Support Chatbot using Custom Knowledge Base with LangChain and Open LLM

Can you build a chatbot that can answer questions about your product or service? What if you could use your existing knowledge base to train the chatbot? In this tutorial, we'll build a chatbot that can answer questions about Skyscanner's services . We'll use the company's FAQ section to extract the questions and answers. Then, we'll use the LangChain library to train a chatbot capable of answering questions in real-time on a single GPU (T4). We'll use free language model and embeddings.

> In this part, we will be using Jupyter Notebook to run the code. If you prefer to follow along, you can find the notebook on GitHub: GitHub Repository

Setup

Let's start by installing the required dependencies:

```
!pip install -Uqqq pip --progress-bar off
!pip install -qqq langchain==0.0.228 --progress-bar off
!pip install -qqq chromadb==0.3.26 --progress-bar off
!pip install -qqq sentence-transformers==2.2.2 --progress-bar off
!pip install -qqq auto-gptq==0.2.2 --progress-bar off
!pip install -qqq einops==0.6.1 --progress-bar off
!pip install -qqq unstructured==0.8.0 --progress-bar off
!pip install -qqq transformers==4.30.2 --progress-bar off
!pip install -qqq torch==2.0.1 --progress-bar off
```

Here's a list of the imports we'll be using:

```
from pathlib import Path

import torch
```

```
from auto_gptq import AutoGPTQForCausalLM
from langchain.chains import ConversationalRetrievalChain
from langchain.chains.question_answering import load_qa_chain
from langchain.document_loaders import DirectoryLoader
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.llms import HuggingFacePipeline
from langchain.memory import ConversationBufferMemory
from langchain.prompts import PromptTemplate
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import Chroma
from transformers import AutoTokenizer, GenerationConfig, TextStreamer, pipel
```

Data

The chatbot's custom knowledge is sourced from Skyscanner's FAQ[1] section in the help center. We'll extract 12 questions and answers, creating separate text files for each. Let's create a helper function to write the files:

```
questions_dir = Path("skyscanner")
questions_dir.mkdir(exist_ok=True, parents=True)


def write_file(question, answer, file_path):
    text = f"""
Q: {question}
A: {answer}
""".strip()
    with Path(questions_dir / file_path).open("w") as text_file:
        text_file.write(text)
```

The format we're going to use for each question and answer is the following:

```
Q: Sample question?
A: Sample answer.
```

Time to write the files:

```
write_file(
    question="How do I search for flights on Skyscanner?",
    answer="""
```

```
Skyscanner helps you find the best options for flights on a specific date, or

If you're looking for inspiration for your next trip, why not try our everywh
""".strip(),
    file_path="question_1.txt",
)

write_file(
    question="What are mash-ups?",
    answer="""
These are routes where you fly with different airlines, because it`s cheaper

If you wanted to fly London to New York, we might find it`s cheaper to fly ou

Another kind of mash-up is what we call a "self-transfer" or a "non-protected

If you wanted to fly London to Sydney, we might find it`s cheaper to fly Lond

Pretty simple, right?

However, what`s really important to bear in mind is that mash-ups are NOT cod
""".strip(),
    file_path="question_2.txt",
)

write_file(
    question="Why have I been blocked from accessing the Skyscanner website?"
    answer="""
Skyscanner's websites are scraped by bots many millions of times a day which

Occasionally, this may mean that a genuine user may be wrongly flagged as a b

You're using a VPN which we have had to block due to excessive bot traffic in
You're using our website at super speed which manages to beat our rate limits
You have a plug-in on your browser which could be interfering with how our we
You're using an automated browser
If you've been blocked during normal use, please send us your IP address (thi
""".strip(),
    file_path="question_3.txt",
)

write_file(
    question="Where is my booking confirmation?",
    answer="""
You should get a booking confirmation by email from the company you bought yo

If you still can't find it, try getting in touch with the company you bought

To find out who you need to contact, check the company name next to the charg
""".strip(),
    file_path="question_4.txt",
)
```

```
write_file(
    question="How do I change or cancel my booking?",
    answer="""
For all questions about changes, cancellations, and refunds - as well as all

You'll find 1000s of travel agencies, airlines, hotels and car rental compan:

If you bought from one of these partners, you'll need to contact them as they
""".strip(),
    file_path="question_5.txt",
)

write_file(
    question="I booked the wrong dates / times",
    answer="""
If you have found that you have booked the wrong dates or times, please conta

The search box below can help you find the contact details for the travel pro

You can search flexible or specific dates on Skyscanner to find your preferre
""".strip(),
    file_path="question_6.txt",
)

write_file(
    question="I entered the wrong email address",
    answer="""
Please contact the airline or travel agent you booked with as Skyscanner does

If you can't remember who you booked with, you can check your credit card sta

The search box below can help you find the contact details for the travel pro
""".strip(),
    file_path="question_7.txt",
)

write_file(
    question="Luggage",
    answer="""
Depending on the flight provider, the rules, conditions and prices for luggag
It's always a good idea to check with the airline or travel agent directly (a
""".strip(),
    file_path="question_8.txt",
)

write_file(
    question="Changes, cancellation and refunds",
    answer="""
For changes, cancellations or refunds, we recommend that you contact the trav

As a travel search engine, Skyscanner doesn't take your booking or payment ou
```

```
""".strip(),
    file_path="question_9.txt",
)

write_file(
    question="Why does the price sometimes change when I am redirected to a t
    answer="""
Flight prices and availability change constantly, so we make sure the data is

We make every effort to ensure the information you see on Skyscanner is accur
""".strip(),
    file_path="question_10.txt",
)

write_file(
    question="Why is Skyscanner free?",
    answer="""
Does Skyscanner charge commission?

Nope. Skyscanner is always free to search, and we never charge you any hidden

Want to know how do we do it?

Well, we search through thousands of sites to find you the best deals for fli

See a price you like? We'll connect you to that airline or travel company so

And that's all there is to it!
""".strip(),
    file_path="question_11.txt",
)

write_file(
    question="Are my details safe?",
    answer="""
We take your privacy and safety online very seriously. We'll never sell, shar
""".strip(),
    file_path="question_12.txt",
)
```

