
  "company_name": "WAVELINE",
  "invoice_date": "May 25, 2023",
  "invoice_number": "INV-75537"
}
```

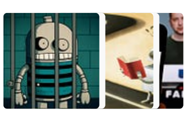

Waveline

Extract Data from Documents with ChatGPT

Guide on how to extract data from documents like PDFs using Large Language Models (LLMs)

4 min read · Jul 19

Lists

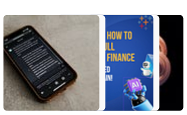


AI Regulation

6 stories · 137 saves

Natural Language Processing

657 stories · 265 saves

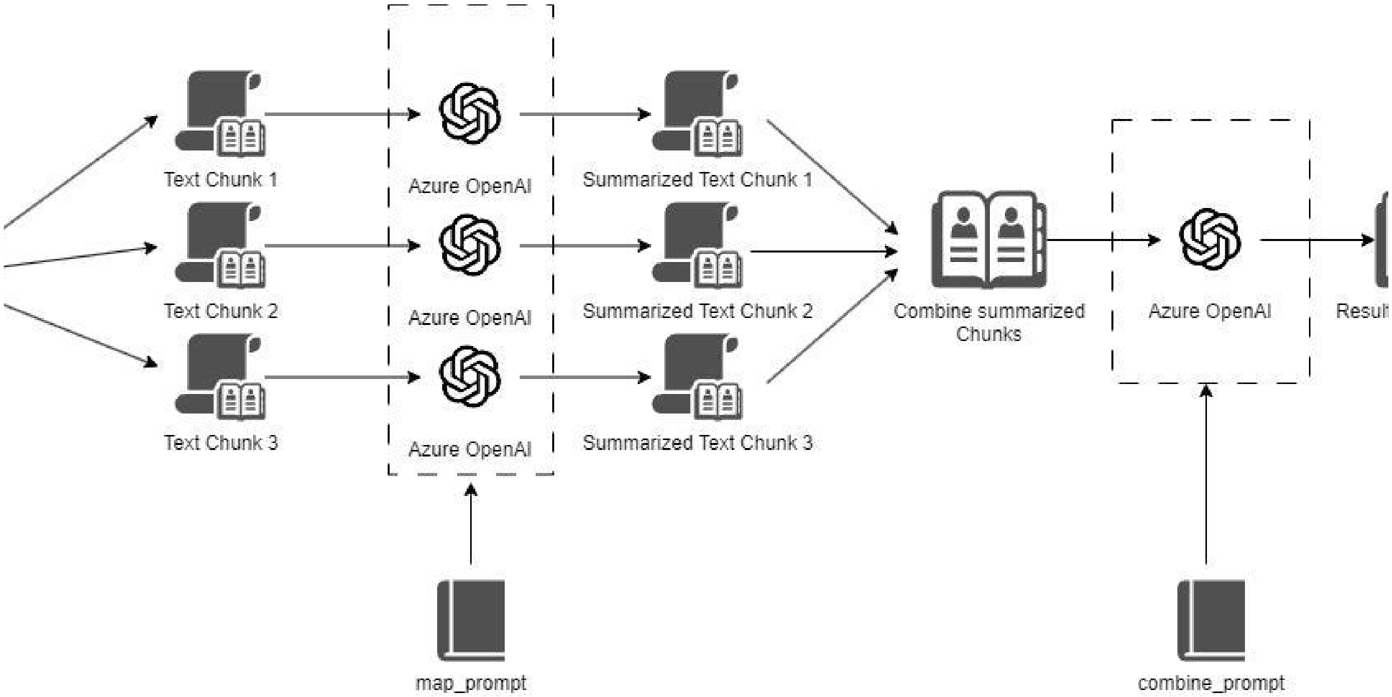


ChatGPT prompts

24 stories · 439 saves

ChatGPT

21 stories · 174 saves



