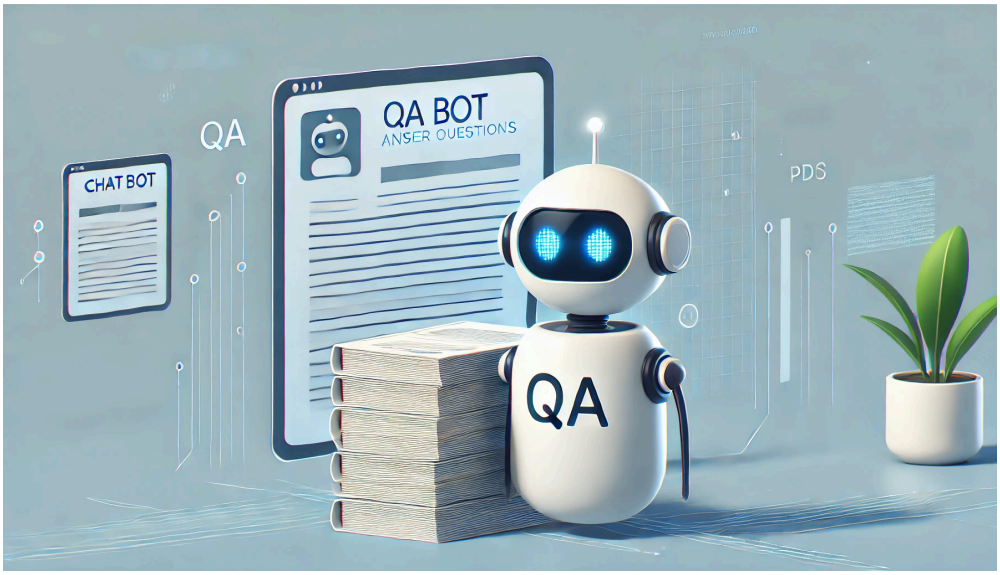
构建一个利用LangChain和LLMs从加载的文档中回答问题的QA机器人

预计所需时间：60分钟

在这个项目中，您将构建一个问答（QA）机器人。这个机器人将利用LangChain和大型语言模型（LLM）根据加载的PDF文档的内容回答问题。为了构建一个完全功能的QA系统，您将结合各种组件，包括文档加载器、文本拆分器、嵌入模型、向量数据库、检索器，以及作为前端界面的Gradio。

想象一下，您被指派创建一个智能助手，能够快速准确地根据公司庞大的PDF文档库响应查询。这可能包括法律文件到技术手册的任何内容。手动搜索这些文档将耗时且效率低下。



来源：DALL-E

在这个项目中，您将构建一个自动化这个过程的QA机器人。通过利用LangChain和LLM，机器人将读取并理解加载的PDF文档的内容，使其能够提供准确的用户查询答案。您将整合工具和技术，从文档加载、文本拆分、嵌入、向量存储到检索，创建一个无缝且用户友好的Gradio界面体验。

学习目标

在本项目结束时，您将能够：

- 结合多个组件，如文档加载器、文本拆分器、嵌入模型和向量数据库，构建一个完全功能的QA机器人
- 利用LangChain和LLMs解决从大型PDF文档中检索和回答问题的问题

设置

设置虚拟环境

让我们创建一个虚拟环境。使用虚拟环境可以让你为不同的项目单独管理依赖，避免包版本之间的冲突。

在你的云IDE的终端中，确保你位于路径 `/home/project`，然后运行以下命令以创建一个Python虚拟环境。

```
pip install virtualenv
virtualenv my_env # create a virtual environment named my_env
source my_env/bin/activate # activate my_env
```

安装必要的库

为了确保脚本的顺利执行，并考虑到这些脚本中的某些功能依赖于外部库，在开始之前安装一些先决库是必不可少的。对于这个项目，您需要的关键库包括 `Gradio` 用于创建用户友好的网页界面，以及 `IBM-watsonx-AI` 用于利用 IBM watsonx API 的高级 LLM 模型。

- **`gradio`** 允许您快速构建交互式网页应用程序，使您的 AI 模型易于用户访问。
- **`ibm-watsonx-ai`** 用于使用 IBM watsonx.ai 的 LLM。
- **`langchain, langchain-ibm, langchain-community`** 用于使用 Langchain 的相关功能。
- **`chromadb`** 用于将 chroma 数据库用作向量数据库。
- **`pypdf`** 用于加载 PDF 文档。

