```
!pip install -q transformers datasets evaluate rouge-score peft accelerate langchain sentence-transformers faiss-gpu
!pip install -q bitsandbytes
```

```
# ## 1. Setup and Configuration
```

```
import torch
from transformers import (
    AutoTokenizer,
    AutoModelForSeq2SeqLM,
    Seq2SeqTrainingArguments,
    Seq2SeqTrainer,
    DataCollatorForSeq2Seq,
    BitsAndBytesConfig
)
```

```
  Preparing metadata (setup.py) ... done
ERROR: Could not find a version that satisfies the requirement faiss-gpu (from versions: none)
ERROR: No matching distribution found for faiss-gpu
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.1/76.1 MB 9.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 57.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 45.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 41.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 11.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 41.5 MB/s eta 0:00:00
```

```
!pip install datasets
```

Show hidden output

```
!pip install -q transformers datasets evaluate rouge-score peft accelerate langchain sentence-transformers faiss-gpu
!pip install -q bitsandbytes
```

```
  Preparing metadata (setup.py) ... done
ERROR: Could not find a version that satisfies the requirement faiss-gpu (from versions: none)
ERROR: No matching distribution found for faiss-gpu
```

```
!pip install evaluate
```

Show hidden output

```
!pip install -U langchain-community
```

Show hidden output

```
from peft import LoraConfig, get_peft_model, TaskType, PeftModel, PeftConfig
from datasets import load_dataset, Dataset
import evaluate
import numpy as np
import pandas as pd
from tqdm import tqdm
import os
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain.document_loaders import CSVLoader
from langchain.text_splitter import CharacterTextSplitter

# Configuration
MODEL_NAME = "google/flan-t5-base"
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
MAX_INPUT_LENGTH = 512
MAX_TARGET_LENGTH = 256
BATCH_SIZE = 8
LEARNING_RATE = 3e-4
NUM_EPOCHS = 5
LORA_R = 16
```

```
LORA_ALPHA = 32
LORA_DROPOUT = 0.05

# Initialize tokenizer
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

# Quantization config for memory efficiency - DISABLED FOR CPU
# bnb_config = BitsAndBytesConfig(
#     load_in_4bit=True,
#     bnb_4bit_quant_type="nf4",
#     bnb_4bit_compute_dtype=torch.float16,
#     bnb_4bit_use_double_quant=False,
# )

# Load base model without quantization
model = AutoModelForSeq2SeqLM.from_pretrained(
    MODEL_NAME,
    # quantization_config=bnb_config, # Removing quantization
    device_map="auto"
)
```

```
tokenizer_config.json: 100%  2.54k/2.54k [00:00<00:00, 122kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfo
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
spiece.model: 100%  792k/792k [00:00<00:00, 5.22MB/s]
tokenizer.json: 100%  2.42M/2.42M [00:00<00:00, 8.36MB/s]
special_tokens_map.json: 100%  2.20k/2.20k [00:00<00:00, 76.3kB/s]
config.json: 100%  1.40k/1.40k [00:00<00:00, 58.1kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfo
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
model.safetensors: 100%  990M/990M [00:09<00:00, 127MB/s]
generation_config.json: 100%  147/147 [00:00<00:00, 2.82kB/s]
```

```
# ## 2. Data Preparation

# %%
# Sample healthcare dataset - in practice, you would use your own proprietary data
healthcare_data = {
    "input_text": [
        "Patient presents with fever, cough, and fatigue for 3 days. No recent travel history. What could be the possible diagnosis?",
        "45-year-old male with hypertension and diabetes reports chest pain. What should be the next steps?",
        "Patient asks about side effects of metformin.",
        "How to manage stress and anxiety during recovery from surgery?",
        "Explain the difference between type 1 and type 2 diabetes to a patient."
