Pattern Recognition

Project Report:

RAG with LLM dataset medical books in pdf format

Group members:

FA21-BCS-066 HUSSAIN ALI

FA21-BCS-080 ALI SHAN

Invigilator:

Dr.Samia Riaz

Project Scope:

Objective:

The objective of this project is to build a **Retriever-Augmented Generation (RAG)** system that can answer medical questions based on information extracted from medical books in PDF format. The system will integrate a retrieval-based model to find the most relevant pieces of information from a large corpus of medical data and a generative language model to synthesize responses based on these retrieved documents. This system will be designed to help medical students, professionals, and researchers quickly access accurate information from a vast range of medical books and references.

Input:

The input to the system will consist of medical books in PDF format, which may cover a variety of topics such as anatomy, pharmacology, diseases, treatment methods, and diagnostic procedures. These PDFs will first be processed into text, which will then be split into smaller chunks to facilitate easier retrieval and better performance when searching for relevant information. These chunks will be stored in a vector database, where they can be efficiently searched using embeddings to retrieve the most relevant information based on a user query. The system will return answers that are both accurate and detailed, relying on the generative capabilities of a fine-tuned language model.

Goal:

The goal is to create a robust system capable of answering a wide array of medical queries with high precision and reliability, using content directly extracted from the medical books.

Dataset Link:

https://drive.google.com/drive/folders/1twxYrqAxzDcimg1iFKmsfByA3A9B-6IT?usp=drive_link

Dataset Information:

Dataset Structure:

The **medical books** in the dataset are in **PDF format**, and they will first need to be processed to extract text. Once the text is extracted, it will be split into smaller **chunks** to facilitate easy retrieval. These chunks will be converted into **vector embeddings** using a model like **sentence-transformers** or **bioBERT** to ensure that semantic meaning is captured.

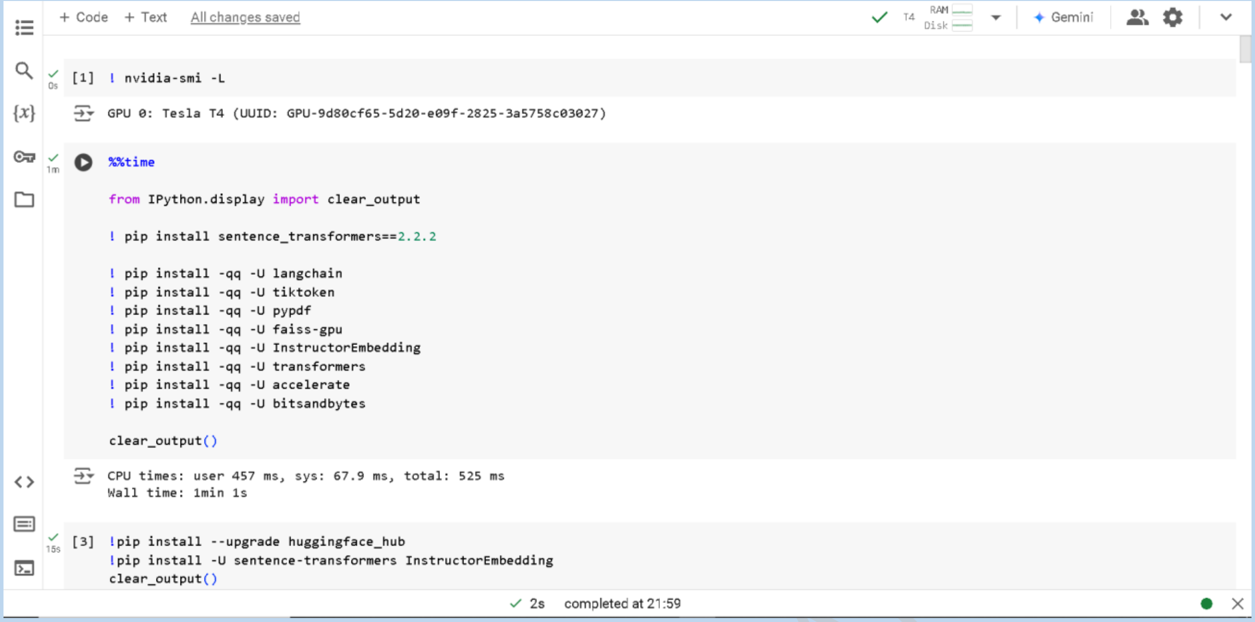
The **vector database** will store these embeddings, and when a user asks a medical question, the system will retrieve the most relevant chunks based on the query's **semantic similarity** and generate a comprehensive answer using the **generative model** (like **Llama-2**).

This dataset is essential for training the retriever and fine-tuning the generative language model so it can provide accurate, context-aware medical information.

Conclusion:

This project illustrates the potential of **RAG systems** in the healthcare domain, providing a powerful tool for information retrieval and natural language generation. By using advanced machine learning techniques, it offers an efficient and scalable solution for medical question answering, which can significantly benefit medical students, healthcare professionals, and patients seeking reliable information.

Code Screenshots:



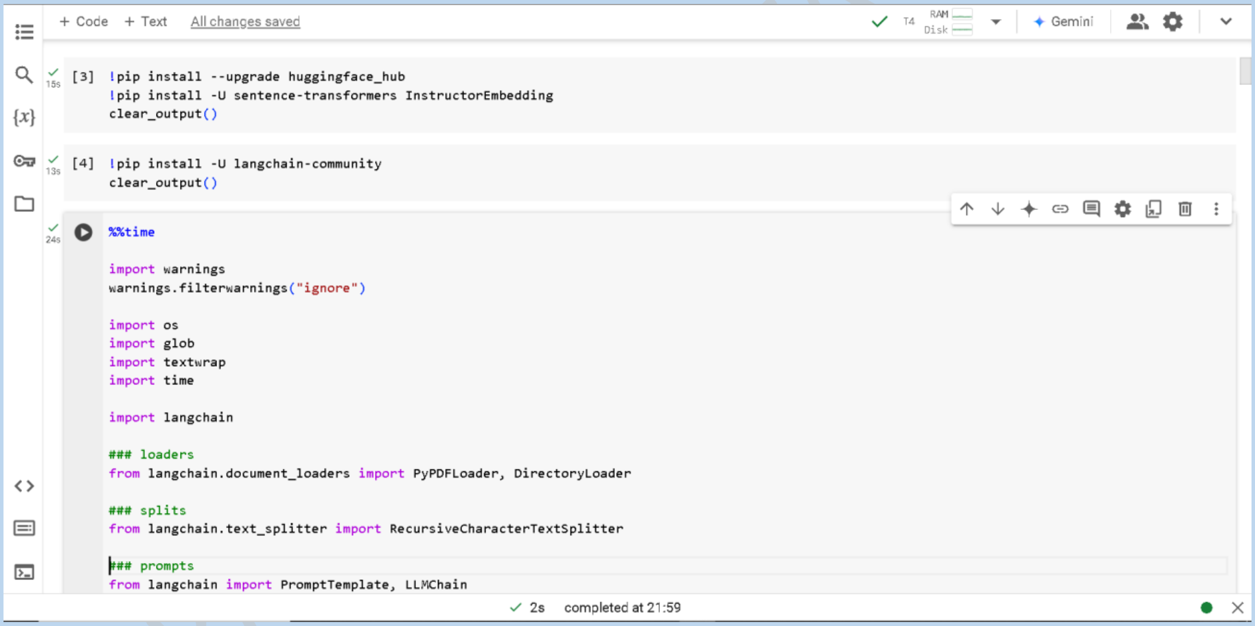
```
[1] ! nvidia-smi -L
```

```
GPU 0: Tesla T4 (UUID: GPU-9d80cf65-5d20-e09f-2825-3a5758c03027)
```

```
%%time

from IPython.display import clear_output

! pip install sentence_transformers==2.2.2

! pip install -qq -U langchain
! pip install -qq -U tiktoken
! pip install -qq -U pypdf
! pip install -qq -U faiss-gpu
! pip install -qq -U InstructorEmbedding
! pip install -qq -U transformers
! pip install -qq -U accelerate
! pip install -qq -U bitsandbytes

clear_output()
```

```
CPU times: user 457 ms, sys: 67.9 ms, total: 525 ms
Wall time: 1min 1s
```

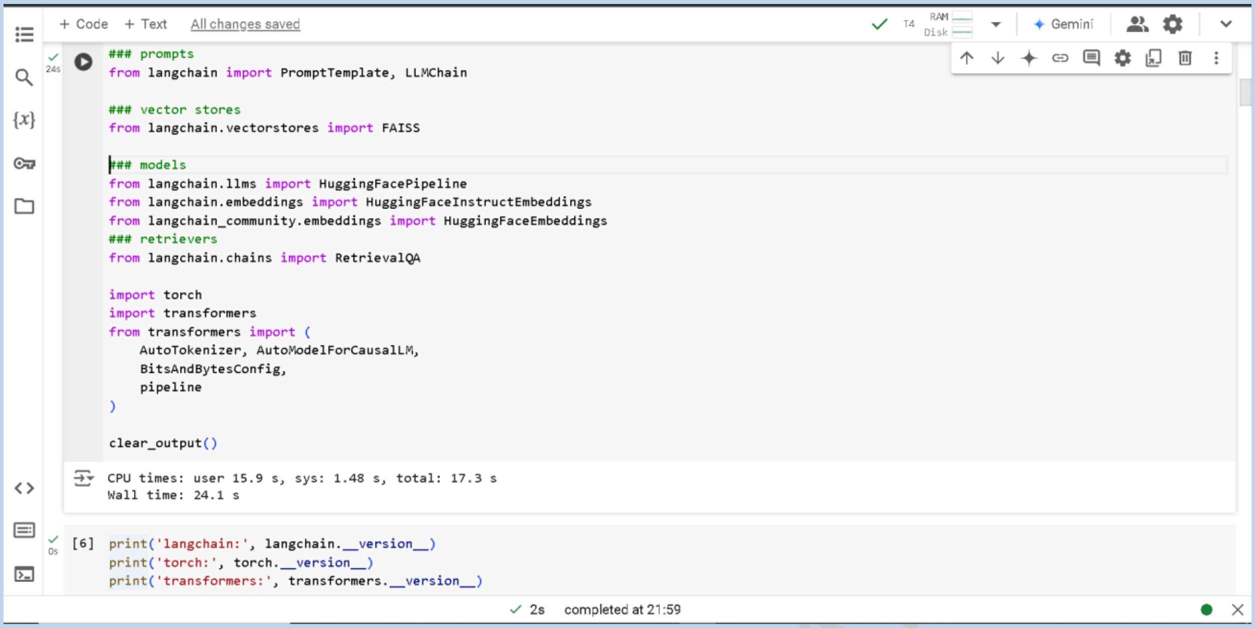
```
[3] !pip install --upgrade huggingface_hub
    !pip install -U sentence-transformers InstructorEmbedding
    clear_output()
```



```
[3] !pip install --upgrade huggingface_hub
    !pip install -U sentence-transformers InstructorEmbedding
    clear_output()
```

```
[4] !pip install -U langchain-community
    clear_output()
```