以下是如何安装这些包（从您的终端）：

```
# installing necessary packages in my_env
python3.11 -m pip install \
gradio==4.44.0 \
ibm-watsonx-ai==1.1.2  \
langchain==0.2.11 \
langchain-community==0.2.10 \
langchain-ibm==0.1.11 \
chromadb==0.4.24 \
pypdf==4.3.1 \
pydantic==2.9.1
```

现在，环境已准备好创建应用程序。

构建QA机器人

现在是构建QA机器人的时候了！

让我们开始创建一个新的Python文件来存储你的机器人。点击下面的按钮创建一个新的Python文件，并命名为 `qabot.py`。如果出于某种原因按钮无法使用，可以通过 `文件 --> 新建文本文件` 来创建新文件。确保将文件保存为 `qabot.py`。

Open **qabot.py** in IDE

你将在接下来的部分中填充 `qabot.py` 以构建你的机器人。

导入必要的库

在 `qabot.py` 中，从 `gradio`、`ibm_watsonx.ai`、`langchain_ibm`、`langchain` 和 `langchain_community` 导入以下内容。这些导入的类对于使用正确的凭据初始化模型、分割文本、初始化向量存储、加载PDF、生成问答检索器以及使用Gradio是必要的。

```
from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
from ibm_watsonx_ai.metanames import EmbedTextParamsMetaNames
from ibm_watsonx_ai import Credentials
from langchain_ibm import WatsonxLLM, WatsonxEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_community.document_loaders import PyPDFLoader
from langchain.chains import RetrievalQA
import gradio as gr
# You can use this section to suppress warnings generated by your code:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')
```

初始化 LLM

您现在将通过创建 `WatsonxLLM` 的实例来初始化 LLM，它是 `langchain_ibm` 中的一个类。`WatsonxLLM` 可以使用多个基础模型。在这个特定的例子中，您将使用 Mixtral 8x7B，尽管您也可以使用其他模型，例如 Llama 3.3 70B。有关 watsonx.ai 中可用的基础模型的列表，请参阅 文档。

要初始化 LLM，请将以下内容粘贴到 `qabot.py` 中。请注意，您正在以 0.5 的温度初始化模型，并允许生成最多 256 个标记。

```
## LLM
```

```
    def get_llm():
        model_id = 'mistralai/mixtral-8x7b-instruct-v01'
        parameters = {
            GenParams.MAX_NEW_TOKENS: 256,
            GenParams.TEMPERATURE: 0.5,
        }
        project_id = "skills-network"
        watsonx_llm = WatsonxLLM(
            model_id=model_id,
            url="https://us-south.ml.cloud.ibm.com",
            project_id=project_id,
            params=parameters,
        )
        return watsonx_llm
```

定义 PDF 文档加载器

接下来，您将定义 PDF 文档加载器。要加载 PDF 文档，您将使用 `langchain_community` 库中的 `PyPDFLoader` 类。语法非常简单。首先，您将 PDF 加载器创建为 `PyPDFLoader` 的一个实例。然后，您加载文档并返回加载的文档。要在您的机器人中加入 PDF 加载器，请将以下内容添加到 `qabot.py`：

```
## Document loader
def document_loader(file):
    loader = PyPDFLoader(file.name)
    loaded_document = loader.load()
    return loaded_document
```

定义文本分割器

PDF 文档加载器加载文档，但在使用 `.load()` 方法时不会将其拆分为块。因此，您必须定义一个文档分割器来将文本拆分为块。将以下代码添加到 `qabot.py` 以定义这样的文本分割器。在此示例中，您定义了一个 `RecursiveCharacterTextSplitter`，块大小为 1000，尽管其他分割器或参数值也是可能的。

```
## Text splitter
def text_splitter(data):
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000,
        chunk_overlap=50,
        length_function=len,
    )
    chunks = text_splitter.split_documents(data)
    return chunks
```