    ],
    "output_text": [
        "Based on symptoms, possible diagnoses include influenza, COVID-19, or common cold. Recommend COVID test, rest, hydration, and fever
        "This could indicate angina or heart attack. Advise immediate ECG, troponin test, and consider ER visit. Monitor vitals closely.",
        "Common side effects include gastrointestinal issues like nausea and diarrhea. Rare but serious side effects can include lactic acic
        "Recommend deep breathing exercises, light physical activity when cleared by doctor, maintaining a routine, and connecting with supp
        "Type 1 is an autoimmune condition where the pancreas produces little to no insulin. Type 2 is metabolic where the body becomes resi
    ]
}

# Convert to HuggingFace dataset
dataset = Dataset.from_dict(healthcare_data)
dataset = dataset.train_test_split(test_size=0.2)

# %%
# Preprocessing function
def preprocess_function(examples):
    inputs = [text for text in examples["input_text"]]
    targets = [text for text in examples["output_text"]]

    model_inputs = tokenizer(
        inputs,
        max_length=MAX_INPUT_LENGTH,
        truncation=True,
        padding="max_length"
```

```
    )

    labels = tokenizer(
        targets,
        max_length=MAX_TARGET_LENGTH,
        truncation=True,
        padding="max_length"
    )

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

# Apply preprocessing
tokenized_dataset = dataset.map(
    preprocess_function,
    batched=True,
    remove_columns=dataset["train"].column_names
)

# %% [markdown]
```

```
Map: 100%                                    4/4 [00:00<00:00, 38.53 examples/s]

Map: 100%                                    1/1 [00:00<00:00, 17.01 examples/s]
```

```
!pip install rouge_scorea
```

```
ERROR: Could not find a version that satisfies the requirement rouge_scorea (from versions: none)
ERROR: No matching distribution found for rouge_scorea
```

```
!pip install rouge-score  # Install the missing rouge_score package
```

Show hidden output

```
# ## 3. Client-Centric Fine-Tuning with LoRA

# Define LoRA config
lora_config = LoraConfig(
    r=LORA_R,
    lora_alpha=LORA_ALPHA,
    lora_dropout=LORA_DROPOUT,
    bias="none",
    task_type=TaskType.SEQ_2_SEQ_LM,
    target_modules=["q", "v"]  # Targeting query and value layers
)

# Prepare PEFT model
peft_model = get_peft_model(model, lora_config)
peft_model.print_trainable_parameters()

# Data collator
data_collator = DataCollatorForSeq2Seq(
    tokenizer=tokenizer,
    model=peft_model
)

# Evaluation metrics
import evaluate
from rouge_score import rouge_scorer # Import the necessary module for rouge

rouge = evaluate.load("rouge") # Now evaluate.load should work correctly


def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)

    # Replace -100 in the labels as we can't decode them
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

    # Compute ROUGE scores
    result = rouge.compute(
        predictions=decoded_preds,
        references=decoded_labels,
```

```
        use_stemmer=True
    )

    # Add mean generated length
    prediction_lens = [np.count_nonzero(pred != tokenizer.pad_token_id) for pred in predictions]
    result["gen_len"] = np.mean(prediction_lens)

    return {k: round(v, 4) for k, v in result.items()}

# %%
# Training arguments
training_args = Seq2SeqTrainingArguments(
    output_dir="healthcare-assistant-output",
    # evaluation_strategy="epoch", # This argument is deprecated
    learning_rate=LEARNING_RATE,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=NUM_EPOCHS,
    predict_with_generate=True,
    fp16=True,
    report_to="none",
    # Use evaluation_strategy instead of evaluation_strategy
    eval_steps = 100 # Or any desired number of steps
)

# Trainer
trainer = Seq2SeqTrainer(
    model=peft_model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics
)

# %%
# Train the model
trainer.train()

# Save the model
peft_model_id = "healthcare-assistant-lora"
trainer.model.save_pretrained(peft_model_id)
tokenizer.save_pretrained(peft_model_id)