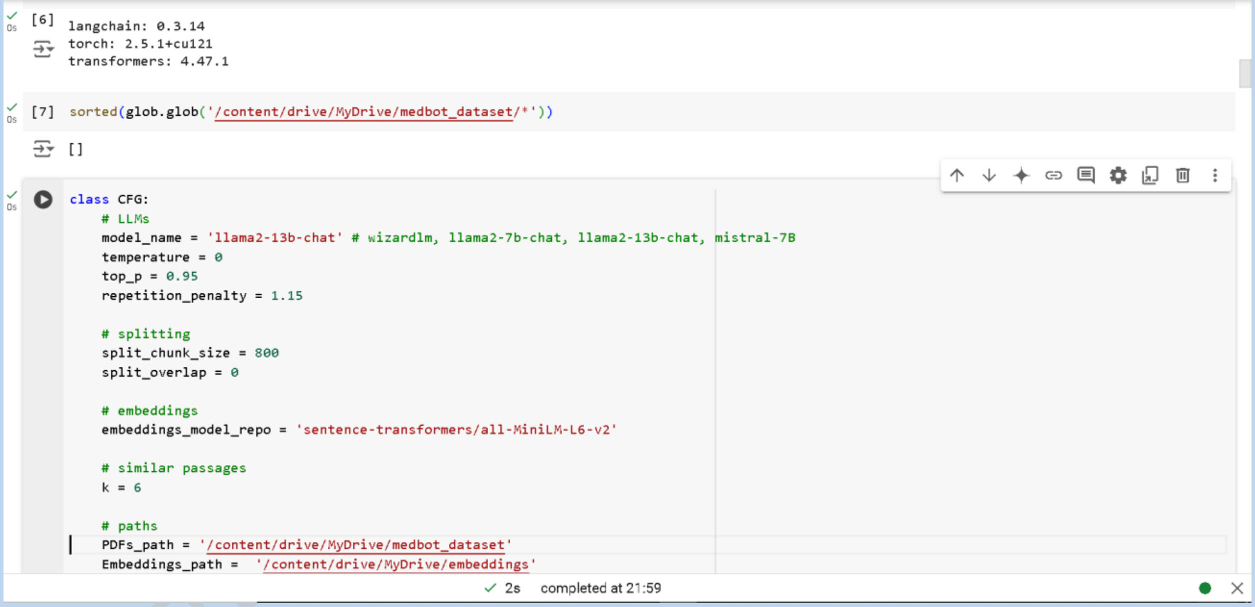
```
%%time

import warnings
warnings.filterwarnings("ignore")

import os
import glob
import textwrap
import time

import langchain

### loaders
from langchain.document_loaders import PyPDFLoader, DirectoryLoader

### splits
from langchain.text_splitter import RecursiveCharacterTextSplitter

### prompts
from langchain import PromptTemplate, LLMChain
```



```python
### prompts
from langchain import PromptTemplate, LLMChain

### vector stores
from langchain.vectorstores import FAISS

### models
from langchain.llms import HuggingFacePipeline
from langchain.embeddings import HuggingFaceInstructEmbeddings
from langchain_community.embeddings import HuggingFaceEmbeddings
### retrievers
from langchain.chains import RetrievalQA

import torch
import transformers
from transformers import (
    AutoTokenizer, AutoModelForCausalLM,
    BitsAndBytesConfig,
    pipeline
)

clear_output()
```

```
CPU times: user 15.9 s, sys: 1.48 s, total: 17.3 s
Wall time: 24.1 s
```

```python
[6] print('langchain:', langchain.__version__)
    print('torch:', torch.__version__)
    print('transformers:', transformers.__version__)
```

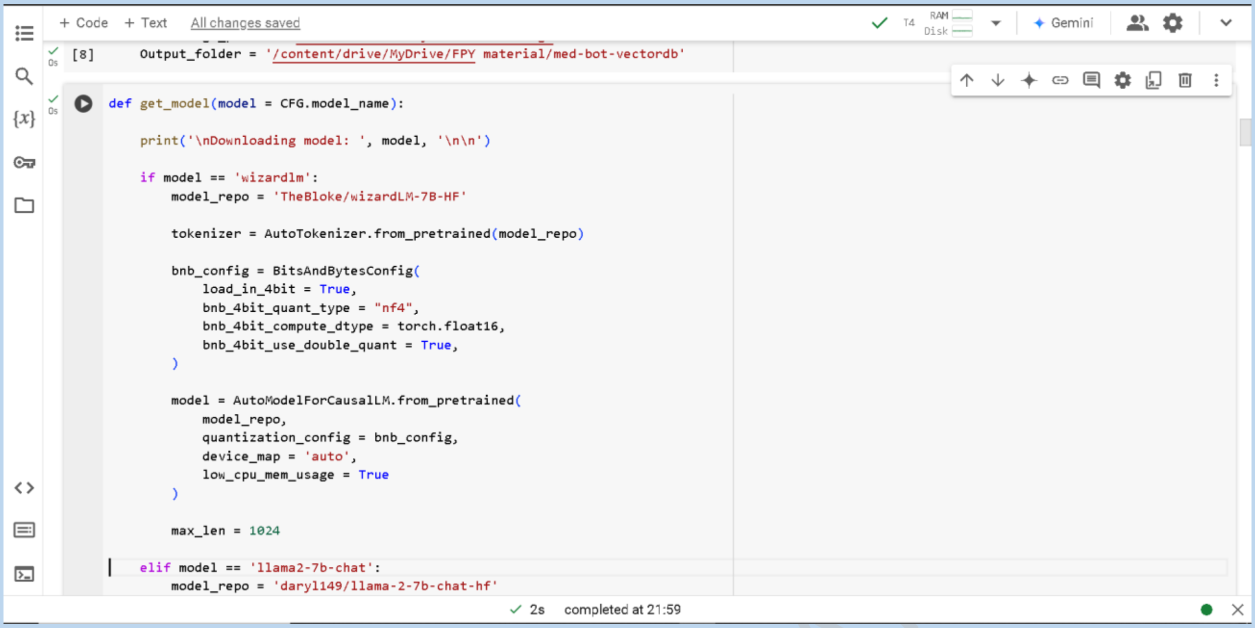


```
[6] langchain: 0.3.14
    torch: 2.5.1+cu121
    transformers: 4.47.1
```

```python
[7] sorted(glob.glob('/content/drive/MyDrive/medbot_dataset/*'))
```

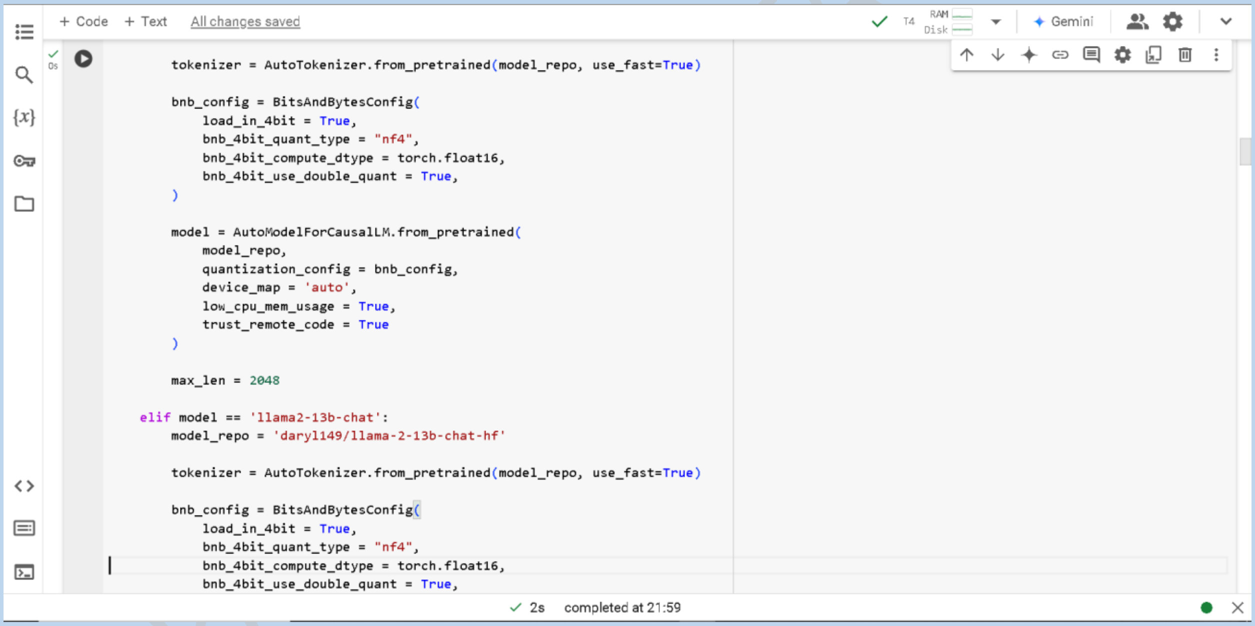
```
[]
```

```python
class CFG:
    # LLMs
    model_name = 'llama2-13b-chat' # wizardlm, llama2-7b-chat, llama2-13b-chat, mistral-7B
    temperature = 0
    top_p = 0.95
    repetition_penalty = 1.15

    # splitting
    split_chunk_size = 800
    split_overlap = 0

    # embeddings
    embeddings_model_repo = 'sentence-transformers/all-MiniLM-L6-v2'

    # similar passages
    k = 6

    # paths
    PDFs_path = '/content/drive/MyDrive/medbot_dataset'
    Embeddings_path =  '/content/drive/MyDrive/embeddings'
```

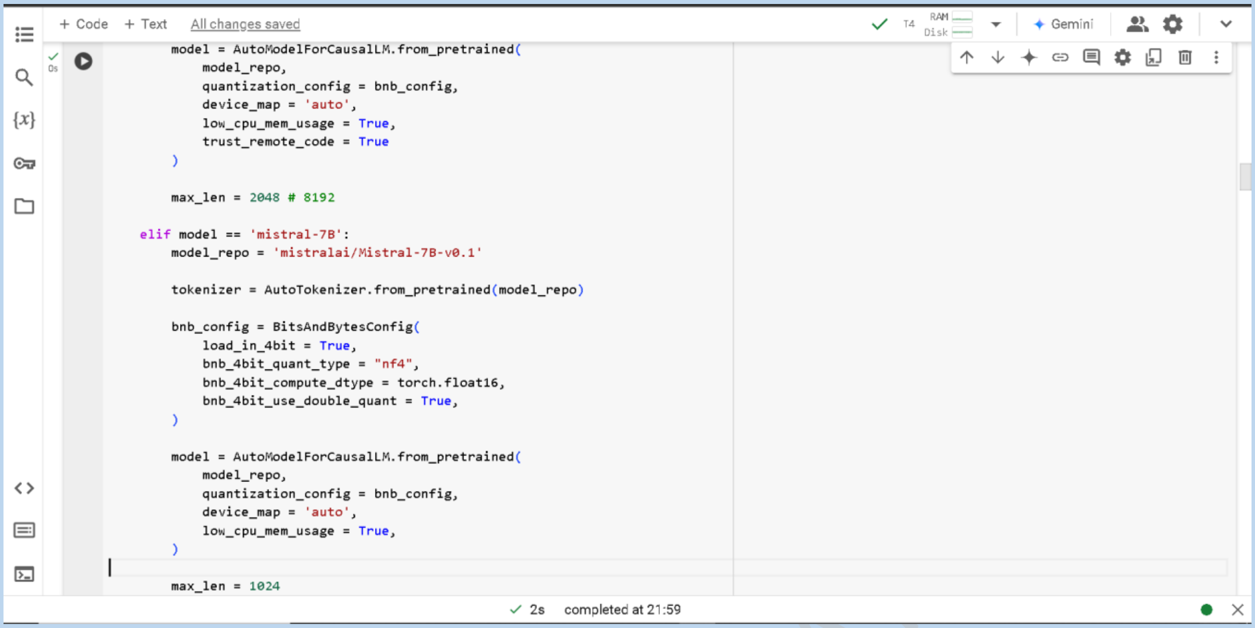


```
[8]    Output_folder = '/content/drive/MyDrive/FPY material/med-bot-vectordb'
```