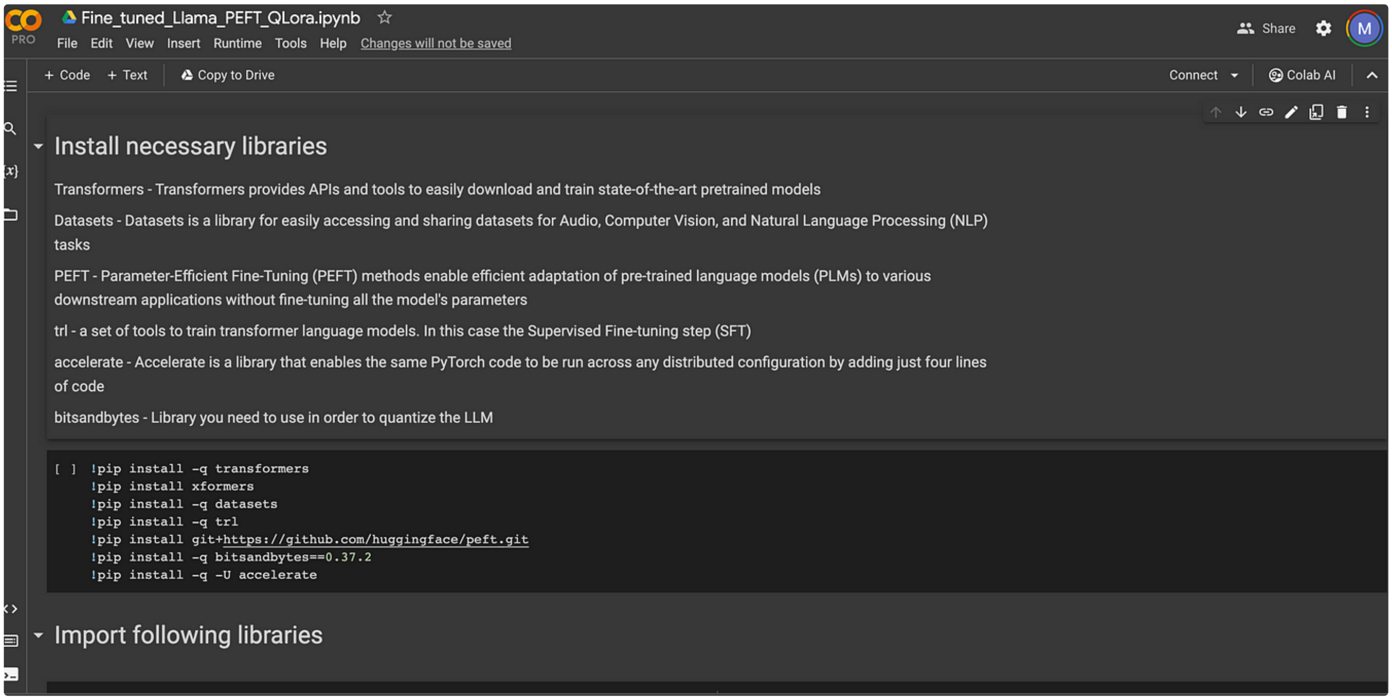
Bluetick Consultants

Advanced Techniques in Text Summarization: Leveraging Generative AI and Prompt Engineering

Generative AI and Language Models like OpenAI's GPT-3.5 have emerged as powerful tools for text summarization. In this blog, we will explore

9 min read · Jun 3

21 1



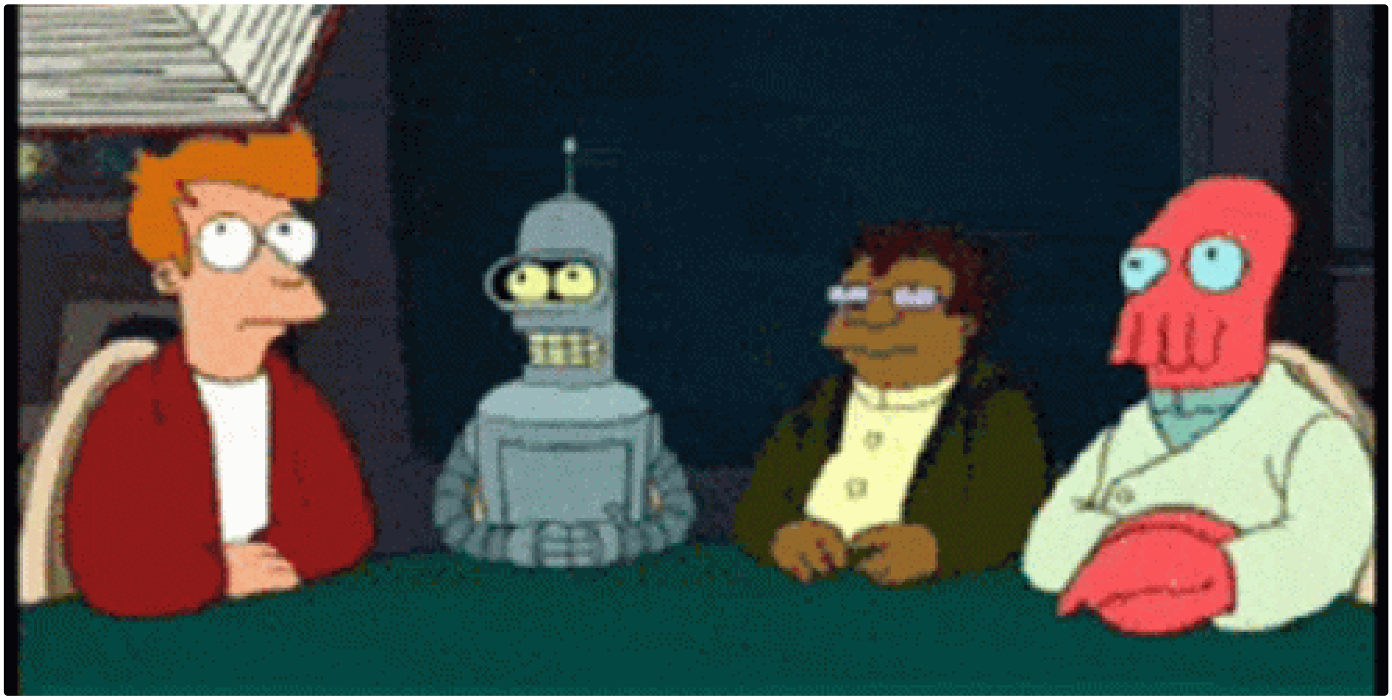
Maya Akim

Complete Guide to LLM Fine Tuning for Beginners

Fine-tuning a model refers to the process of adapting a pre-trained, foundational model (such as Falcom or Llama) to perform a new task or...