# %% [markdown]
```

```
WARNING:bitsandbytes.cextension:The installed version of bitsandbytes was compiled without GPU support. 8-bit optimizers, 8-bit multipli
trainable params: 1,769,472 || all params: 249,347,328 || trainable%: 0.7096
Downloading builder script: 100%                                    6.27k/6.27k [00:00<00:00, 408kB/s]
<ipython-input-14-c76d4ba611f1>:72: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Seq2SeqTrainer.__
  trainer = Seq2SeqTrainer(
No label_names provided for model class `PeftModelForSeq2SeqLM`. Since `PeftModel` hides base models input arguments, if label_names is
Passing a tuple of `past_key_values` is deprecated and will be removed in Transformers v4.48.0. You should pass an instance of `EncoderD
[5/5 02:08, Epoch 5/5]
```

Step	Training Loss

```
('healthcare-assistant-lora/tokenizer_config.json',
 'healthcare-assistant-lora/special_tokens_map.json',
 'healthcare-assistant-lora/spiece.model',
 'healthcare-assistant-lora/added_tokens.json',
 'healthcare-assistant-lora/tokenizer.json')
```

```
!pip install faiss-cpu
```

```
Collecting faiss-cpu
  Downloading faiss_cpu-1.10.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (4.4 kB)
Requirement already satisfied: numpy<3.0,>=1.25.0 in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (2.0.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from faiss-cpu) (24.2)
Downloading faiss_cpu-1.10.0-cp311-cp311-manylinux_2_28_x86_64.whl (30.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 30.7/30.7 MB 25.4 MB/s eta 0:00:00
Installing collected packages: faiss-cpu
Successfully installed faiss-cpu-1.10.0
```

```
# ## 4. Knowledge Base Integration (RAG)

# %%
# Sample healthcare knowledge base (in practice, use your own data)
healthcare_knowledge = [
    {"content": "Metformin is a first-line medication for type 2 diabetes that improves insulin sensitivity."},
    {"content": "Normal blood pressure range is less than 120/80 mmHg."},
    {"content": "Symptoms of COVID-19 include fever, cough, and loss of taste or smell."},
    {"content": "The ABCDE approach is used for melanoma detection: Asymmetry, Border irregularity, Color variation, Diameter >6mm, Evolving
    {"content": "CDC recommends at least 150 minutes of moderate exercise per week for adults."}
]

# Create a DataFrame and save as CSV
df = pd.DataFrame(healthcare_knowledge)
df.to_csv("healthcare_knowledge.csv", index=False)

# %%
# Load and process knowledge base
loader = CSVLoader(file_path="healthcare_knowledge.csv")
documents = loader.load()

# Split documents into chunks
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
docs = text_splitter.split_documents(documents)

# Create embeddings and vector store
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")
vectorstore = FAISS.from_documents(docs, embeddings)
vectorstore.save_local("healthcare_faiss_index")

# %% [markdown]
# ## 5. Inference with RAG and Client Adaptation

# %%
from transformers import pipeline
from langchain.llms import HuggingFacePipeline
from langchain.chains import RetrievalQA

# Load fine-tuned model
config = PeftConfig.from_pretrained(peft_model_id)
base_model = AutoModelForSeq2SeqLM.from_pretrained(
    config.base_model_name_or_path,
    # Remove the quantization_config to load model without quantization
    # quantization_config=bnb_config,
    device_map="auto"
)
model = PeftModel.from_pretrained(base_model, peft_model_id)
model.eval()

# Create text generation pipeline
pipe = pipeline(
    "text2text-generation",
    model=model,
    tokenizer=tokenizer,
    # Remove or comment out the 'device' argument
    # device=DEVICE,
    max_length=MAX_TARGET_LENGTH
)

# Create LangChain LLM wrapper
llm = HuggingFacePipeline(pipeline=pipe)

# Load vector store with allow_dangerous_deserialization=True
vectorstore = FAISS.load_local("healthcare_faiss_index", embeddings, allow_dangerous_deserialization=True)

# Create retrieval chain
retriever = vectorstore.as_retriever(search_kwargs={"k": 3})
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    return_source_documents=True
)
```