Model

I selected the model for our project based on the rankings provided by LMSYS Org[2]. My criteria included strong performance, availability on the HuggingFace Hub, and the ability to perform real-time inference on a single T4 GPU. Our chosen model is *Nous-*

Hermes-13b[3] by Nous Research, trained on GPT-4 synthetic data. To improve the inference speed, we'll utilize a quantized model using the AutoGPTQ library.

Luckily, the model has a quantized version available on the HuggingFace Hub, so we can use the AutoGPTQ library to load it. The quantized model[4] we'll use is 4-bit, and it is provided by *TheBloke*. Let's load it:

```
DEVICE = "cuda:0" if torch.cuda.is_available() else "cpu"

model_name_or_path = "TheBloke/Nous-Hermes-13B-GPTQ"
model_basename = "nous-hermes-13b-GPTQ-4bit-128g.no-act.order"

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

model = AutoGPTQForCausalLM.from_quantized(
    model_name_or_path,
    model_basename=model_basename,
    use_safetensors=True,
    trust_remote_code=True,
    device=DEVICE,
)

generation_config = GenerationConfig.from_pretrained(model_name_or_path)
```

Note that we point to the exact file we want to load, and we also specify the `model_basename` parameter. We also load the tokenizer and the generation config.

To test the model, we need to format the prompt according to the standard format used by the model, which is similar to LLaMa models:

```
question = (
    "Which programming language is more suitable for a beginner: Python or Ja
)
prompt = f"""
### Instruction: {question}
### Response:
""".strip()
```

Let's run the prompt throught the tokenizer and the model:

```
%%time
input_ids = tokenizer(prompt, return_tensors="pt").input_ids.to(DEVICE)
with torch.inference_mode():
    output = model.generate(inputs=input_ids, temperature=0.7, max_new_tokens
```

```
CPU times: user 3.59 s, sys: 1.1 s, total: 4.69 s
Wall time: 12.4 s
```

This took about 12 seconds on a single T4 GPU. Let's see what the model generated:

```
print(tokenizer.decode(output[0]))
```

```
<s> ### Instruction: Which programming language is more suitable for a
beginner: Python or JavaScript?
### Response:Python is generally considered more suitable for beginners due t
its readability and simplicity compared to JavaScript.</s>
```

Note that the model generated a response that is relevant to the question. It also generated a `<s>` token at the beginning of the response, and a `</s>` token at the end. These tokens are used by the model to identify the beginning and the end of the response. We'll need to remove them before returning the response to the user.