5 min read · Aug 14

94 1

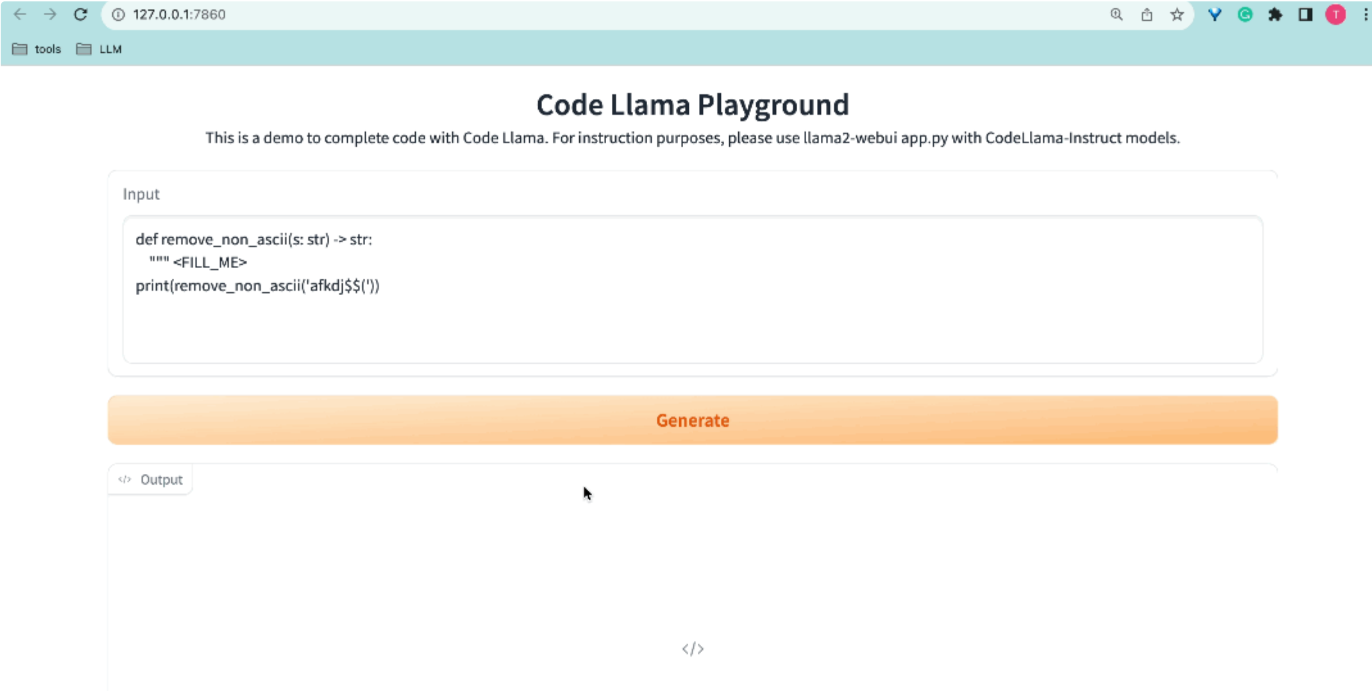


Tushit Dave

Multiple Document Summary and LLM Powered QA-System

In this blog post, we will discuss how we can summarize multiple documents and develop a summary using Llama-Index and also develop a QA…

3 min read · Aug 28



Tom Gou

Run Code Llama locally on Your Macbook

llama2-wrapper is the package wrapping multiple llama2 backends to run chatbot and code playground for Code Llama.

3 min read · Sep 1

38 2

See more recommendations