定义向量存储

现在您可以将 PDF 加载为文本并将该文本拆分为块，您必须定义一种方法来嵌入并存储这些块在向量数据库中。将以下代码添加到 `qabot.py` 中，以定义一个使用尚未定义的嵌入模型嵌入块并将嵌入存储在 ChromaDB 向量存储中的函数：

```
## Vector db
def vector_database(chunks):
    embedding_model = watsonx_embedding()
    vectordb = Chroma.from_documents(chunks, embedding_model)
    return vectordb
```

定义嵌入模型

上述 `vector_database()` 函数假设存在一个 `watsonx_embedding()` 函数，该函数加载嵌入模型的实例。这个嵌入模型用于将文本块转换为向量表示。以下代码定义了一个 `watsonx_embedding()` 函数，该函数返回 `WatsonxEmbeddings` 的实例，这是来自 `langchain_ibm` 的一个类，用于生成嵌入。在这种情况下，嵌入是使用 IBM 的 Slate 125M 英语嵌入模型生成的。将此代码粘贴到 `qabot.py` 文件中：

```
## Embedding model
def watsonx_embedding():
    embed_params = {
        EmbedTextParamsMetaNames.TRUNCATE_INPUT_TOKENS: 3,
        EmbedTextParamsMetaNames.RETURN_OPTIONS: {"input_text": True},
    }
    watsonx_embedding = WatsonxEmbeddings(
        model_id="ibm/slate-125m-english-rtrvr",
        url="https://us-south.ml.cloud.ibm.com",
        project_id="skills-network",
        params=embed_params,
    )
    return watsonx_embedding
```

Python 并不在乎 `watsonx_embedding()` 的定义顺序在 `vector_database()` 之后。定义的顺序可以反转，这不会改变机器人的基本功能。

定义检索器

现在您的向量存储已定义，您必须定义一个检索器，从中检索文档的片段。在这种情况下，您将定义一个基于向量存储的检索器，通过简单的相似性搜索来检索信息。为此，请将以下行添加到 `qabot.py`：

```
## Retriever
def retriever(file):
    splits = document_loader(file)
    chunks = text_splitter(splits)
    vectordb = vector_database(chunks)
    retriever = vectordb.as_retriever()
    return retriever
```

定义问答链

最后，是时候定义一个问答链了！在这个具体的例子中，您将使用 `RetrievalQA` 来自 `langchain`，这是一个通过检索增强生成（RAG）在数据源上执行自然语言问答的链。将以下代码添加到 `qabot.py` 以定义问答链：

```
## QA Chain
def retriever_qa(file, query):
    llm = get_llm()
    retriever_obj = retriever(file)
    qa = RetrievalQA.from_chain_type(llm=llm,
                                     chain_type="stuff",
                                     retriever=retriever_obj,
                                     return_source_documents=False)
    response = qa.invoke(query)
    return response['result']
```

让我们回顾一下我们机器人的所有元素是如何链接的。请注意，`RetrievalQA` 接受一个 LLM（`get_llm()`）和一个检索器对象（由 `retriever()` 生成的实例）作为参数。然而，检索器是基于向量存储（`vector_database()`），而向量存储又需要一个嵌入模型（`watsonx_embedding()`）和使用文本分割器（`text_splitter()`）生成的文本块。文本分割器又需要原始文本，而这些文本是通过 `PyPDFLoader` 从 PDF 中加载的。这有效地定义了您的 QA 机器人的核心功能！

设置 Gradio 界面

鉴于您已经创建了机器人的核心功能，最后需要定义的是 Gradio 界面。您的 Gradio 界面应包括：

- 文件上传功能（由 Gradio 中的 `File` 类提供）
- 一个可以提问的输入文本框（由 Gradio 中的 `Textbox` 类提供）
- 一个可以回答问题的输出文本框（由 Gradio 中的 `Textbox` 类提供）

将以下代码添加到 `qabot.py` 中以添加 Gradio 界面：

```
# Create Gradio interface
rag_application = gr.Interface(
    fn=retriever_qa,
    allow_flagging="never",
    inputs=[
        gr.File(label="Upload PDF File", file_count="single", file_types=['.pdf'], type="filepath"),  # Drag and drop file upload
        gr.Textbox(label="Input Query", lines=2, placeholder="Type your question here...")
    ],
    outputs=gr.Textbox(label="Output"),
    title="RAG Chatbot",
    description="Upload a PDF document and ask any question. The chatbot will try to answer using the provided document."
)
```