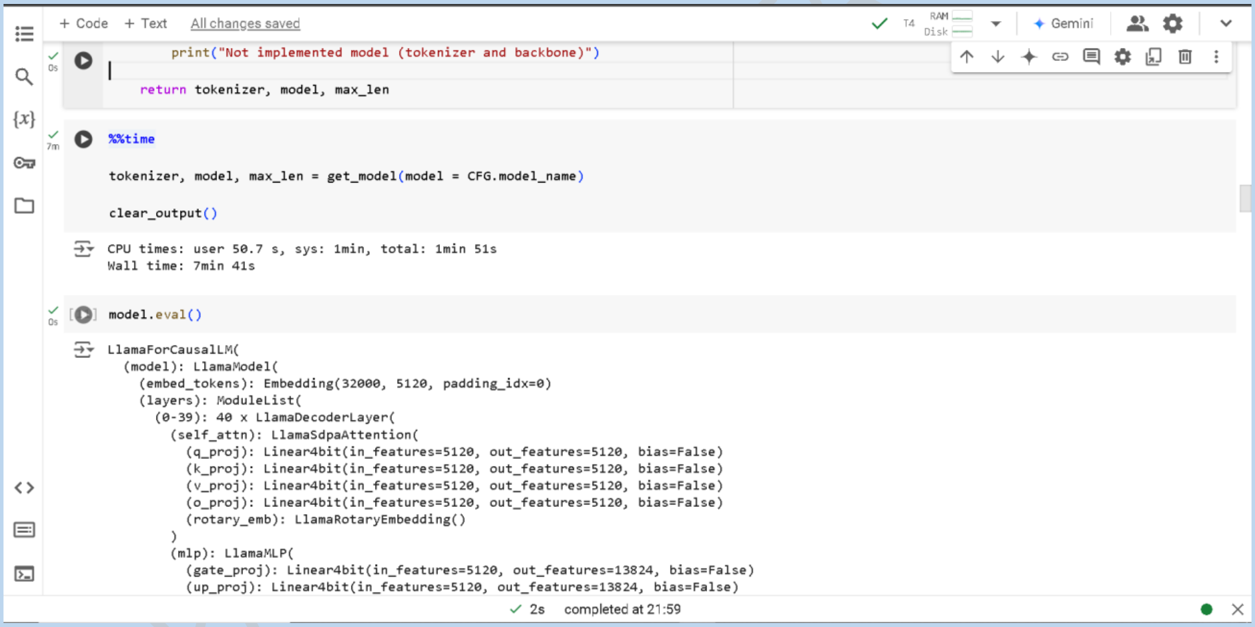
```
def get_model(model = CFG.model_name):

    print('\nDownloading model: ', model, '\n\n')

    if model == 'wizardlm':
        model_repo = 'TheBloke/wizardLM-7B-HF'

        tokenizer = AutoTokenizer.from_pretrained(model_repo)

        bnb_config = BitsAndBytesConfig(
            load_in_4bit = True,
            bnb_4bit_quant_type = "nf4",
            bnb_4bit_compute_dtype = torch.float16,
            bnb_4bit_use_double_quant = True,
        )

        model = AutoModelForCausalLM.from_pretrained(
            model_repo,
            quantization_config = bnb_config,
            device_map = 'auto',
            low_cpu_mem_usage = True
        )

        max_len = 1024

    elif model == 'llama2-7b-chat':
        model_repo = 'daryl149/llama-2-7b-chat-hf'
```



```
        tokenizer = AutoTokenizer.from_pretrained(model_repo, use_fast=True)

        bnb_config = BitsAndBytesConfig(
            load_in_4bit = True,
            bnb_4bit_quant_type = "nf4",
            bnb_4bit_compute_dtype = torch.float16,
            bnb_4bit_use_double_quant = True,
        )

        model = AutoModelForCausalLM.from_pretrained(
            model_repo,
            quantization_config = bnb_config,
            device_map = 'auto',
            low_cpu_mem_usage = True,
            trust_remote_code = True
        )

        max_len = 2048

    elif model == 'llama2-13b-chat':
        model_repo = 'daryl149/llama-2-13b-chat-hf'

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

        bnb_config = BitsAndBytesConfig(
            load_in_4bit = True,
            bnb_4bit_quant_type = "nf4",
            bnb_4bit_compute_dtype = torch.float16,
            bnb_4bit_use_double_quant = True,
```



```
        model = AutoModelForCausalLM.from_pretrained(
            model_repo,
            quantization_config = bnb_config,
            device_map = 'auto',
            low_cpu_mem_usage = True,
            trust_remote_code = True
        )

        max_len = 2048 # 8192

    elif model == 'mistral-7B':
        model_repo = 'mistralai/Mistral-7B-v0.1'

        tokenizer = AutoTokenizer.from_pretrained(model_repo)

        bnb_config = BitsAndBytesConfig(
            load_in_4bit = True,
            bnb_4bit_quant_type = "nf4",
            bnb_4bit_compute_dtype = torch.float16,
            bnb_4bit_use_double_quant = True,
        )

        model = AutoModelForCausalLM.from_pretrained(
            model_repo,
            quantization_config = bnb_config,
            device_map = 'auto',
            low_cpu_mem_usage = True,
        )

        max_len = 1024
```



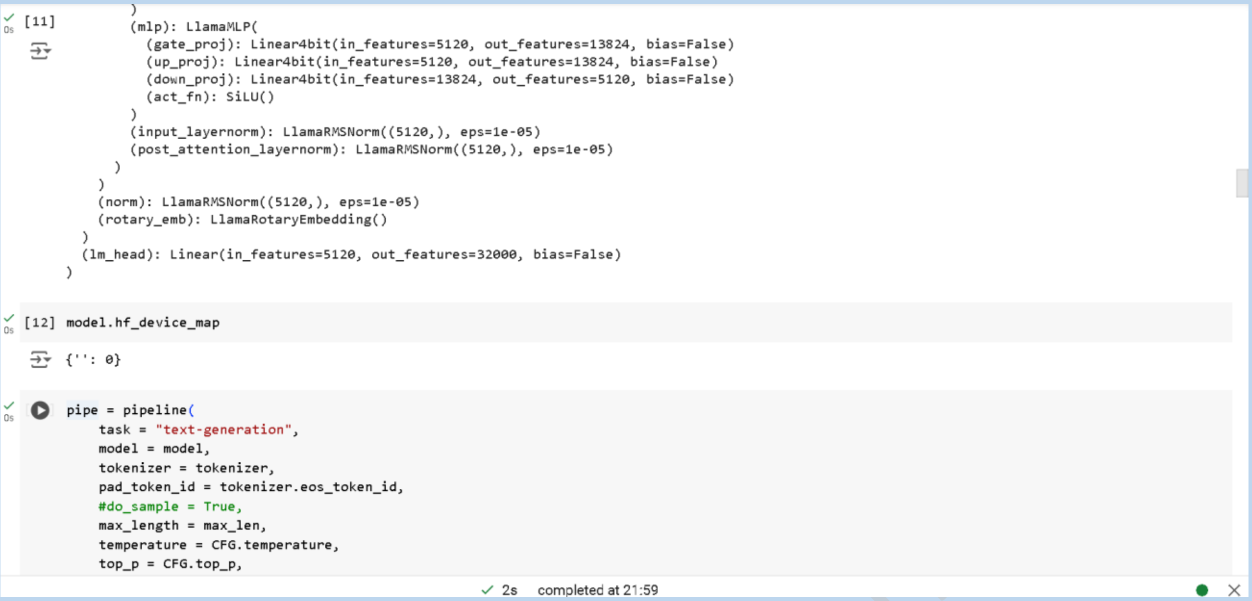
```
        print("Not implemented model (tokenizer and backbone)")

    return tokenizer, model, max_len
```

```
%%time

tokenizer, model, max_len = get_model(model = CFG.model_name)

clear_output()
```

```
CPU times: user 50.7 s, sys: 1min, total: 1min 51s
Wall time: 7min 41s
```

```
model.eval()
```

```
LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(32000, 5120, padding_idx=0)
    (layers): ModuleList(
      (0-39): 40 x LlamaDecoderLayer(
        (self_attn): LlamaSdpaAttention(
          (q_proj): Linear4bit(in_features=5120, out_features=5120, bias=False)
          (k_proj): Linear4bit(in_features=5120, out_features=5120, bias=False)
          (v_proj): Linear4bit(in_features=5120, out_features=5120, bias=False)
          (o_proj): Linear4bit(in_features=5120, out_features=5120, bias=False)
          (rotary_emb): LlamaRotaryEmbedding()
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear4bit(in_features=5120, out_features=13824, bias=False)
          (up_proj): Linear4bit(in_features=5120, out_features=13824, bias=False)
```

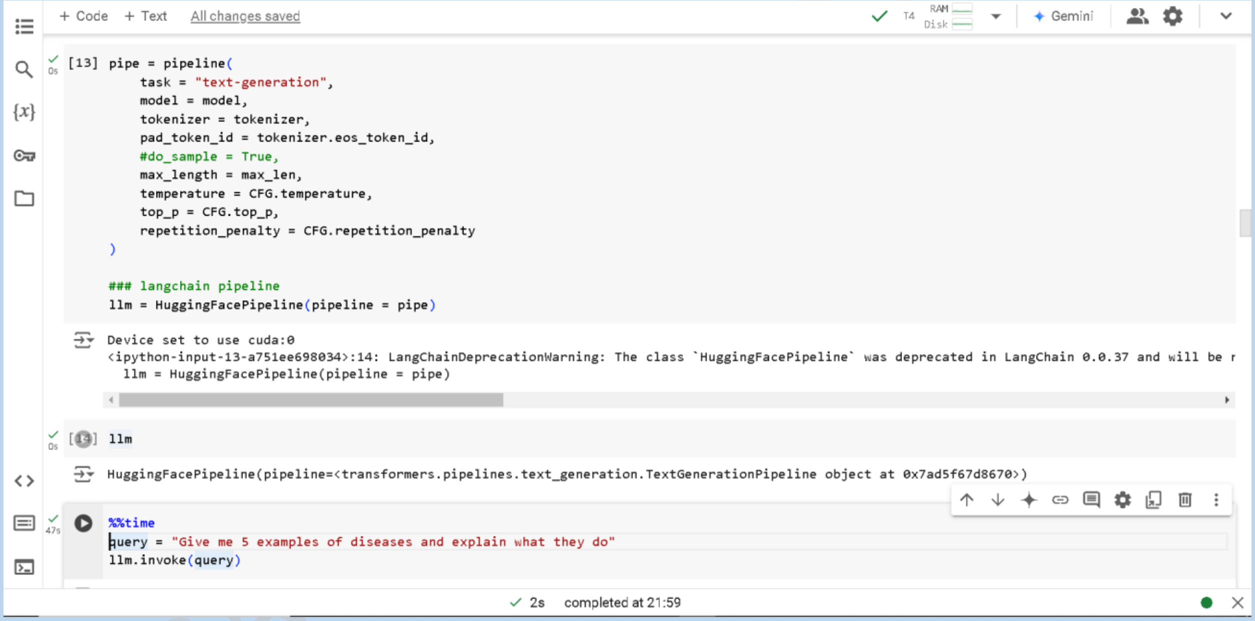


```
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear4bit(in_features=5120, out_features=13824, bias=False)
          (up_proj): Linear4bit(in_features=5120, out_features=13824, bias=False)
          (down_proj): Linear4bit(in_features=13824, out_features=5120, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): LlamaRMSNorm((5120,), eps=1e-05)
        (post_attention_layernorm): LlamaRMSNorm((5120,), eps=1e-05)
      )
    )
    (norm): LlamaRMSNorm((5120,), eps=1e-05)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (lm_head): Linear(in_features=5120, out_features=32000, bias=False)
)
```

```
[12] model.hf_device_map
```

```
{'': 0}
```

```
pipe = pipeline(
    task = "text-generation",
    model = model,
    tokenizer = tokenizer,
    pad_token_id = tokenizer.eos_token_id,
    #do_sample = True,
    max_length = max_len,
    temperature = CFG.temperature,
    top_p = CFG.top_p,
```



