## NVIDIA AI-AGENT夏季训练营

项目名称：AI-AGENT夏季训练营 — RAG智能对话机器人

报告日期：2024年8月18日

项目负责人：LuoYe (罗烨)

#### 项目概述：

此项目是一个基于Retrieval-augmented Generation (RAG)的文字形式的智能对话机器人，旨在帮助学生在学习英语的过程中提高对于单词的记忆能力。RAG模型结合了检索和生成的优势，能够更准确地理解和生成自然语言，使得对话更加流畅和自然。同时还能能够提供个性化学习建议和易于使用的界面，学生只需输入问题或回答便能自动给出回应，使得学习过程更加便捷和高效。

#### 技术方案与实施步骤

* 模型选择： 详细描述项目采用的技术方案，包括大模型的选择理由、RAG模型的优势分析。

此项目选择了基于 llama-3.1-405b-instruct的技术方案。Llama-3.1-405b-instruct是一个大型语言模型，具有3.1万亿个参数，采用了先进的自然语言处理技术，具备强大的语言理解和生成能力。选择使用这个模型来构建智能对话机器人，主要因为它具有强大的语言理解能力,能够理解和解释自然语言。其次是优秀的生成能力，能得对话更加自然和流畅，提供更好的用户体验。此外Llama-3.1-405b-instruct模型使用了指令微调技术，通过大量的指令和响应对模型进行训练，使得模型能够更好地理解和执行人类的指令。最后是高效的训练和推理技术，使得模型能够在有限的计算资源下进行训练和推理，提高了模型的可用性，使得它适合成为构建智能对话机器人的大模型。

我们还采用了Retrieval-augmented Generation (RAG)模型。RAG模型通过检索，能够提供更准确和可靠的回答，提高对话的质量。同时RAG模型通过检索和融合，能够更好地理解和解释自然语言，提高语言理解能力。

* 数据的构建（必写）： 说明数据构建过程、向量化处理方法及其优势。

数据构建过程包括数据的收集和向量化处理。从txt文件收集数据，去除不相关的信息，使用ai-embed-qa-4模型对数据进行向量化处理，将文本转换为数值向量，便于模型的输入和处理。

ai-embed-qa-4是一种基于深度学习的文本向量化模型，能够将文本转换为高维空间的数值向量。选择ai-embed-qa-4进行向量化处理，能够生成高质量的文字向量，包括词语的语义信息和上下文信息，有利于模型进一步的处理。同时ai-embed-qa-4支持各种预训练模型和自定义模型，可以根据不同的需求进行调整和优化，以获得最佳的向量化效果。通过以上方法，能够将文本数据转换为适合模型输入的数值向量，提高模型的训练效果和性能。

#### 实施步骤：

* 环境搭建： 描述开发环境的搭建过程，包括必要的软件、库的安装与配置。

1 创建Python环境

首先需要安装Miniconda：

大家可以根据自己的网络情况从下面的地址下载：

miniconda官网地址：https://docs.conda.io/en/latest/miniconda.html

清华大学镜像地址： https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/

安装完之后，打开Anaconda Powershell:

在打开的终端中按照下面的步骤执行,配置环境:

创建python 3.8虚拟环境

conda create --name ai\_endpoint python=3.8

进入虚拟环境

conda activate ai\_endpoint

安装nvidia\_ai\_endpoint工具

pip install langchain-nvidia-ai-endpoints

安装Jupyter Lab

pip install jupyterlab

安装langchain\_core

pip install langchain\_core

安装langchain

pip install langchain

安装matplotlib

pip install matplotlib

安装Numpy

pip install numpy

安装faiss, 这里如果没有GPU可以安装CPU版本

pip install faiss-cpu==1.7.2

安装OPENAI库

pip install openai

2.利用Jupyter Lab打开课件执行

在上面打开的终端中, 输入以下命令:

jupyter-lab

* 代码实现（必写）： 列出关键代码的实现步骤，可附上关键代码截图或代码块。

## 第一步, 导入工具包

from langchain\_nvidia\_ai\_endpoints import ChatNVIDIA

from langchain\_core.output\_parsers import StrOutputParser

from langchain\_core.prompts import ChatPromptTemplate

from langchain.schema.runnable import RunnableLambda

from langchain.schema.runnable.passthrough import RunnableAssign

from langchain\_core.runnables import RunnableBranch

from langchain\_core.runnables import RunnablePassthrough

from langchain.chains import ConversationChain

from langchain.memory import ConversationBufferMemory

import os

import base64

import matplotlib.pyplot as plt

import numpy as np

import gradio as gr

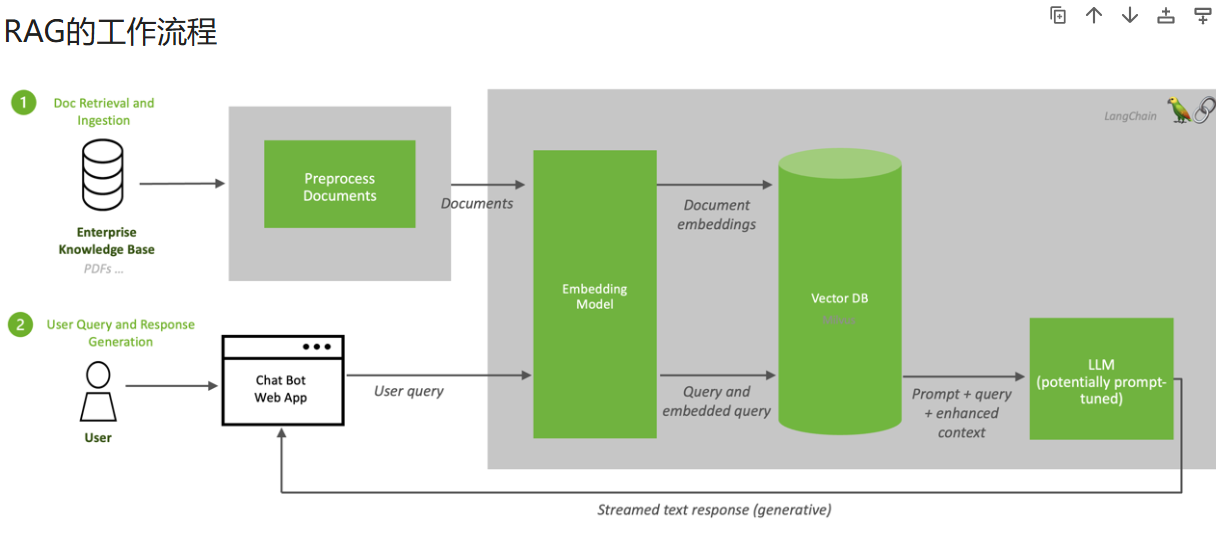
from langchain.memory import ConversationSummaryBufferMemory

### Step 1 - 使用NVIDIA\_API\_KEY

os.environ["NVIDIA\_API\_KEY"] = "yours\_api\_key"

网址：[NVIDIA NIM | llama-3\_1-405b-instruct](https://build.nvidia.com/meta/llama-3_1-405b-instruct?snippet_tab=Python)

### Step 2 - 初始化SLM

llm = ChatNVIDIA(model="ai-phi-3-small-128k-instruct", nvidia\_api\_key= "yours\_api\_key", max\_tokens=512)

### Step 3 - 初始化ai-embed-qa-4向量模型

from langchain\_nvidia\_ai\_endpoints import NVIDIAEmbeddings

embedder = NVIDIAEmbeddings(model="ai-embed-qa-4")

### Step 4 - 获取文本数据集

import os

from tqdm import tqdm

from pathlib import Path

# Here we read in the text data and prepare them into vectorstore

ps = os.listdir("./zh\_data/")

data = []

sources = []

for p in ps:

if p.endswith('.txt'):

path2file="./zh\_data/"+p

with open(path2file,encoding="utf-8") as f:

lines=f.readlines()

for line in lines:

if len(line)>=1:

data.append(line)

sources.append(path2file)

### Step 5 - 进行一些基本的清理并删除空行

documents=[d for d in data if d != '\n']

len(data), len(documents), data[0]