```
# %%
def healthcare_assistant(query, client_context=None):
    # Add client context to query if provided
    if client_context:
        query = f"Client context: {client_context}\n\nQuestion: {query}"

    # Get response
    result = qa_chain(query)

    return {
        "response": result["result"],
        "sources": [doc.metadata["source"] for doc in result["source_documents"]]
    }

# %% [markdown]
```

```
<ipython-input-16-e9ff4c6fd92f>:27: LangChainDeprecationWarning: The class `HuggingFaceEmbeddings` was deprecated in LangChain 0.2.2 and
  embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")
modules.json: 100%  349/349 [00:00<00:00, 32.7kB/s]
config_sentence_transformers.json: 100%  116/116 [00:00<00:00, 8.88kB/s]
README.md: 100%  10.4k/10.4k [00:00<00:00, 842kB/s]
sentence_bert_config.json: 100%  53.0/53.0 [00:00<00:00, 1.82kB/s]
config.json: 100%  571/571 [00:00<00:00, 37.3kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfo
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
model.safetensors: 100%  438M/438M [00:06<00:00, 98.7MB/s]
tokenizer_config.json: 100%  363/363 [00:00<00:00, 7.72kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 4.72MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 4.57MB/s]
special_tokens_map.json: 100%  239/239 [00:00<00:00, 4.75kB/s]
config.json: 100%  190/190 [00:00<00:00, 3.33kB/s]
Device set to use cpu
The model 'PeftModelForSeq2SeqLM' is not supported for text2text-generation. Supported models are ['BartForConditionalGeneration', 'BigB
<ipython-input-16-e9ff4c6fd92f>:61: LangChainDeprecationWarning: The class `HuggingFacePipeline` was deprecated in LangChain 0.0.37 and
  llm = HuggingFacePipeline(pipeline=pipe)
```

```
# ## 6. Example Usage

# %%
# General healthcare question
response = healthcare_assistant("What are the first-line treatments for type 2 diabetes?")
print("Response:", response["response"])
print("Sources:", response["sources"])

# %%
# Client-specific question (simulating client context)
client_context = "55-year-old female with history of hypertension and prediabetes. Allergic to sulfa drugs."
response = healthcare_assistant(
    "What medication would you recommend for my condition?",
    client_context=client_context
)
print("Response:", response["response"])
print("Sources:", response["sources"])

# %% [markdown]
# ## 7. Client-Specific Fine-Tuning Function

# %%
def client_specific_finetuning(client_data, client_id):
    """
    Performs client-specific fine-tuning on the base model.

    Args:
        client_data: Dict with "input_text" and "output_text" keys containing client-specific examples
        client_id: Unique identifier for the client
    """
    # Prepare dataset
```

```
    client_dataset = Dataset.from_dict(client_data)
    tokenized_client_data = client_dataset.map(
        preprocess_function,
        batched=True,
        remove_columns=client_dataset.column_names
    )

    # Training arguments for client-specific tuning
    client_args = Seq2SeqTrainingArguments(
        output_dir=f"client-models/{client_id}",
        per_device_train_batch_size=2,  # Smaller batch size for client-specific data
        num_train_epochs=3,  # Fewer epochs to avoid overfitting
        save_strategy="no",
        report_to="none"
    )

    # Trainer for client-specific data
    client_trainer = Seq2SeqTrainer(
        model=peft_model,
        args=client_args,
        train_dataset=tokenized_client_data,
        tokenizer=tokenizer,
        data_collator=data_collator
    )

    # Perform fine-tuning
    client_trainer.train()

    # Save client-specific adapter
    client_model_path = f"client-models/{client_id}"
    client_trainer.model.save_pretrained(client_model_path)

    return client_model_path

# %%
# Example of client-specific fine-tuning
client_data = {
    "input_text": [
        "What's the best time to take my blood pressure medication given I work night shifts?",
        "Can I take my diabetes medication with my new heart medication?",
        "How should I adjust my medication when fasting?"