添加代码以启动应用程序

最后，您需要在 `qabot.py` 中添加一行代码，以使用端口 7860 启动您的应用程序：

```
# Launch the app
rag_application.launch(server_name="0.0.0.0", server_port= 7860)
```

在添加上述行后，保存 `qabot.py`。

验证 `qabot.py`

您的 `qabot.py` 现在应该如下所示：

```
from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
from ibm_watsonx_ai.metanames import EmbedTextParamsMetaNames
from ibm_watsonx_ai import Credentials
from langchain_ibm import WatsonxLLM, WatsonxEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_community.document_loaders import PyPDFLoader
from langchain.chains import RetrievalQA
import gradio as gr
# You can use this section to suppress warnings generated by your code:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')
## LLM
def get_llm():
    model_id = 'mistralai/mixtral-8x7b-instruct-v01'
    parameters = {
        GenParams.MAX_NEW_TOKENS: 256,
        GenParams.TEMPERATURE: 0.5,
    }
    project_id = "skills-network"
    watsonx_llm = WatsonxLLM(
        model_id=model_id,
        url="https://us-south.ml.cloud.ibm.com",
        project_id=project_id,
        params=parameters,
    )
    return watsonx_llm
## Document loader
def document_loader(file):
    loader = PyPDFLoader(file.name)
    loaded_document = loader.load()
    return loaded_document
## Text splitter
def text_splitter(data):
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000,
        chunk_overlap=50,
```

```
            length_function=len,
        )
        chunks = text_splitter.split_documents(data)
        return chunks
    ## Vector db
    def vector_database(chunks):
        embedding_model = watsonx_embedding()
        vectordb = Chroma.from_documents(chunks, embedding_model)
        return vectordb
    ## Embedding model
    def watsonx_embedding():
        embed_params = {
            EmbedTextParamsMetaNames.TRUNCATE_INPUT_TOKENS: 3,
            EmbedTextParamsMetaNames.RETURN_OPTIONS: {"input_text": True},
        }
        watsonx_embedding = WatsonxEmbeddings(
            model_id="ibm/slate-125m-english-rtrvr",
            url="https://us-south.ml.cloud.ibm.com",
            project_id="skills-network",
            params=embed_params,
        )
        return watsonx_embedding
    ## Retriever
    def retriever(file):
        splits = document_loader(file)
        chunks = text_splitter(splits)
        vectordb = vector_database(chunks)
        retriever = vectordb.as_retriever()
        return retriever
    ## QA Chain
    def retriever_qa(file, query):
        llm = get_llm()
        retriever_obj = retriever(file)
        qa = RetrievalQA.from_chain_type(llm=llm,
                                         chain_type="stuff",
                                         retriever=retriever_obj,
                                         return_source_documents=False)
        response = qa.invoke(query)
        return response['result']
    # Create Gradio interface
    rag_application = gr.Interface(
        fn=retriever_qa,
        allow_flagging="never",
        inputs=[
            gr.File(label="Upload PDF File", file_count="single", file_types=['.pdf'], type="filepath"),  # Drag and drop file upload
            gr.Textbox(label="Input Query", lines=2, placeholder="Type your question here...")
        ],
        outputs=gr.Textbox(label="Output"),
        title="RAG Chatbot",
        description="Upload a PDF document and ask any question. The chatbot will try to answer using the provided document."
    )
    # Launch the app
    rag_application.launch(server_name="0.0.0.0", server_port= 7860)
```

服务应用程序

要服务应用程序，请将以下内容粘贴到您的 Python 终端中：

```
python3.11 qabot.py
```

如果您无法找到打开的 Python 终端，或者上面的单元格中的按钮无法使用，您可以通过转到 `Terminal --> New Terminal` 启动一个终端。不过，如果您启动了一个新终端，请不要忘记在运行这行代码之前，先激活您在本实验开始时创建的虚拟环境：

```
source my_env/bin/activate # activate my_env
```

