Question-answering with LangChain

In this example we create a large-language-model (LLM) powered question answering web endpoint and CLI. Only a single document is used as the knowledge-base of the application, the 2022 USA State of the Union address by President Joe Biden. However, this same application structure could be extended to do question-answering over all State of the Union speeches, or other large text corpuses.

It's the LangChain library that makes this all so easy. This demo is only around 100 lines of code!

Defining dependencies

The example uses three PyPi packages to make scraping easy, and three to build and run the question-answering functionality. These are installed into a Debian Slim base image using the `pip_install` function.

Because OpenAI's API is used, we also specify the `openai-secret` Modal Secret, which contains an OpenAI API key.

A `docsearch` global variable is also declared to facilitate caching a slow operation in the code below.

```
from pathlib import Path

from modal import App, Image, Secret, web_endpoint

image = Image.debian_slim().pip_install(
```

```python
    # scraping pkgs
    "beautifulsoup4~=4.11.1",
    "httpx~=0.23.3",
    "lxml~=4.9.2",
    # langchain pkgs
    "faiss-cpu~=1.7.3",
    "langchain~=0.0.138",
    "openai~=0.27.4",
    "tiktoken==0.3.0",
)
app = App(
    name="example-langchain-qanda",
    image=image,
    secrets=[Secret.from_name("openai-secret")],
)
docsearch = None  # embedding index that's relatively expensive to compute, so caching wit
```

Scraping the speech from whitehouse.gov

It's super easy to scrape the transcipt of Biden's speech using `httpx` and `BeautifulSoup`. This speech is just one document and it's relatively short, but it's enough to demonstrate the question-answering capability of the LLM chain.

```python
def scrape_state_of_the_union() -> str:
    import httpx
    from bs4 import BeautifulSoup

    url = "https://www.whitehouse.gov/state-of-the-union-2022/"

    # fetch article; simulate desktop browser
    headers = {
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9
    }
    response = httpx.get(url, headers=headers)
    soup = BeautifulSoup(response.text, "lxml")

    # get all text paragraphs & construct string of article text
    speech_text = ""
    speech_section = soup.find_all(
        "div", {"class": "sotu-annotations__content"}
    )
    if speech_section:
        paragraph_tags = speech_section[0].find_all("p")
        speech_text = "".join([p.get_text() for p in paragraph_tags])

    return speech_text.replace("\t", "")
```

Constructing the Q&A chain

At a high-level, this LLM chain will be able to answer questions asked about Biden's speech and provide references to which parts of the speech contain the evidence for given answers.

The chain combines a text-embedding index over parts of Biden's speech with OpenAI's GPT-3 LLM. The index is used to select the most likely relevant parts of the speech given the question, and these are used to build a specialized prompt for the OpenAI language model.

For more information on this, see LangChain's "Question Answering" notebook.

```python
def retrieve_sources(sources_refs: str, texts: list[str]) -> list[str]:
    """
    Map back from the references given by the LLM's output to the original text parts.
    """
    clean_indices = [
        r.replace("-pl", "").strip() for r in sources_refs.split(",")
    ]
    numeric_indices = (int(r) if r.isnumeric() else None for r in clean_indices)
    return [
        texts[i] if i is not None else "INVALID SOURCE" for i in numeric_indices
    ]


def qanda_langchain(query: str) -> tuple[str, list[str]]:
    from langchain.chains.qa_with_sources import load_qa_with_sources_chain
    from langchain.embeddings.openai import OpenAIEmbeddings
    from langchain.llms import OpenAI
    from langchain.text_splitter import CharacterTextSplitter
    from langchain.vectorstores.faiss import FAISS

    # Support caching speech text on disk.
    speech_file_path = Path("state-of-the-union.txt")

    if speech_file_path.exists():
        state_of_the_union = speech_file_path.read_text()
    else:
        print("scraping the 2022 State of the Union speech")
        state_of_the_union = scrape_state_of_the_union()
        speech_file_path.write_text(state_of_the_union)

    # We cannot send the entire speech to the model because OpenAI's model
    # has a maximum limit on input tokens. So we split up the speech
    # into smaller chunks.
    text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
    print("splitting speech into text chunks")
    texts = text_splitter.split_text(state_of_the_union)

    # Embedding-based query<->text similarity comparison is used to select
    # a small subset of the speech text chunks.
```

```python
    # Generating the `docsearch` index is too slow to re-run on every request,
    # so we do rudimentary caching using a global variable.
    global docsearch

    if not docsearch:
        # New OpenAI accounts have a very low rate-limit for their first 48 hrs.
        # It's too low to embed even just this single Biden speech.
        # The `chunk_size` parameter is set to a low number, and internally LangChain
        # will retry the embedding requests, which should be enough to handle the rate-lim
        #
        # Ref: https://platform.openai.com/docs/guides/rate-limits/overview.
        print("generating docsearch indexer")
        docsearch = FAISS.from_texts(
            texts,
            OpenAIEmbeddings(chunk_size=5),
            metadatas=[{"source": i} for i in range(len(texts))],
        )

    print("selecting text parts by similarity to query")
    docs = docsearch.similarity_search(query)

    chain = load_qa_with_sources_chain(
        OpenAI(model_name="gpt-3.5-turbo-instruct", temperature=0),
        chain_type="stuff",
    )
    print("running query against Q&A chain.\n")
    result = chain(
        {"input_documents": docs, "question": query}, return_only_outputs=True
    )
    output: str = result["output_text"]
    parts = output.split("SOURCES: ")
    if len(parts) == 2:
        answer, sources_refs = parts
        sources = retrieve_sources(sources_refs, texts)
    elif len(parts) == 1:
        answer = parts[0]
        sources = []
    else:
        raise RuntimeError(
            f"Expected to receive an answer with a single 'SOURCES' block, got:\n{output}"
        )
    return answer.strip(), sources
```

Modal Functions

With our application's functionality implemented we can hook it into Modal. As said above, we're implementing a web endpoint, `web`, and a CLI command, `cli`.

```python
@app.function()
@web_endpoint(method="GET")
def web(query: str, show_sources: bool = False):
    answer, sources = qanda_langchain(query)
    if show_sources:
        return {
            "answer": answer,
            "sources": sources,
        }
    else:
        return {
            "answer": answer,
        }


@app.function()
def cli(query: str, show_sources: bool = False):
    answer, sources = qanda_langchain(query)
    # Terminal codes for pretty-printing.
    bold, end = "\033[1m", "\033[0m"

    print(f"🦅 {bold}ANSWER:{end}")
    print(answer)
    if show_sources:
        print(f"🔗 {bold}SOURCES:{end}")
        for text in sources:
            print(text)
            print("----")
```

Test run the CLI

```
modal run potus_speech_qanda.py --query "What did the president say about Justice Breyer"
🦅 ANSWER:
The president thanked Justice Breyer for his service and mentioned his legacy of excellenc
```

To see the text of the sources the model chain used to provide the answer, set the `--show-sources` flag.

```
modal run potus_speech_qanda.py \
    --query "How many oil barrels were released from reserves" \
    --show-sources=True
```

Test run the web endpoint

Modal makes it trivially easy to ship LangChain chains to the web. We can test drive this app's web endpoint by running `modal serve potus_speech_qanda.py` and then hitting the endpoint with `curl`:

```
curl --get \
  --data-urlencode "query=What did the president say about Justice Breyer" \
  https://modal-labs--example-langchain-qanda-web.modal.run
```

```
{
  "answer": "The president thanked Justice Breyer for his service and mentioned his legacy
}
```

About

Status

Changelog

Documentation

Slack Community

Pricing

Examples