Finally, let's have a look at the model's generation config to see what parameters it uses:

```
generation_config
```

```
GenerationConfig {
  "_from_model_config": true,
  "bos_token_id": 1,
  "eos_token_id": 2,
  "pad_token_id": 0,
  "transformers_version": "4.30.2"
}
```

Build a Pipeline

LangChain simplifies the usage of HuggingFace LLMs by providing the `HuggingFacePipeline` class. In this case, we will create a pipeline for text generation using our specific model and tokenizer. Additionally, we will apply a text streamer to stream the responses back to the user in real-time. It will also remove the `<s>` and `</s>` tokens, and the prompt from the response:

```
streamer = TextStreamer(
    tokenizer, skip_prompt=True, skip_special_tokens=True, use_multiprocessir
)

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,
    streamer=streamer,
    batch_size=1,
)

llm = HuggingFacePipeline(pipeline=pipe)
```

Let's pass our prompt through the pipeline to test it out:

```
response = llm(prompt)
```

```
Python is generally considered to be more suitable for beginners due to its
readability and simplicity compared to JavaScript.
```

Great! The response is the same as the one we got when we ran the model directly, but it is cleaned up and ready to be returned to the user.

Embed Documents

To align with our preference for free and open-source models, we will utilize the `E5-base-v2` [5] embeddings, which were initially presented in the paper titled "Text Embeddings by Weakly-Supervised Contrastive Pre-training"[6]. The selection of this embedding model was based on its performance on the Massive Text Embedding Benchmark (MTEB) Leaderboard[7]. The model is available on the HuggingFace Hub, so we can load it using the `HuggingFaceEmbeddings` class:

```
embeddings = HuggingFaceEmbeddings(
    model_name="embaas/sentence-transformers-multilingual-e5-base",
    model_kwargs={"device": DEVICE},
)
```

We can utilize the `DirectoryLoader` in LangChain to load the text files as LangChain documents. This will allow us to conveniently work with the content of the files:

```
loader = DirectoryLoader("./skyscanner/", glob="**/*txt")
documents = loader.load()
len(documents)
```

```
12
```

Due to the context limit of our model (2048 tokens) and the length of the documents, we will need to split the documents into smaller chunks. We can use the `CharacterTextSplitter` to split the documents into chunks of 512 characters with no overlap:

```
text_splitter = CharacterTextSplitter(chunk_size=512, chunk_overlap=0)
texts = text_splitter.split_documents(documents)
texts[4]
```

```
Document(
    page_content='Q: Changes, cancellation and refunds A: For changes, cancel
    metadata={'source': 'skyscanner/question_9.txt'}
)
```

To create (and store) the embeddings we'll use the Chroma. We can create a database using the `Chroma.from_documents` method:

```
db = Chroma.from_documents(texts, embeddings)
db.similarity_search("flight search")
```

```
[
    Document(
        page_content="Q: How do I search for flights on Skyscanner? A: Skysca
        metadata={'source': 'skyscanner/question_1.txt'}
    ),
    Document(
        page_content="You're using a VPN which we have had to block due to ex
        metadata={'source': 'skyscanner/question_3.txt'}
    ),
    Document(
        page_content='Q: I booked the wrong dates / times A: If you have four
        metadata={'source': 'skyscanner/question_6.txt'}
    ),
    Document(
        page_content='You can search flexible or specific dates on Skyscanner
        metadata={'source': 'skyscanner/question_6.txt'}
    )
]
```

With the smart agent (LLM) and the ability to retrieve relevant information (similarity search from the database), you have the necessary prerequisites to build your chatbot.

Conversational Chain

To bring everything together, we'll use a chain from LangChain. Let's begin by defining the prompt:

```
template = """
### Instruction: You're a travelling support agent that is talking to a custo

Use only the chat history and the following information
{context}
to answer in a helpful manner to the question. If you don't know the answer -
say that you don't know. Keep your replies short, compassionate and informati

{chat_history}
### Input: {question}
### Response:
""".strip()

prompt = PromptTemplate(
    input_variables=["context", "question", "chat_history"], template=templat
)
```

Our prompt has 3 variables:

- `context` - the relevant information from the retrieved documents (filled by the chain)
- `chat_history` - the chat history thus far (filled by the memory)
- `question` - the question asked by the user

As for the memory - we'll use the `ConversationBufferMemory` class from LangChain. It will store the chat history:

```
memory = ConversationBufferMemory(
    memory_key="chat_history",
    human_prefix="### Input",
    ai_prefix="### Response",
    output_key="answer",
    return_messages=True,
)
```

Note that we've also defined the `human_prefix` and `ai_prefix` - these will be used to format the chat history in the prompt.