    ],
    "output_text": [
        "Given your night shift schedule, take your blood pressure medication at the start of your 'day' (when you wake up for work).",
        "Your metformin can be taken with most heart medications, but space out your lisinopril by 2 hours from your diuretic.",
        "When fasting, take your morning dose with your first meal and evening dose with your last meal before sunset."
    ]
}

# Perform client-specific fine-tuning
client_id = "client_123"
client_model_path = client_specific_finetuning(client_data, client_id)
print(f"Client-specific model saved to {client_model_path}")

# %% [markdown]
# ## 8. Loading Client-Specific Models

# %%
def load_client_model(base_model, client_model_path):
    """Loads a client-specific adapter onto the base model"""
    client_model = PeftModel.from_pretrained(base_model, client_model_path)
    client_model.eval()
    return client_model

# Example usage:
# client_model = load_client_model(base_model, "client-models/client_123")
# Then use this client_model for inference for that specific client

# %% [markdown]
```

```
<ipython-input-16-e9ff4c6fd92f>:82: LangChainDeprecationWarning: The method `Chain.__call__` was deprecated in langchain 0.1.0 and will
  result = qa_chain(query)
Response: Metformin
Sources: ['healthcare_knowledge.csv', 'healthcare_knowledge.csv', 'healthcare_knowledge.csv']
Response: Metformin
Sources: ['healthcare_knowledge.csv', 'healthcare_knowledge.csv', 'healthcare_knowledge.csv']
```

Map: 100% 3/3 [00:00<00:00, 47.35 examples/s]

```
<ipython-input-17-cfbcf1f9ba60>:49: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Seq2SeqTrainer.__
  client_trainer = Seq2SeqTrainer(
No label_names provided for model class `PeftModelForSeq2SeqLM`. Since `PeftModel` hides base models input arguments, if label_names is
```

[6/6 01:12, Epoch 3/3]

Step	Training Loss

```
Client-specific model saved to client-models/client_123
```

```
!pip install fastapi
```

Show hidden output

```
!pip install uvicorn
```

```
Collecting uvicorn
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.11/dist-packages (from uvicorn) (8.1.8)
Requirement already satisfied: h11>=0.8 in /usr/local/lib/python3.11/dist-packages (from uvicorn) (0.14.0)
Downloading uvicorn-0.34.2-py3-none-any.whl (62 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 1.6 MB/s eta 0:00:00
Installing collected packages: uvicorn
Successfully installed uvicorn-0.34.2
```

```
!pip install nest_asyncio
import nest_asyncio
nest_asyncio.apply()
```

```
Requirement already satisfied: nest_asyncio in /usr/local/lib/python3.11/dist-packages (1.6.0)
```

```
# ## 9. Deployment Setup

# %%
from fastapi import FastAPI
from pydantic import BaseModel
import uvicorn
from typing import Optional

app = FastAPI()

class Query(BaseModel):
    text: str
    client_id: Optional[str] = None
    client_context: Optional[str] = None

# Load base model and vector store for deployment
@app.on_event("startup")
async def startup_event():
    global qa_chain, base_model

    # Load base model
    config = PeftConfig.from_pretrained(peft_model_id)
    base_model = AutoModelForSeq2SeqLM.from_pretrained(
        config.base_model_name_or_path,
        # quantization_config=bnb_config,  # Remove quantization_config for CPU
        device_map="auto"
    )
    model = PeftModel.from_pretrained(base_model, peft_model_id)
    model.eval()

    # Create pipeline
    pipe = pipeline(
        "text2text-generation",
        model=model,
        tokenizer=tokenizer,
        # device=DEVICE,  # Remove or comment out 'device' for CPU
        max_length=MAX_TARGET_LENGTH
```

```
    )

    # Create LangChain components
    llm = HuggingFacePipeline(pipeline=pipe)
    vectorstore = FAISS.load_local("healthcare_faiss_index", embeddings, allow_dangerous_deserialization=True)  # allow_dangerous_deserializ
    retriever = vectorstore.as_retriever(search_kwargs={"k": 3})