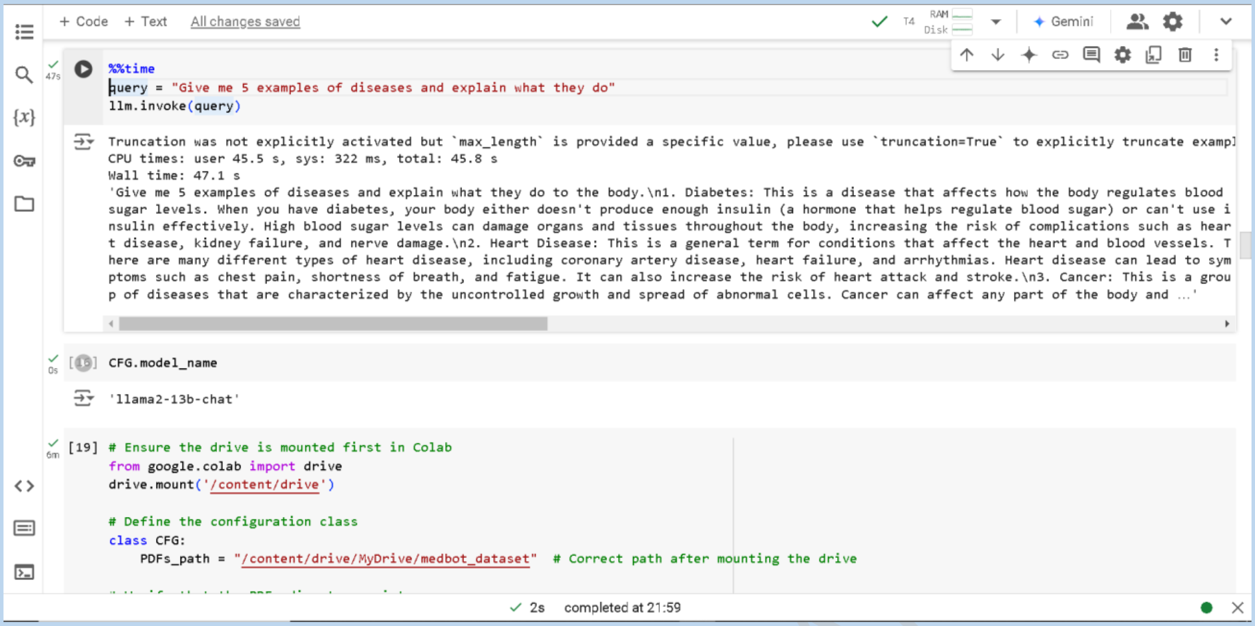
```
[13] pipe = pipeline(
         task = "text-generation",
         model = model,
         tokenizer = tokenizer,
         pad_token_id = tokenizer.eos_token_id,
         #do_sample = True,
         max_length = max_len,
         temperature = CFG.temperature,
         top_p = CFG.top_p,
         repetition_penalty = CFG.repetition_penalty
     )

     ### langchain pipeline
     llm = HuggingFacePipeline(pipeline = pipe)
```

```
Device set to use cuda:0
<ipython-input-13-a751ee698034>:14: LangChainDeprecationWarning: The class `HuggingFacePipeline` was deprecated in LangChain 0.0.37 and will be r
  llm = HuggingFacePipeline(pipeline = pipe)
```

```
[14] llm
```

```
HuggingFacePipeline(pipeline=<transformers.pipelines.text_generation.TextGenerationPipeline object at 0x7ad5f67d8670>)
```

```
%%time
query = "Give me 5 examples of diseases and explain what they do"
llm.invoke(query)
```



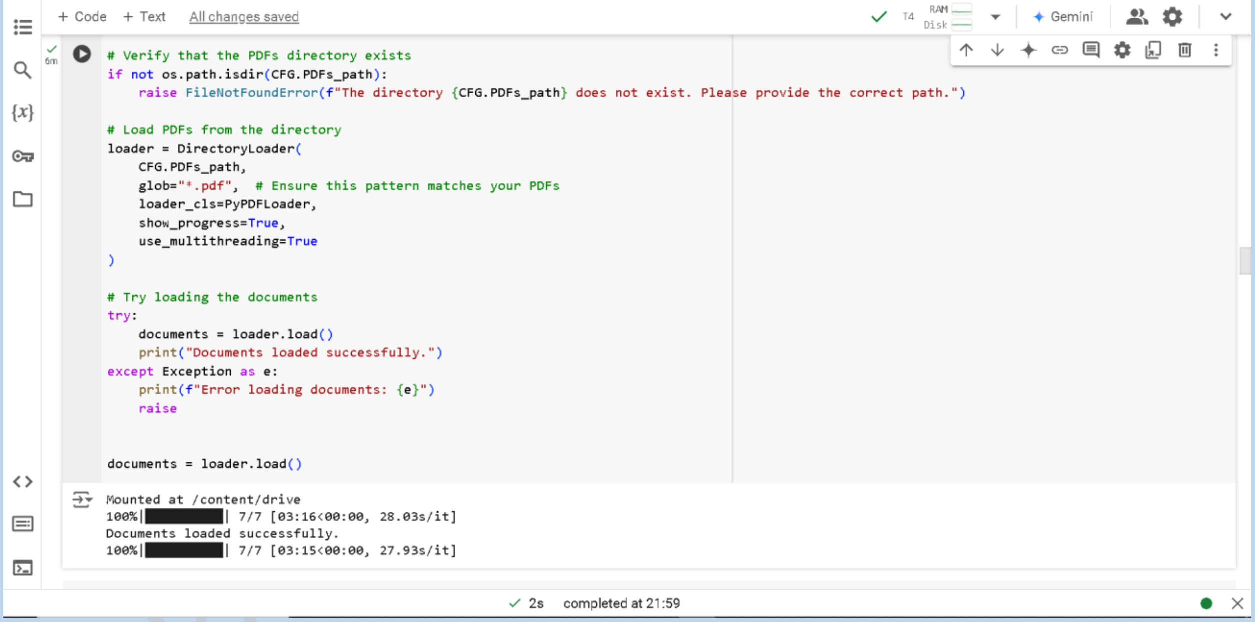
```
%%time
query = "Give me 5 examples of diseases and explain what they do"
llm.invoke(query)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate exampl
CPU times: user 45.5 s, sys: 322 ms, total: 45.8 s
Wall time: 47.1 s
'Give me 5 examples of diseases and explain what they do to the body.\n1. Diabetes: This is a disease that affects how the body regulates blood
sugar levels. When you have diabetes, your body either doesn't produce enough insulin (a hormone that helps regulate blood sugar) or can't use i
nsulin effectively. High blood sugar levels can damage organs and tissues throughout the body, increasing the risk of complications such as hear
t disease, kidney failure, and nerve damage.\n2. Heart Disease: This is a general term for conditions that affect the heart and blood vessels. T
here are many different types of heart disease, including coronary artery disease, heart failure, and arrhythmias. Heart disease can lead to sym
ptoms such as chest pain, shortness of breath, and fatigue. It can also increase the risk of heart attack and stroke.\n3. Cancer: This is a grou
p of diseases that are characterized by the uncontrolled growth and spread of abnormal cells. Cancer can affect any part of the body and ...'
```

```
CFG.model_name
```

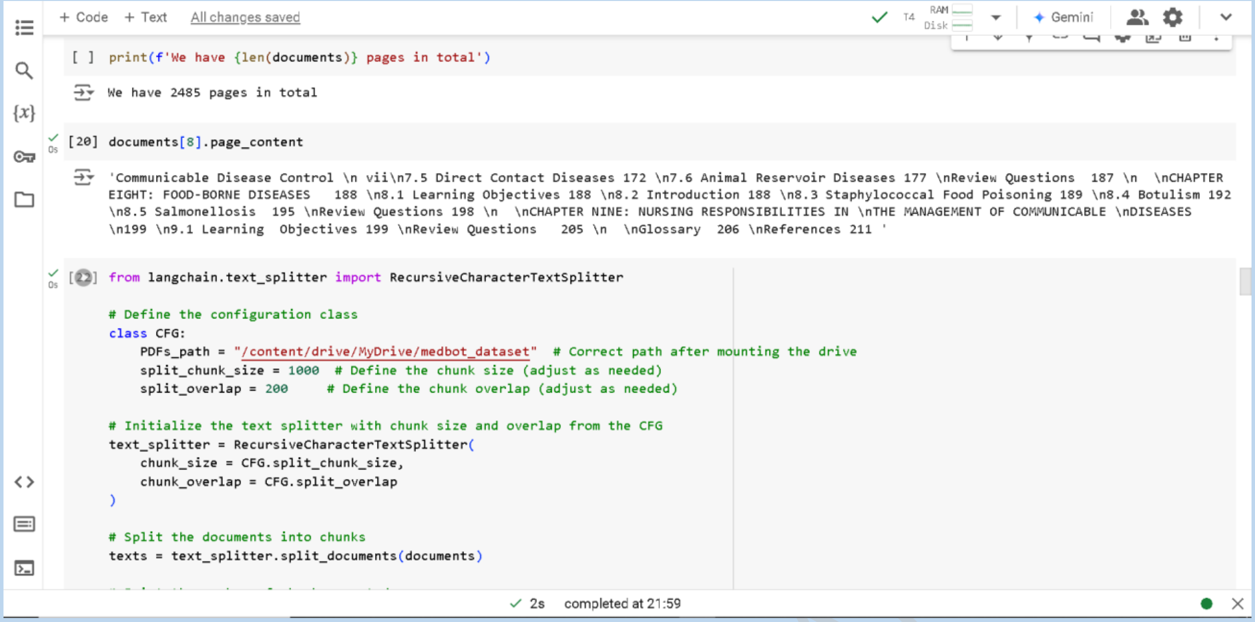
```
'llama2-13b-chat'
```

```
[19] # Ensure the drive is mounted first in Colab
from google.colab import drive
drive.mount('/content/drive')

# Define the configuration class
class CFG:
    PDFs_path = "/content/drive/MyDrive/medbot_dataset"  # Correct path after mounting the drive
```



```
# Verify that the PDFs directory exists
if not os.path.isdir(CFG.PDFs_path):
    raise FileNotFoundError(f"The directory {CFG.PDFs_path} does not exist. Please provide the correct path.")

# Load PDFs from the directory
loader = DirectoryLoader(
    CFG.PDFs_path,
    glob="*.pdf",  # Ensure this pattern matches your PDFs
    loader_cls=PyPDFLoader,
    show_progress=True,
    use_multithreading=True
)

# Try loading the documents
try:
    documents = loader.load()
    print("Documents loaded successfully.")
except Exception as e:
    print(f"Error loading documents: {e}")
    raise


documents = loader.load()
```

```
Mounted at /content/drive
100%|██████████| 7/7 [03:16<00:00, 28.03s/it]
Documents loaded successfully.
100%|██████████| 7/7 [03:15<00:00, 27.93s/it]
```



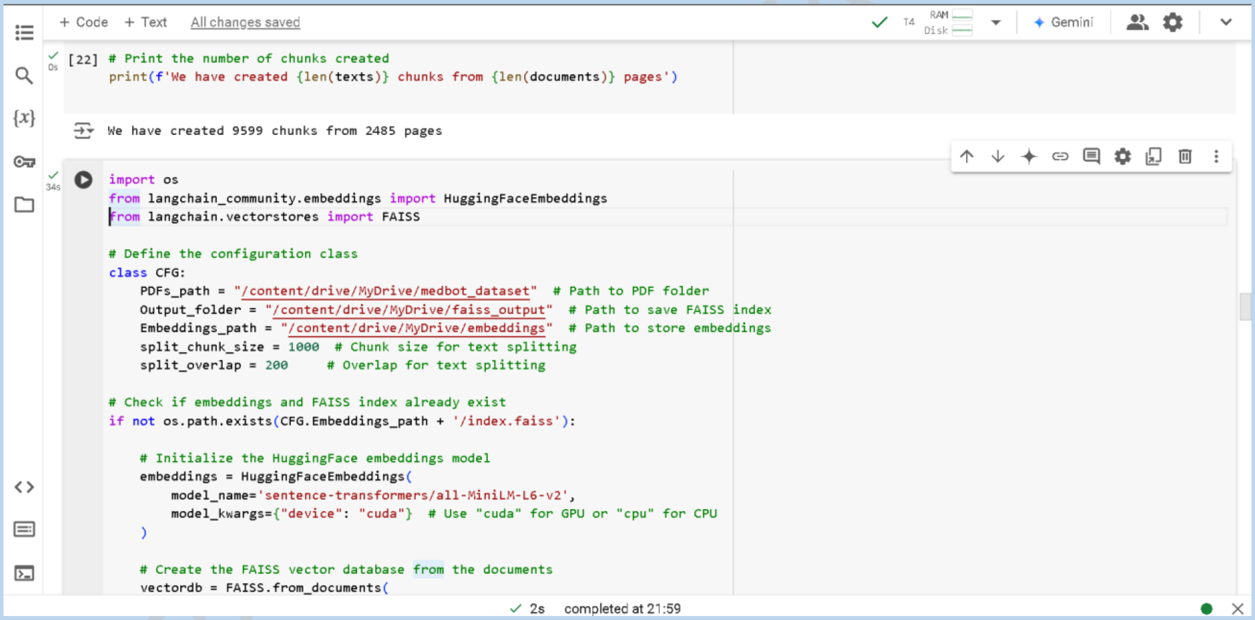
```
[ ] print(f'We have {len(documents)} pages in total')
```

```
We have 2485 pages in total
```

```
[20] documents[8].page_content
```

```
'Communicable Disease Control \n vii\n7.5 Direct Contact Diseases 172 \n7.6 Animal Reservoir Diseases 177 \nReview Questions  187 \n  \nCHAPTER
EIGHT: FOOD-BORNE DISEASES   188 \n8.1 Learning Objectives 188 \n8.2 Introduction 188 \n8.3 Staphylococcal Food Poisoning 189 \n8.4 Botulism 192
\n8.5 Salmonellosis  195 \nReview Questions 198 \n  \nCHAPTER NINE: NURSING RESPONSIBILITIES IN \nTHE MANAGEMENT OF COMMUNICABLE \nDISEASES
\n199 \n9.1 Learning  Objectives 199 \nReview Questions   205 \n  \nGlossary  206 \nReferences 211 '
```