启动应用程序

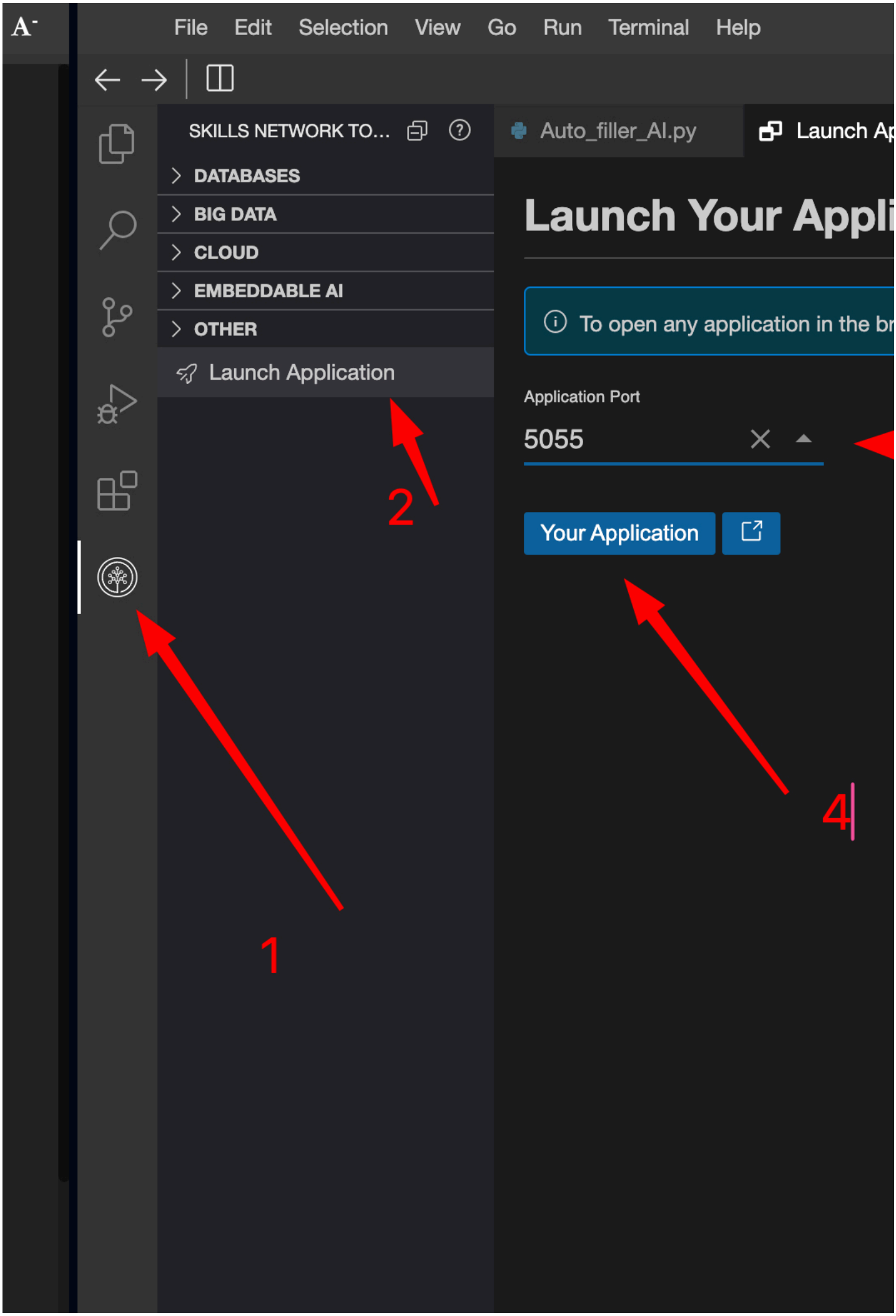
您现在准备好启动服务的应用程序了！要做到这一点，请点击以下按钮：

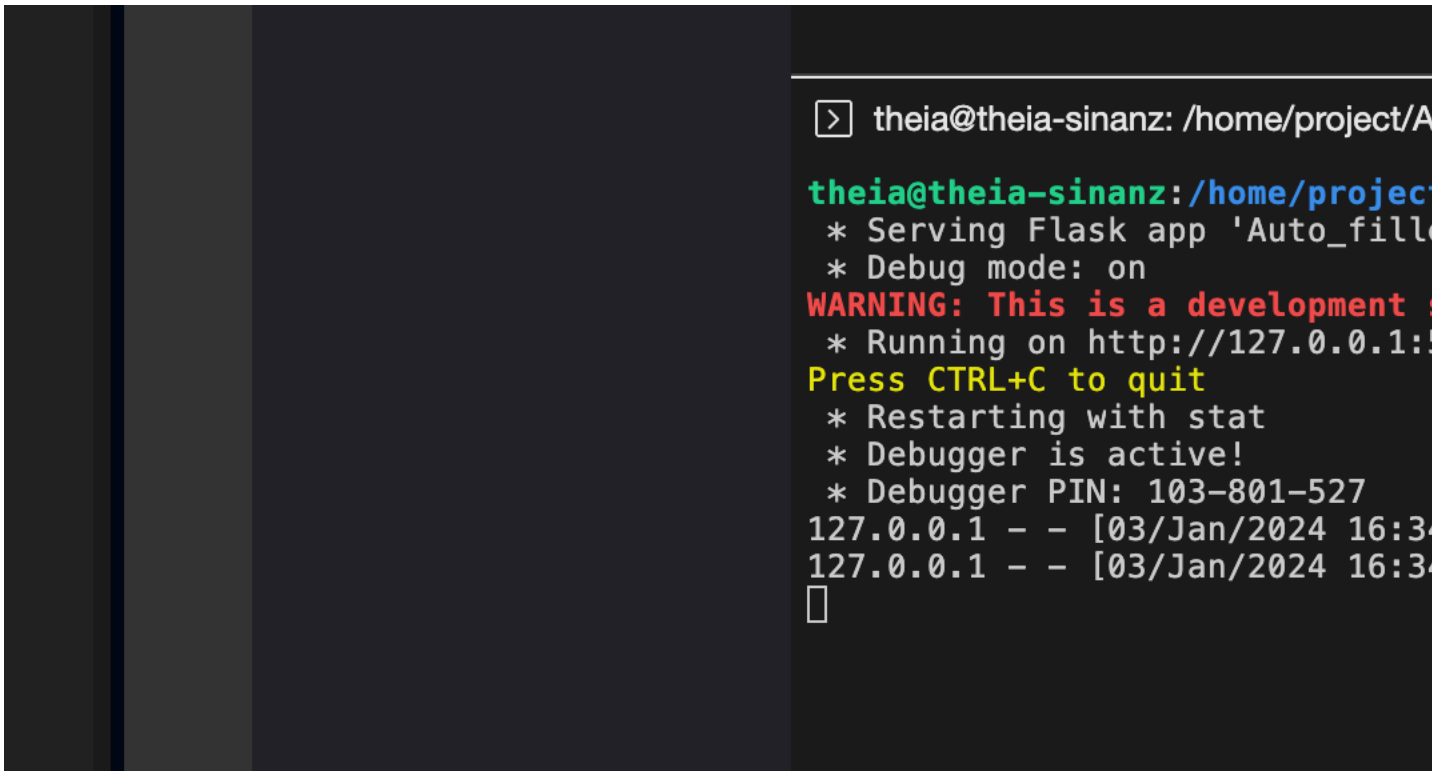
启动应用程序

如果上述按钮无法使用，请按照以下说明操作：

1. 选择 Skills Network 扩展。
2. 点击 **启动应用程序**
3. 输入端口号（在本例中为 7860，这是我们在 `qabot.py` 中设置的服务器端口）
4. 点击 **您的应用程序** 以启动应用程序。

注意：如果使用 **您的应用程序** 无法正常工作，请使用图标 **在新浏览器选项卡中打开**。


File Edit Selection View Go Run Terminal Help
SKILLS NETWORK TO...
DATABASES
BIG DATA
CLOUD
EMBEDDABLE AI
OTHER
Launch Application
Auto_filler_AI.py
Launch Ap
Launch Your Appli
To open any application in the br
Application Port
5055
Your Application
1
2
4



您现在可以通过上传可读取的 PDF 文档并询问其内容来与应用程序互动！

为了获得最佳效果，请确保 PDF 文档不太大。大文档在当前设置下会失败。

如果您完成了对应用程序的实验并想退出，请在终端中按 ctrl+c 并关闭应用程序选项卡。

作者

作者

Kang Wang

Wojciech "Victor" Fulmyk

Kunal Makwana

Ricky Shi

贡献者

Hailey Quach