Let's create the chain using our prompt:

```
chain = ConversationalRetrievalChain.from_llm(
    llm,
```

```
    chain_type="stuff",
    retriever=db.as_retriever(),
    memory=memory,
    combine_docs_chain_kwargs={"prompt": prompt},
    return_source_documents=True,
    verbose=True,
)
```

The chain combines the LLM, the retriever and the memory. It also uses the `prompt` to format the output. Finally, we've set `return_source_documents=True` to return the source documents from the database.

Let's try it out:

```
question = "How flight search works?"
answer = chain(question)
```

Chain output

```
Skyscanner helps you find the best options for flights on a specific date, or
on any day in a given month or even year. Our search algorithm scans hundreds
airlines and travel agents to bring you the most comprehensive range of flight
including direct and connecting flights, stopovers, and layovers.  For tips o
to search, please visit our Search Tips page. As a travel search engine,
Skyscanner doesn't take your booking or payment ourselves. Instead, we pass y
to your chosen airline or travel agent where you make your booking directly.
therefore don't have access or visibility to any of your booking information.
```

Great! Our agent has answered the question using the custom knowledge. Let's have a look at the answer structure:

```
answer.keys()
```

```
dict_keys(['question', 'chat_history', 'answer', 'source_documents'])
```

The answer contains the question, the chat history, the answer and the source documents. Let's try to ask another question:

```
question = "I bought flight tickets, but I can't find any confirmation. Where
response = chain(question)
```

Chain output

```
You should receive an email confirmation from the airline or travel agent you
booked with. Sometimes this email might end up in your junk/spam folder. Try
searching your inbox and spam folder first before reaching out to the airline
agent for assistance.
```

The answer looks good! But, the chain is doing something quite strange. It rephrases the question (see the detailed output) and then answers it (haven't found a way to disable this in the current version of LangChain). This is done to reduce the amount of information that the agent needs to remember, but makes our agent slower and it's not always desirable. Next, we'll remove this rephrasing step.

QA Chain with Memory

We'll recreate our chain using the `load_qa_chain` :

```
memory = ConversationBufferMemory(
    memory_key="chat_history",
    human_prefix="### Input",
    ai_prefix="### Response",
    input_key="question",
    output_key="output_text",
    return_messages=False,
)

chain = load_qa_chain(
    llm, chain_type="stuff", prompt=prompt, memory=memory, verbose=True
)
```

This type of chain doesn't rephrase the question and doesn't have a retriever to search for the documents. We'll have to do it ourselves:

```
question = "How flight search works?"
docs = db.similarity_search(question)
answer = chain.run({"input_documents": docs, "question": question})
```

```
Skyscanner helps you find the best options for flights on a specific date, or
on any day in a given month or even year. Our search algorithm scans hundreds
airlines and travel agents to bring you the most comprehensive range of fligh
including direct and connecting flights, stopovers, and layovers.
For tips on how best to search, please visit our Search Tips page. As a trave
Skyscanner doesn't take your booking or payment ourselves. Instead, we pass y
to your chosen airline or travel agent where you make your booking directly.
therefore don't have access or visibility to any of your booking information.
```

Good, we got the same response! While it might be a bit more work, it's much more flexible and allows us to use the same chain as we want. Let's try another question:

```
question = "I entered wrong email address during my flight booking. What shou
docs = db.similarity_search(question)
answer = chain.run({"input_documents": docs, "question": question})
```

```
Please contact the airline or travel agent you booked with as Skyscanner does
not have access to bookings made with airlines or travel agents. If you can't
remember who you booked with, you can check your credit card statement for a
name. The search box below can help you find the contact details for the trav
provider you booked with.
```

No reprhasing and the answer is still good! Let's wrap everything into an easy to use package.