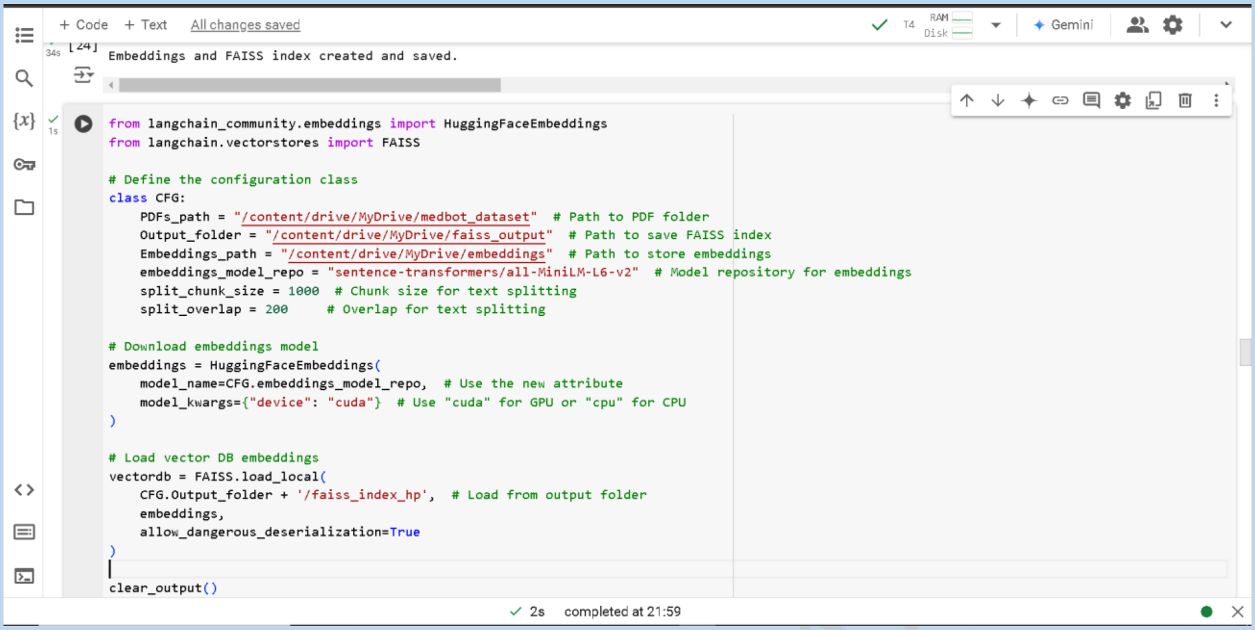
```
from langchain.text_splitter import RecursiveCharacterTextSplitter

# Define the configuration class
class CFG:
    PDFs_path = "/content/drive/MyDrive/medbot_dataset"  # Correct path after mounting the drive
    split_chunk_size = 1000  # Define the chunk size (adjust as needed)
    split_overlap = 200     # Define the chunk overlap (adjust as needed)

# Initialize the text splitter with chunk size and overlap from the CFG
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = CFG.split_chunk_size,
    chunk_overlap = CFG.split_overlap
)

# Split the documents into chunks
texts = text_splitter.split_documents(documents)
```



```
[22] # Print the number of chunks created
     print(f'We have created {len(texts)} chunks from {len(documents)} pages')
```

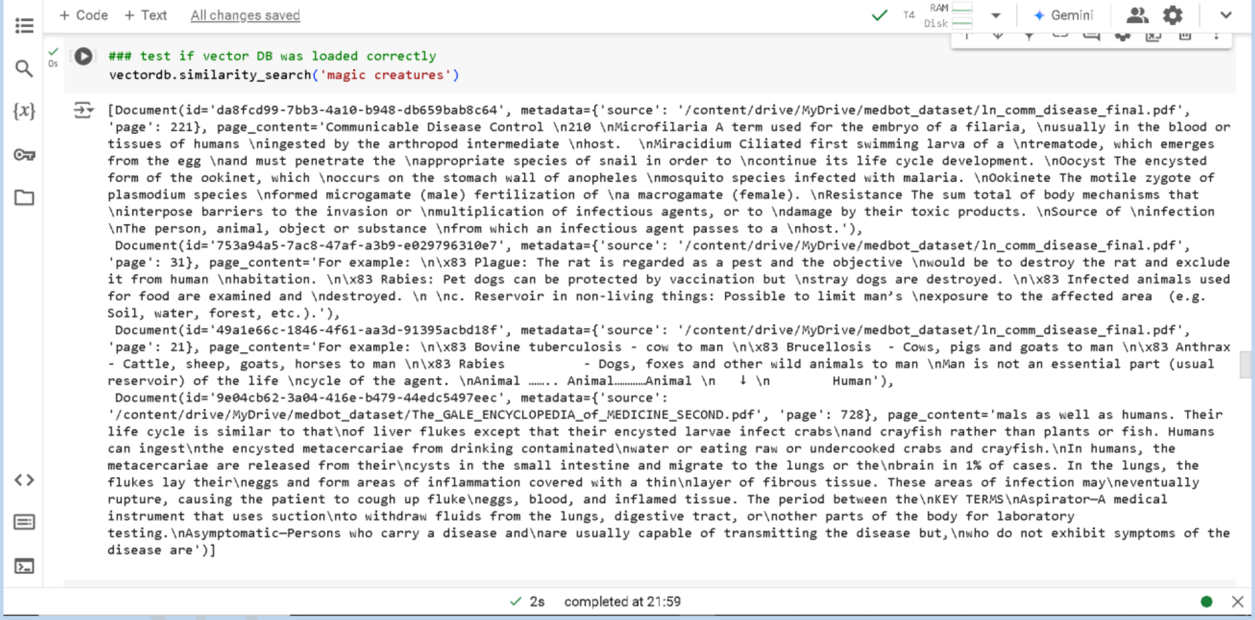
```
We have created 9599 chunks from 2485 pages
```

```
import os
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS

# Define the configuration class
class CFG:
    PDFs_path = "/content/drive/MyDrive/medbot_dataset"  # Path to PDF folder
    Output_folder = "/content/drive/MyDrive/faiss_output"  # Path to save FAISS index
    Embeddings_path = "/content/drive/MyDrive/embeddings"  # Path to store embeddings
    split_chunk_size = 1000  # Chunk size for text splitting
    split_overlap = 200     # Overlap for text splitting

# Check if embeddings and FAISS index already exist
if not os.path.exists(CFG.Embeddings_path + '/index.faiss'):

    # Initialize the HuggingFace embeddings model
    embeddings = HuggingFaceEmbeddings(
        model_name='sentence-transformers/all-MiniLM-L6-v2',
        model_kwargs={"device": "cuda"}  # Use "cuda" for GPU or "cpu" for CPU
    )

    # Create the FAISS vector database from the documents
    vectordb = FAISS.from_documents(
```



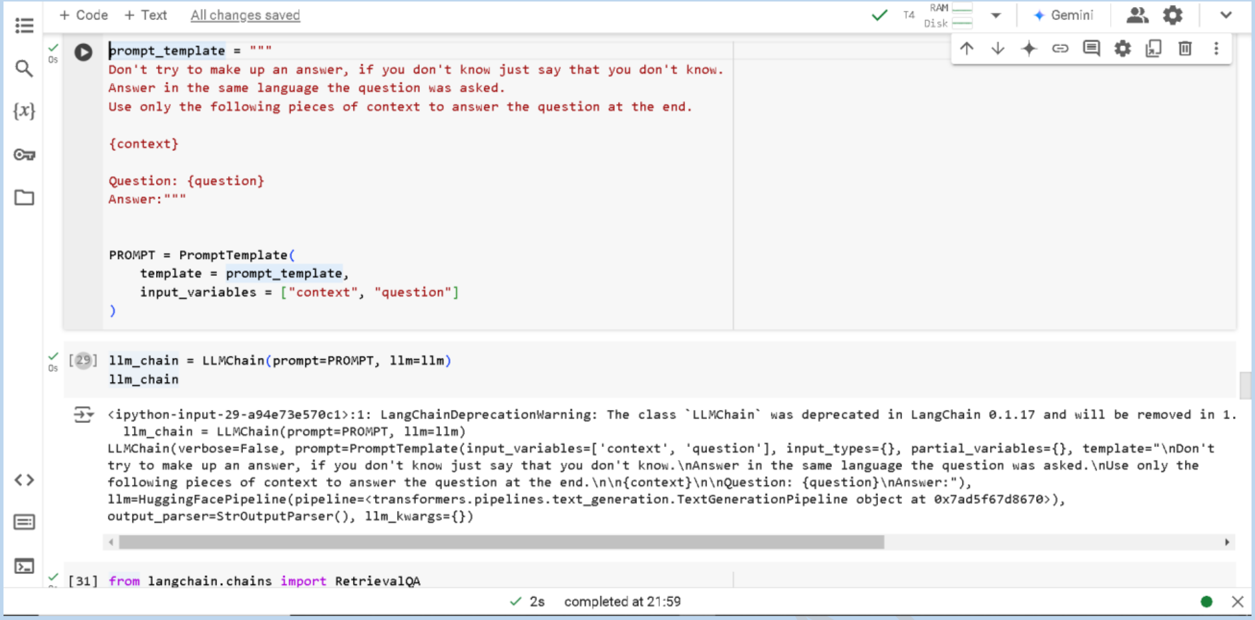
```
Embeddings and FAISS index created and saved.
```

```python
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS

# Define the configuration class
class CFG:
    PDFs_path = "/content/drive/MyDrive/medbot_dataset"  # Path to PDF folder
    Output_folder = "/content/drive/MyDrive/faiss_output"  # Path to save FAISS index
    Embeddings_path = "/content/drive/MyDrive/embeddings"  # Path to store embeddings
    embeddings_model_repo = "sentence-transformers/all-MiniLM-L6-v2"  # Model repository for embeddings
    split_chunk_size = 1000  # Chunk size for text splitting
    split_overlap = 200     # Overlap for text splitting

# Download embeddings model
embeddings = HuggingFaceEmbeddings(
    model_name=CFG.embeddings_model_repo,  # Use the new attribute
    model_kwargs={"device": "cuda"}  # Use "cuda" for GPU or "cpu" for CPU
)

# Load vector DB embeddings
vectordb = FAISS.load_local(
    CFG.Output_folder + '/faiss_index_hp',  # Load from output folder
    embeddings,
    allow_dangerous_deserialization=True
)

clear_output()
```



```python
### test if vector DB was loaded correctly
vectordb.similarity_search('magic creatures')
```

```
[Document(id='da8fcd99-7bb3-4a10-b948-db659bab8c64', metadata={'source': '/content/drive/MyDrive/medbot_dataset/ln_comm_disease_final.pdf',
'page': 221}, page_content='Communicable Disease Control \n210 \nMicrofilaria A term used for the embryo of a filaria, \nusually in the blood or
tissues of humans \ningested by the arthropod intermediate \nhost.  \nMiracidium Ciliated first swimming larva of a \ntrematode, which emerges
from the egg \nand must penetrate the \nappropriate species of snail in order to \ncontinue its life cycle development. \nOocyst The encysted
form of the ookinet, which \noccurs on the stomach wall of anopheles \nmosquito species infected with malaria. \nOokinete The motile zygote of
plasmodium species \nformed microgamate (male) fertilization of \na macrogamate (female). \nResistance The sum total of body mechanisms that
\ninterpose barriers to the invasion or \nmultiplication of infectious agents, or to \ndamage by their toxic products. \nSource of \ninfection
\nThe person, animal, object or substance \nfrom which an infectious agent passes to a \nhost.'),
 Document(id='753a94a5-7ac8-47af-a3b9-e029796310e7', metadata={'source': '/content/drive/MyDrive/medbot_dataset/ln_comm_disease_final.pdf',
'page': 31}, page_content='For example: \n\x83 Plague: The rat is regarded as a pest and the objective \nwould be to destroy the rat and exclude
it from human \nhabitation. \n\x83 Rabies: Pet dogs can be protected by vaccination but \nstray dogs are destroyed. \n\x83 Infected animals used
for food are examined and \ndestroyed. \n \nc. Reservoir in non-living things: Possible to limit man’s \nexposure to the affected area  (e.g.
Soil, water, forest, etc.).'),
 Document(id='49a1e66c-1846-4f61-aa3d-91395acbd18f', metadata={'source': '/content/drive/MyDrive/medbot_dataset/ln_comm_disease_final.pdf',
'page': 21}, page_content='For example: \n\x83 Bovine tuberculosis - cow to man \n\x83 Brucellosis  - Cows, pigs and goats to man \n\x83 Anthrax
- Cattle, sheep, goats, horses to man \n\x83 Rabies          - Dogs, foxes and other wild animals to man \nMan is not an essential part (usual
reservoir) of the life \ncycle of the agent. \nAnimal ……. Animal………Animal \n   ↓ \n        Human'),
 Document(id='9e04cb62-3a04-416e-b479-44edc5497eec', metadata={'source':
'/content/drive/MyDrive/medbot_dataset/The_GALE_ENCYCLOPEDIA_of_MEDICINE_SECOND.pdf', 'page': 728}, page_content='mals as well as humans. Their
life cycle is similar to that\nof liver flukes except that their encysted larvae infect crabs\nand crayfish rather than plants or fish. Humans
can ingest\nthe encysted metacercariae from drinking contaminated\nwater or eating raw or undercooked crabs and crayfish.\nIn humans, the
metacercariae are released from their\ncysts in the small intestine and migrate to the lungs or the\nbrain in 1% of cases. In the lungs, the
flukes lay their\neggs and form areas of inflammation covered with a thin\nlayer of fibrous tissue. These areas of infection may\neventually
rupture, causing the patient to cough up fluke\neggs, blood, and inflamed tissue. The period between the\nKEY TERMS\nAspirator—A medical
instrument that uses suction\nto withdraw fluids from the lungs, digestive tract, or\nother parts of the body for laboratory
testing.\nAsymptomatic—Persons who carry a disease and\nare usually capable of transmitting the disease but,\nwho do not exhibit symptoms of the
disease are')]
```