    global qa_chain
    qa_chain = RetrievalQA.from_chain_type(
        llm=llm,
        chain_type="stuff",
        retriever=retriever,
        return_source_documents=True
    )

@app.post("/ask")
async def ask_question(query: Query):
    try:
        # Load client-specific model if provided
        if query.client_id:
            client_model_path = f"client-models/{query.client_id}"
            model = load_client_model(base_model, client_model_path)

            # Update pipeline with client-specific model
            pipe = pipeline(
                "text2text-generation",
                model=model,
                tokenizer=tokenizer,
                # device=DEVICE,  # Remove or comment out 'device' for CPU
                max_length=MAX_TARGET_LENGTH
            )
            llm = HuggingFacePipeline(pipeline=pipe)
            qa_chain.llm_chain.llm = llm

        # Add client context if provided
        question = query.text
        if query.client_context:
            question = f"Client context: {query.client_context}\n\nQuestion: {question}"

        # Get response
        result = qa_chain(question)

        return {
            "response": result["result"],
            "sources": [doc.metadata["source"] for doc in result["source_documents"]]
        }
    except Exception as e:
        return {"error": str(e)}

# Uncomment to run locally
#if __name__ == "__main__":  # Correct indentation
#    uvicorn.run(app, host="0.0.0.0", port="8000")
```

```
<ipython-input-21-d08c7d22c461>:17: DeprecationWarning:
        on_event is deprecated, use lifespan event handlers instead.

        Read more about it in the
        [FastAPI docs for Lifespan Events](https://fastapi.tiangolo.com/advanced/events/).

   @app.on_event("startup")
```

```
def format_prompt_with_examples(query, similar_examples):
    prompt = "Answer based on these examples:\n"
    for ex in similar_examples:
        prompt += f"Q: {ex['input_text']}\nA: {ex['output_text']}\n\n"
    prompt += f"Q: {query}\nA:"
    return prompt
```

```
!pip install rank_bm25
```

```
Collecting rank_bm25
  Downloading rank_bm25-0.2.2-py3-none-any.whl.metadata (3.2 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from rank_bm25) (2.0.2)
Downloading rank_bm25-0.2.2-py3-none-any.whl (8.6 kB)
Installing collected packages: rank_bm25
Successfully installed rank_bm25-0.2.2
```

```
from langchain.retrievers import BM25Retriever, EnsembleRetriever

bm25_retriever = BM25Retriever.from_documents(docs)
ensemble_retriever = EnsembleRetriever(
    retrievers=[vectorstore.as_retriever(), bm25_retriever],
    weights=[0.7, 0.3]
)
```

```
from huggingface_hub import notebook_login
notebook_login()
peft_model.push_to_hub("your-username/healthcare-assistant-lora")
```

```
Token lohith not found in /root/.cache/huggingface/stored_tokens

README.md: 100%                                    5.17k/5.17k [00:00<00:00, 329kB/s]

adapter_model.safetensors: 100%                    7.10M/7.10M [00:00<00:00, 5.59MB/s]

CommitInfo(commit_url='https://huggingface.co/your-username/healthcare-assistant-lora/commit/b7f84495c8d0f1d127945433d59aedf6620a33cb',
commit_message='Upload model', commit_description='', oid='b7f84495c8d0f1d127945433d59aedf6620a33cb', pr_url=None,
repo_url=RepoUrl('https://huggingface.co/your-username/healthcare-assistant-lora', endpoint='https://huggingface.co',
```

```
!pip install gradio
```

Show hidden output

```
import gradio as gr

def respond(message, history):
    result = healthcare_assistant(message)
    return result["response"]

gr.ChatInterface(respond).launch()
```

```
/usr/local/lib/python3.11/dist-packages/gradio/chat_interface.py:338: UserWarning: The 'tuples' format for chatbot messages is deprecate
  self.chatbot = Chatbot(
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://b6c6c52f8d967530c4.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working dir
```