Support Chatbot

Let's create a class that makes it easy to use our chatbot:

```
DEFAULT_TEMPLATE = """
```

```
### Instruction: You're a travelling support agent that is talking to a custo

Use only the chat history and the following information
{context}
to answer in a helpful manner to the question. If you don't know the answer -
say that you don't know. Keep your replies short, compassionate and informati

{chat_history}
### Input: {question}
### Response:
""".strip()


class Chatbot:
    def __init__(
        self,
        text_pipeline: HuggingFacePipeline,
        embeddings: HuggingFaceEmbeddings,
        documents_dir: Path,
        prompt_template: str = DEFAULT_TEMPLATE,
        verbose: bool = False,
    ):
        prompt = PromptTemplate(
            input_variables=["context", "question", "chat_history"],
            template=prompt_template,
        )
        self.chain = self._create_chain(text_pipeline, prompt, verbose)
        self.db = self._embed_data(documents_dir, embeddings)

    def _create_chain(
        self,
        text_pipeline: HuggingFacePipeline,
        prompt: PromptTemplate,
        verbose: bool = False,
    ):
        memory = ConversationBufferMemory(
            memory_key="chat_history",
            human_prefix="### Input",
            ai_prefix="### Response",
            input_key="question",
            output_key="output_text",
            return_messages=False,
        )

        return load_qa_chain(
            text_pipeline,
            chain_type="stuff",
            prompt=prompt,
            memory=memory,
            verbose=verbose,
        )
```

```
    def _embed_data(
        self, documents_dir: Path, embeddings: HuggingFaceEmbeddings
    ) -> Chroma:
        loader = DirectoryLoader(documents_dir, glob="**/*txt")
        documents = loader.load()
        text_splitter = CharacterTextSplitter(chunk_size=512, chunk_overlap=(
        texts = text_splitter.split_documents(documents)
        return Chroma.from_documents(texts, embeddings)

    def __call__(self, user_input: str) -> str:
        docs = self.db.similarity_search(user_input)
        return self.chain.run({"input_documents": docs, "question": user_inpu
```

Our chatbot class takes a text pipeline, embeddings, and a directory with documents. The two main tasks are to create a chain and a database. We also override the `__call__` method to make calling our chatbot as easy as calling a function.

Let's create an instance of our class:

```
chatbot = Chatbot(llm, embeddings, "./skyscanner/")
```

To create the chat, we'll use the `while` loop and ask the user for input. If the user enters *bye* or *goodbye*, we'll break the loop:

```
import warnings

warnings.filterwarnings("ignore", category=UserWarning)

while True:
    user_input = input("You: ")
    if user_input.lower() in ["bye", "goodbye"]:
        break
    answer = chatbot(user_input)
    print()
```

You: How flight search works?

Assistant: Skyscanner helps you find the best options for flights on a specific date, or on any day in a given month or even year. Our search algorithm scans hundreds of airlines and travel agents to bring you the most comprehensive range of flight options, including direct and connecting flights, stopovers, and layovers.

You: I bought flight tickets, but I can't find any confirmation. Where is it?

Assistant: It sounds like you purchased your flight tickets from another company or airline, rather than through Skyscanner. To locate your booking confirmation, please contact the company you purchased your ticket from directly. They will be able to provide you with more information regarding your booking and confirmation details.

You: I entered wrong email address during my flight booking. What should I do?

Assistant: Unfortunately, since Skyscanner isn't involved in the actual booking process, we are unable to assist with changing email addresses. However, you may reach out to the airline or travel agent you booked with to see if they can update your email address for you. Alternatively, you could try searching for your booking using the search bar provided on our website to see if you can locate your booking yourself.

You: What are the advantages of mash-ups?

Assistant: Mash-ups offer several benefits over traditional round-trip flights. Here are some key advantages:

1. Cost savings: By combining two separate one-way flights, you can often save money compared to buying a single round-trip ticket.
2. Flexibility: With a mash-up, you have more control over your travel plans. If you need to change your return date or time, you can simply adjust one leg of your trip without having to rebook everything else.
3. More options: By flying with multiple airlines, you have access to a wider range of destinations and departure times. This means you can choose the flight that best fits your schedule and preferences.

References

1. Skyscanner help center ↩
2. Chatbot Arena Leaderboard by LMSYS Org ↩
3. Nous Hermes 13B (Original model) ↩
4. Nous-Hermes-13B GPTQ ↩
5. E5-base-v2 embeddings ↩

6. Text Embeddings by Weakly-Supervised Contrastive Pre-training ↩️
7. Embeddings Leaderboard ↩️