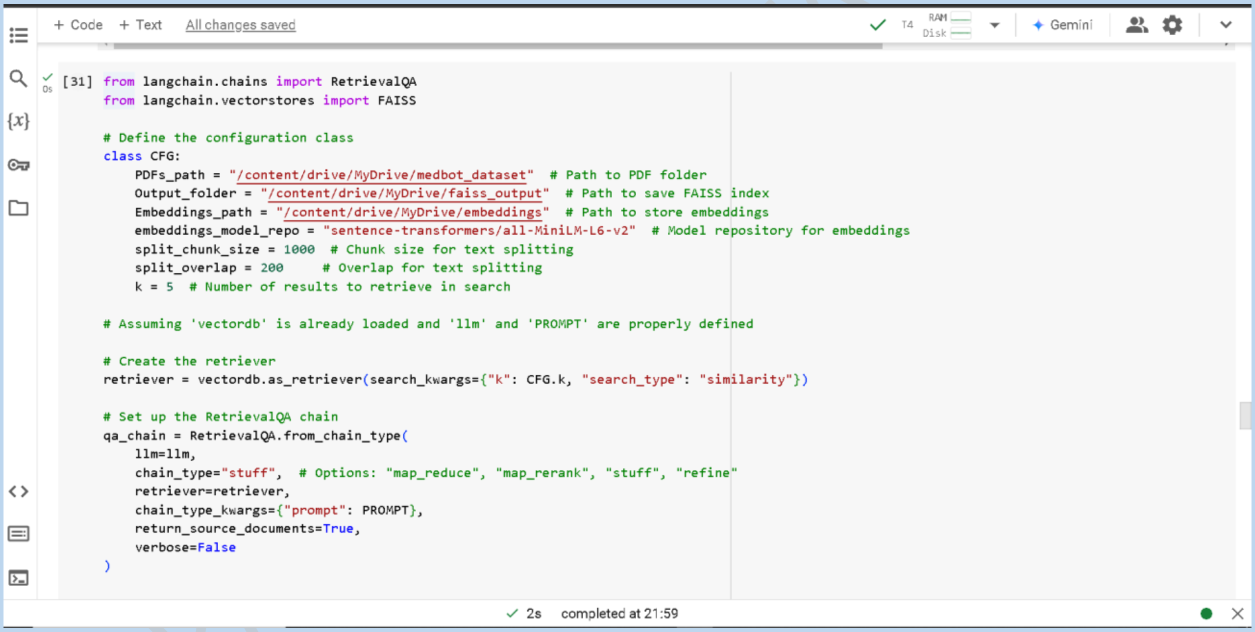


```python
prompt_template = """
Don't try to make up an answer, if you don't know just say that you don't know.
Answer in the same language the question was asked.
Use only the following pieces of context to answer the question at the end.

{context}

Question: {question}
Answer:"""


PROMPT = PromptTemplate(
    template = prompt_template,
    input_variables = ["context", "question"]
)
```

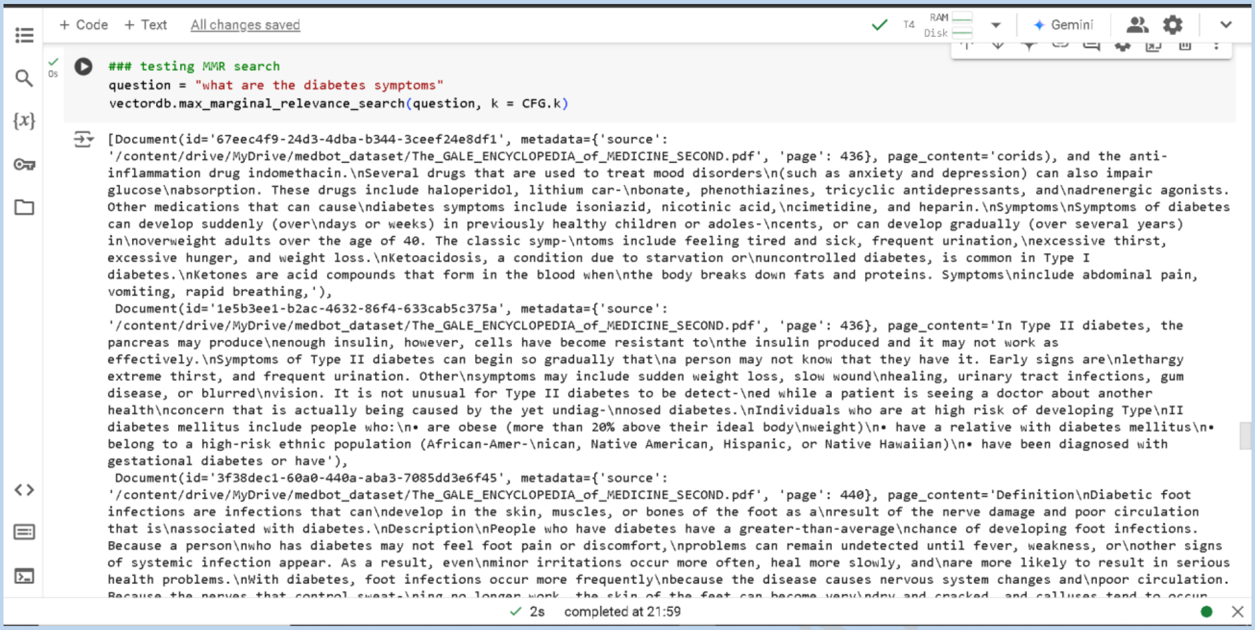
```python
[29] llm_chain = LLMChain(prompt=PROMPT, llm=llm)
     llm_chain
```

```
<ipython-input-29-a94e73e570c1>:1: LangChainDeprecationWarning: The class `LLMChain` was deprecated in LangChain 0.1.17 and will be removed in 1.
  llm_chain = LLMChain(prompt=PROMPT, llm=llm)
LLMChain(verbose=False, prompt=PromptTemplate(input_variables=['context', 'question'], input_types={}, partial_variables={}, template="\nDon't
try to make up an answer, if you don't know just say that you don't know.\nAnswer in the same language the question was asked.\nUse only the
following pieces of context to answer the question at the end.\n\n{context}\n\nQuestion: {question}\nAnswer:"),
llm=HuggingFacePipeline(pipeline=<transformers.pipelines.text_generation.TextGenerationPipeline object at 0x7ad5f67d8670>),
output_parser=StrOutputParser(), llm_kwargs={})
```

```python
[31] from langchain.chains import RetrievalQA
```

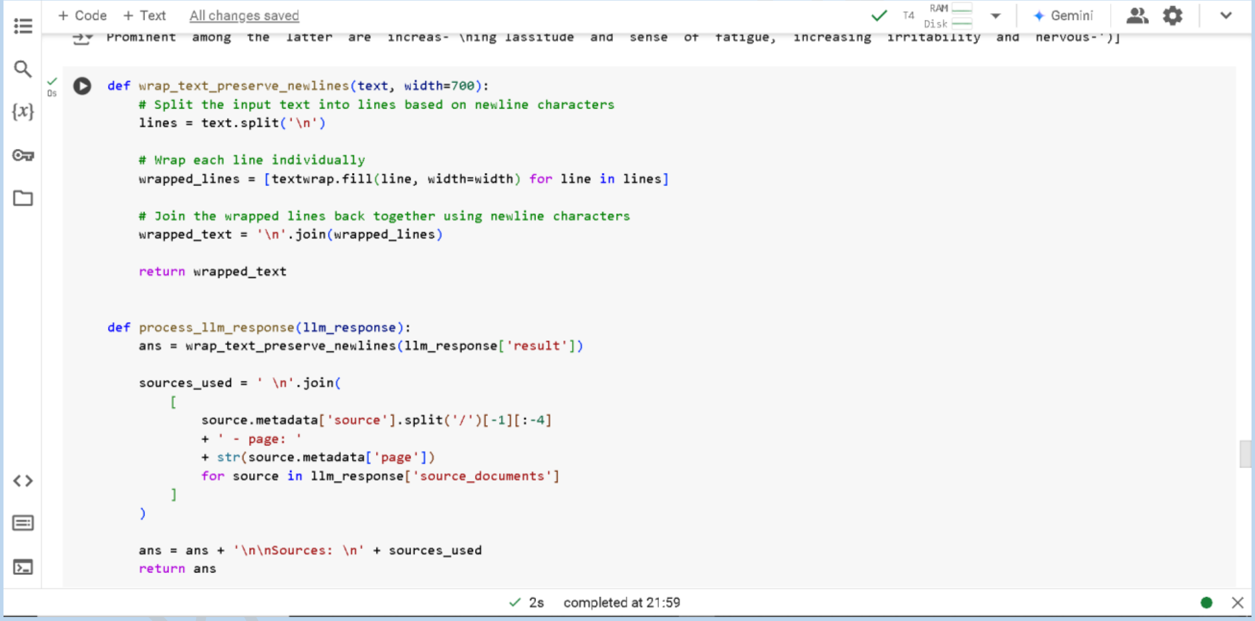


```python
[31] from langchain.chains import RetrievalQA
     from langchain.vectorstores import FAISS

     # Define the configuration class
     class CFG:
         PDFs_path = "/content/drive/MyDrive/medbot_dataset"  # Path to PDF folder
         Output_folder = "/content/drive/MyDrive/faiss_output"  # Path to save FAISS index
         Embeddings_path = "/content/drive/MyDrive/embeddings"  # Path to store embeddings
         embeddings_model_repo = "sentence-transformers/all-MiniLM-L6-v2"  # Model repository for embeddings
         split_chunk_size = 1000  # Chunk size for text splitting
         split_overlap = 200     # Overlap for text splitting
         k = 5  # Number of results to retrieve in search

     # Assuming 'vectordb' is already loaded and 'llm' and 'PROMPT' are properly defined

     # Create the retriever
     retriever = vectordb.as_retriever(search_kwargs={"k": CFG.k, "search_type": "similarity"})

     # Set up the RetrievalQA chain
     qa_chain = RetrievalQA.from_chain_type(
         llm=llm,
         chain_type="stuff",  # Options: "map_reduce", "map_rerank", "stuff", "refine"
         retriever=retriever,
         chain_type_kwargs={"prompt": PROMPT},
         return_source_documents=True,
         verbose=False
     )
```



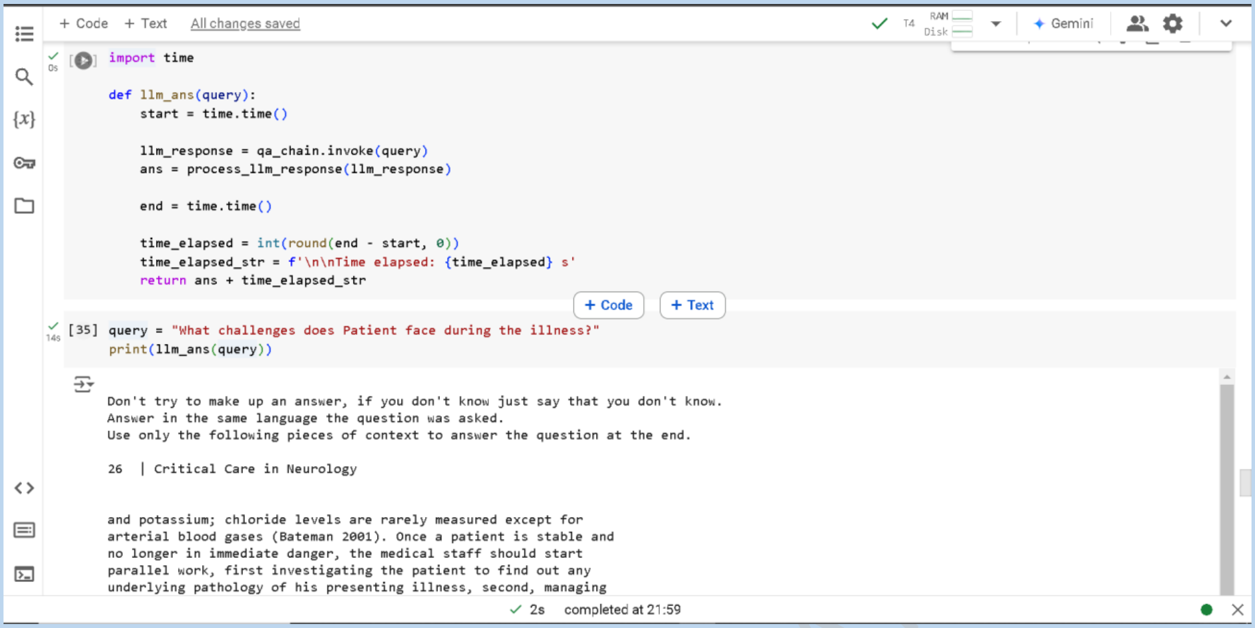
```
### testing MMR search
question = "what are the diabetes symptoms"
vectordb.max_marginal_relevance_search(question, k = CFG.k)
```

```
[Document(id='67eec4f9-24d3-4dba-b344-3ceef24e8df1', metadata={'source':
'/content/drive/MyDrive/medbot_dataset/The_GALE_ENCYCLOPEDIA_of_MEDICINE_SECOND.pdf', 'page': 436}, page_content='corids), and the anti-
inflammation drug indomethacin.\nSeveral drugs that are used to treat mood disorders\n(such as anxiety and depression) can also impair
glucose\nabsorption. These drugs include haloperidol, lithium car-\nbonate, phenothiazines, tricyclic antidepressants, and\nadrenergic agonists.
Other medications that can cause\ndiabetes symptoms include isoniazid, nicotinic acid,\ncimetidine, and heparin.\nSymptoms\nSymptoms of diabetes
can develop suddenly (over\ndays or weeks) in previously healthy children or adoles-\ncents, or can develop gradually (over several years)
in\noverweight adults over the age of 40. The classic symp-\ntoms include feeling tired and sick, frequent urination,\nexcessive thirst,
excessive hunger, and weight loss.\nKetoacidosis, a condition due to starvation or\nuncontrolled diabetes, is common in Type I
diabetes.\nKetones are acid compounds that form in the blood when\nthe body breaks down fats and proteins. Symptoms\ninclude abdominal pain,
vomiting, rapid breathing,'),
 Document(id='1e5b3ee1-b2ac-4632-86f4-633cab5c375a', metadata={'source':
'/content/drive/MyDrive/medbot_dataset/The_GALE_ENCYCLOPEDIA_of_MEDICINE_SECOND.pdf', 'page': 436}, page_content='In Type II diabetes, the
pancreas may produce\nenough insulin, however, cells have become resistant to\nthe insulin produced and it may not work as
effectively.\nSymptoms of Type II diabetes can begin so gradually that\na person may not know that they have it. Early signs are\nlethargy
extreme thirst, and frequent urination. Other\nsymptoms may include sudden weight loss, slow wound\nhealing, urinary tract infections, gum
disease, or blurred\nvision. It is not unusual for Type II diabetes to be detect-\ned while a patient is seeing a doctor about another
health\nconcern that is actually being caused by the yet undiag-\nnosed diabetes.\nIndividuals who are at high risk of developing Type\nII
diabetes mellitus include people who:\n• are obese (more than 20% above their ideal body\nweight)\n• have a relative with diabetes mellitus\n•
belong to a high-risk ethnic population (African-Amer-\nican, Native American, Hispanic, or Native Hawaiian)\n• have been diagnosed with
gestational diabetes or have'),
 Document(id='3f38dec1-60a0-440a-aba3-7085dd3e6f45', metadata={'source':
'/content/drive/MyDrive/medbot_dataset/The_GALE_ENCYCLOPEDIA_of_MEDICINE_SECOND.pdf', 'page': 440}, page_content='Definition\nDiabetic foot
infections are infections that can\ndevelop in the skin, muscles, or bones of the foot as a\nresult of the nerve damage and poor circulation
that is\nassociated with diabetes.\nDescription\nPeople who have diabetes have a greater-than-average\nchance of developing foot infections.
Because a person\nwho has diabetes may not feel foot pain or discomfort,\nproblems can remain undetected until fever, weakness, or\nother signs
of systemic infection appear. As a result, even\nminor irritations occur more often, heal more slowly, and\nare more likely to result in serious
health problems.\nWith diabetes, foot infections occur more frequently\nbecause the disease causes nervous system changes and\npoor circulation.
Because the nerves that control sweat-\ning no longer work, the skin of the feet can become very\ndry and cracked, and calluses tend to occur
```



```
Prominent among the latter are increas- \ning lassitude and sense of fatigue, increasing irritability and nervous-')]
```

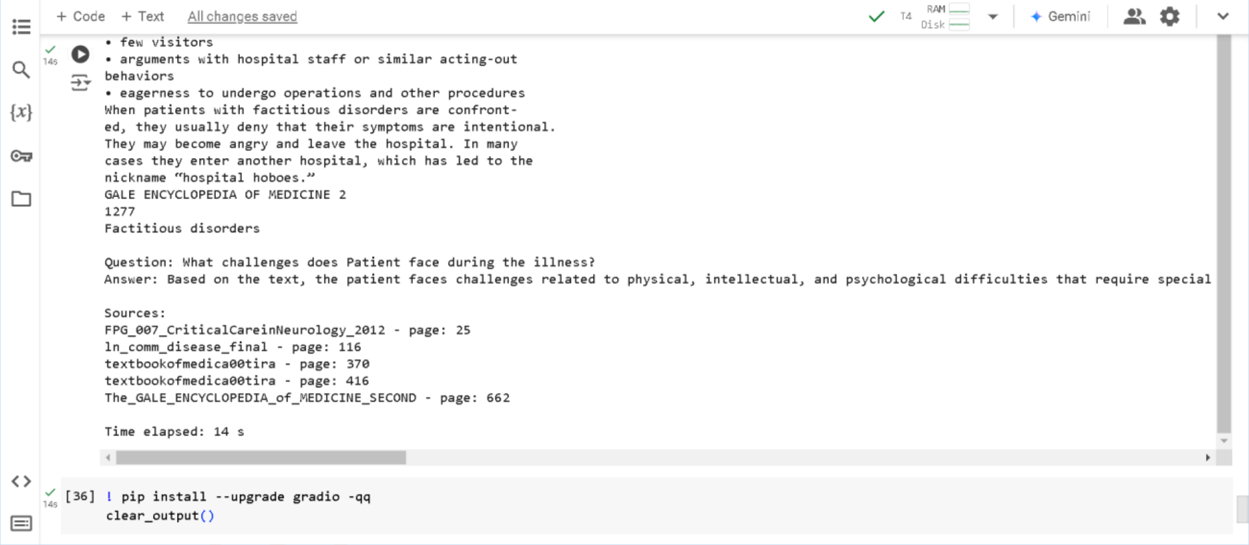
```
def wrap_text_preserve_newlines(text, width=700):
    # Split the input text into lines based on newline characters
    lines = text.split('\n')

    # Wrap each line individually
    wrapped_lines = [textwrap.fill(line, width=width) for line in lines]

    # Join the wrapped lines back together using newline characters
    wrapped_text = '\n'.join(wrapped_lines)

    return wrapped_text


def process_llm_response(llm_response):
    ans = wrap_text_preserve_newlines(llm_response['result'])

    sources_used = ' \n'.join(
        [
            source.metadata['source'].split('/')[-1][:-4]
            + ' - page: '
            + str(source.metadata['page'])
            for source in llm_response['source_documents']
        ]
    )

    ans = ans + '\n\nSources: \n' + sources_used
    return ans
```



```
import time

def llm_ans(query):
    start = time.time()

    llm_response = qa_chain.invoke(query)
    ans = process_llm_response(llm_response)

    end = time.time()

    time_elapsed = int(round(end - start, 0))
    time_elapsed_str = f'\n\nTime elapsed: {time_elapsed} s'
    return ans + time_elapsed_str
```

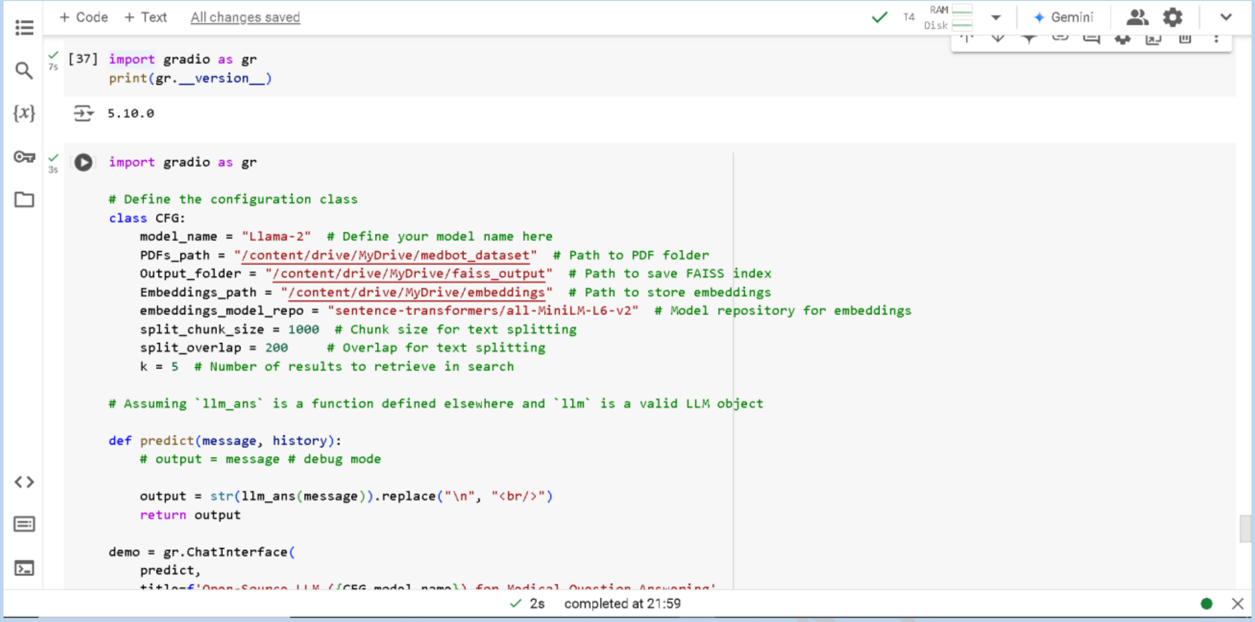
```
[35] query = "What challenges does Patient face during the illness?"
     print(llm_ans(query))
```

```
Don't try to make up an answer, if you don't know just say that you don't know.
Answer in the same language the question was asked.
Use only the following pieces of context to answer the question at the end.

26  | Critical Care in Neurology


and potassium; chloride levels are rarely measured except for
arterial blood gases (Bateman 2001). Once a patient is stable and
no longer in immediate danger, the medical staff should start
parallel work, first investigating the patient to find out any
underlying pathology of his presenting illness, second, managing
```



```
• few visitors
• arguments with hospital staff or similar acting-out
behaviors
• eagerness to undergo operations and other procedures
When patients with factitious disorders are confront-
ed, they usually deny that their symptoms are intentional.
They may become angry and leave the hospital. In many
cases they enter another hospital, which has led to the
nickname "hospital hoboes."
GALE ENCYCLOPEDIA OF MEDICINE 2
1277
Factitious disorders

Question: What challenges does Patient face during the illness?
Answer: Based on the text, the patient faces challenges related to physical, intellectual, and psychological difficulties that require special

Sources:
FPG_007_CriticalCareinNeurology_2012 - page: 25
ln_comm_disease_final - page: 116
textbookofmedica00tira - page: 370
textbookofmedica00tira - page: 416
The_GALE_ENCYCLOPEDIA_of_MEDICINE_SECOND - page: 662

Time elapsed: 14 s
```

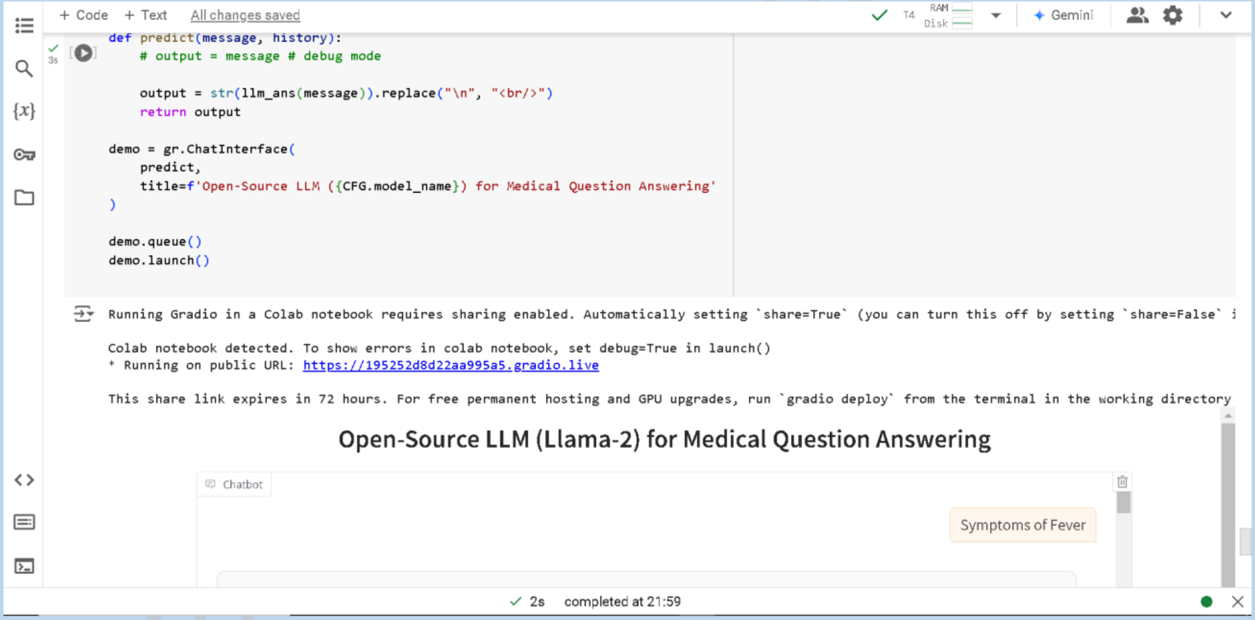
```
[36] ! pip install --upgrade gradio -qq
     clear_output()
```



```
[37] import gradio as gr
     print(gr.__version__)
```

```
5.10.0
```

```
import gradio as gr

# Define the configuration class
class CFG:
    model_name = "Llama-2"  # Define your model name here
    PDFs_path = "/content/drive/MyDrive/medbot_dataset"  # Path to PDF folder
    Output_folder = "/content/drive/MyDrive/faiss_output"  # Path to save FAISS index
    Embeddings_path = "/content/drive/MyDrive/embeddings"  # Path to store embeddings
    embeddings_model_repo = "sentence-transformers/all-MiniLM-L6-v2"  # Model repository for embeddings
    split_chunk_size = 1000  # Chunk size for text splitting
    split_overlap = 200     # Overlap for text splitting
    k = 5  # Number of results to retrieve in search

# Assuming `llm_ans` is a function defined elsewhere and `llm` is a valid LLM object

def predict(message, history):
    # output = message # debug mode

    output = str(llm_ans(message)).replace("\n", "<br/>")
    return output

demo = gr.ChatInterface(
    predict,
    title=f'Open-Source LLM ({CFG.model_name}) for Medical Question Answering'
```

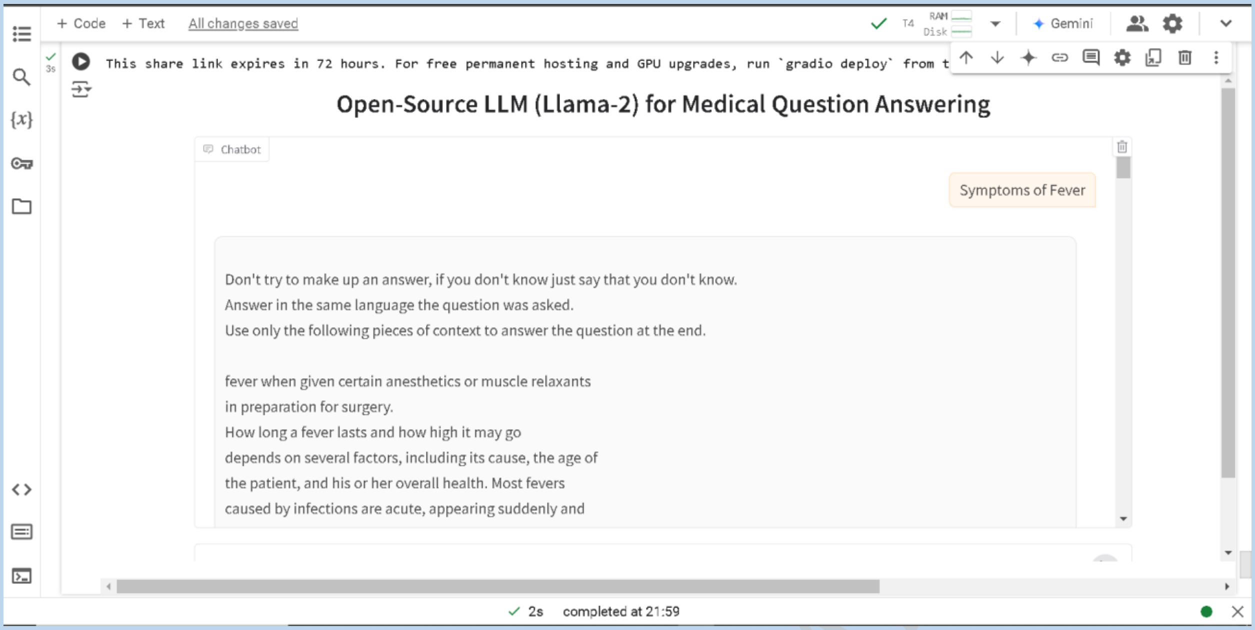


```
def predict(message, history):
    # output = message # debug mode

    output = str(llm_ans(message)).replace("\n", "<br/>")
    return output

demo = gr.ChatInterface(
    predict,
    title=f'Open-Source LLM ({CFG.model_name}) for Medical Question Answering'
)

demo.queue()
demo.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `share=False` i

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://195252d8d22aa995a5.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory
```

Open-Source LLM (Llama-2) for Medical Question Answering

Output_screen:



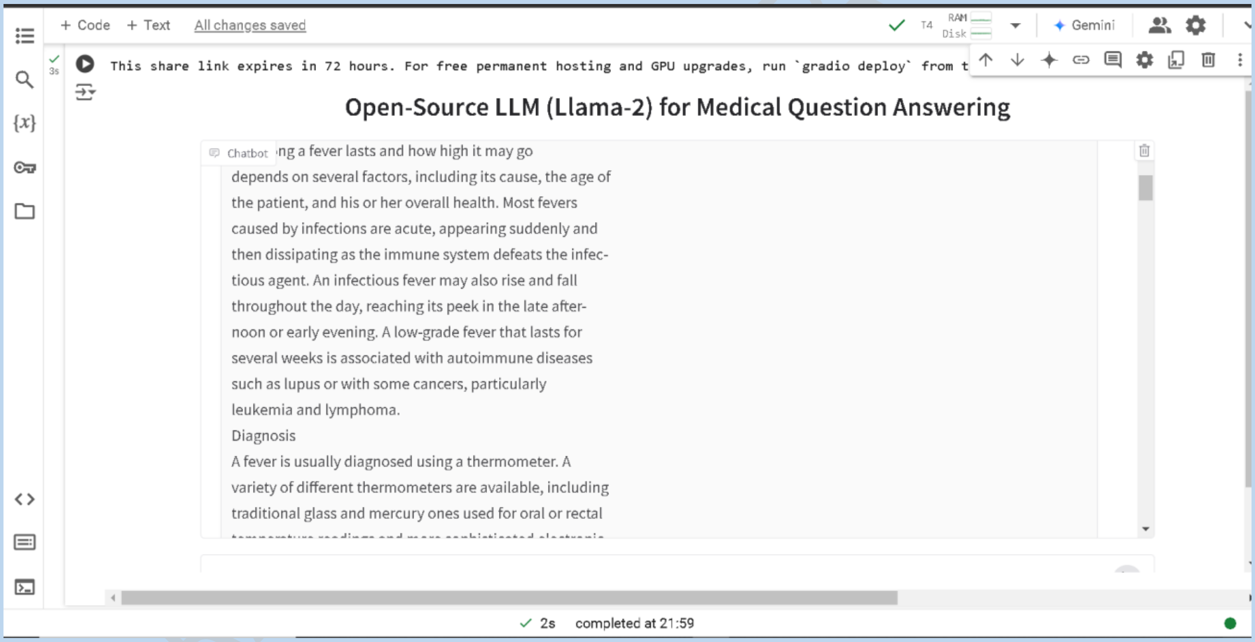
```
This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from t
```

Open-Source LLM (Llama-2) for Medical Question Answering

Symptoms of Fever

Don't try to make up an answer, if you don't know just say that you don't know.
Answer in the same language the question was asked.
Use only the following pieces of context to answer the question at the end.

fever when given certain anesthetics or muscle relaxants
in preparation for surgery.
How long a fever lasts and how high it may go
depends on several factors, including its cause, the age of
the patient, and his or her overall health. Most fevers
caused by infections are acute, appearing suddenly and



```
This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from t
```

Open-Source LLM (Llama-2) for Medical Question Answering

ng a fever lasts and how high it may go
depends on several factors, including its cause, the age of
the patient, and his or her overall health. Most fevers
caused by infections are acute, appearing suddenly and
then dissipating as the immune system defeats the infectious agent. An infectious fever may also rise and fall
throughout the day, reaching its peek in the late afternoon or early evening. A low-grade fever that lasts for
several weeks is associated with autoimmune diseases
such as lupus or with some cancers, particularly
leukemia and lymphoma.
Diagnosis
A fever is usually diagnosed using a thermometer. A
variety of different thermometers are available, including
traditional glass and mercury ones used for oral or rectal
temperature readings and more sophisticated electronic