**Step 6a - 将文档处理到 faiss vectorstore 并将其保存到磁盘**

# Here we create a vector store from the documents and save it to disk.

from operator import itemgetter

from langchain.vectorstores import FAISS

from langchain\_core.output\_parsers import StrOutputParser

from langchain\_core.prompts import ChatPromptTemplate

from langchain\_core.runnables import RunnablePassthrough

from langchain.text\_splitter import CharacterTextSplitter

from langchain\_nvidia\_ai\_endpoints import ChatNVIDIA

import faiss

# 只需要执行一次，后面可以重读已经保存的向量存储

text\_splitter = CharacterTextSplitter(chunk\_size=400, separator=" ")

docs = []

metadatas = []

for i, d in enumerate(documents):

splits = text\_splitter.split\_text(d)

#print(len(splits))

docs.extend(splits)

metadatas.extend([{"source": sources[i]}] \* len(splits))

store = FAISS.from\_texts(docs, embedder , metadatas=metadatas)

store.save\_local('./zh\_data/nv\_embedding')

### Step 6b - 重读之前处理并保存的 Faiss Vectore 存储

# Load the vectorestore back.

store=FAISS.load\_local("./zh\_data/nv\_embedding",

embedder,allow\_dangerous\_deserialization=True)

### Step 7- 创建一个链式调用

# 假设您的store和llm已经定义好

store = FAISS.from\_texts(docs, embedder , metadatas=metadatas) # 您的知识库对象

llm = ChatNVIDIA(model="ai-phi-3-small-128k-instruct", nvidia\_api\_key="nvapi-OwhsA8DZUGPKwQXARE\_eDD3SfYIro0HZmJHogFcqOBc72MzppDP2\_EGzJ69m4Dcy", max\_tokens=512) # 您的语言模型对象

# 创建一个链式调用

def chain\_call(question, context):

retriever = store.as\_retriever()

prompt = ChatPromptTemplate.from\_messages(

[

(

"system",

"Answer solely based on the following context:\n<Documents>\n{context}\n</Documents>",

),

("user", "{question}"),

]

)

chain = (

{"context": retriever, "question": RunnablePassthrough()}

| prompt

| llm

| StrOutputParser()

)

return chain.invoke(question)

### Step 8- 创建 Gradio 界面

# 创建 Gradio 界面

iface = gr.Interface(

fn=chain\_call,

inputs=gr.Textbox(lines=2, placeholder="Enter your question here"),

outputs=gr.Textbox(),

title="Query Answering",

description="Ask any question and get an answer based on the provided context."

)

# 启动界面

iface.launch()

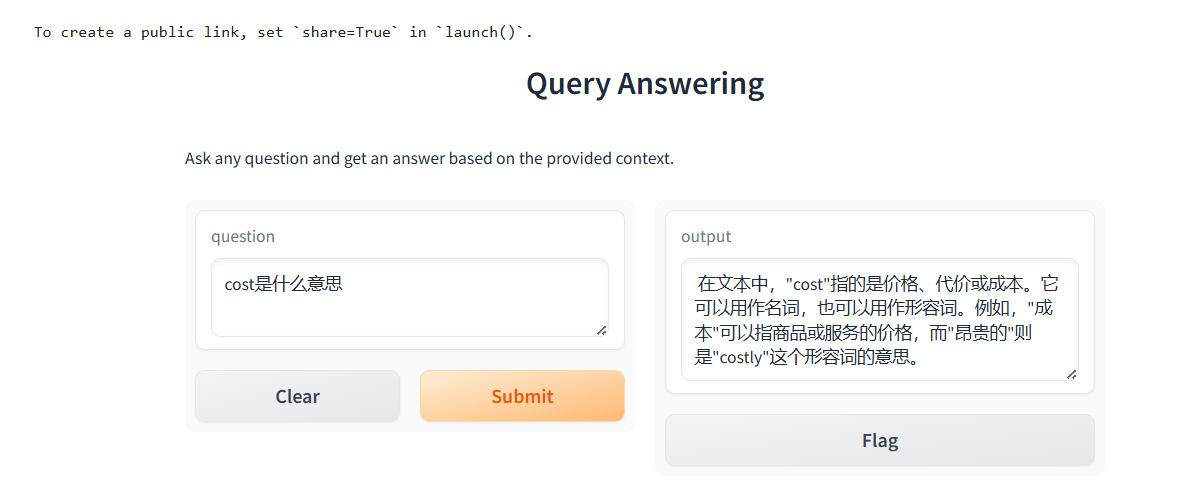
#### 项目成果与展示：

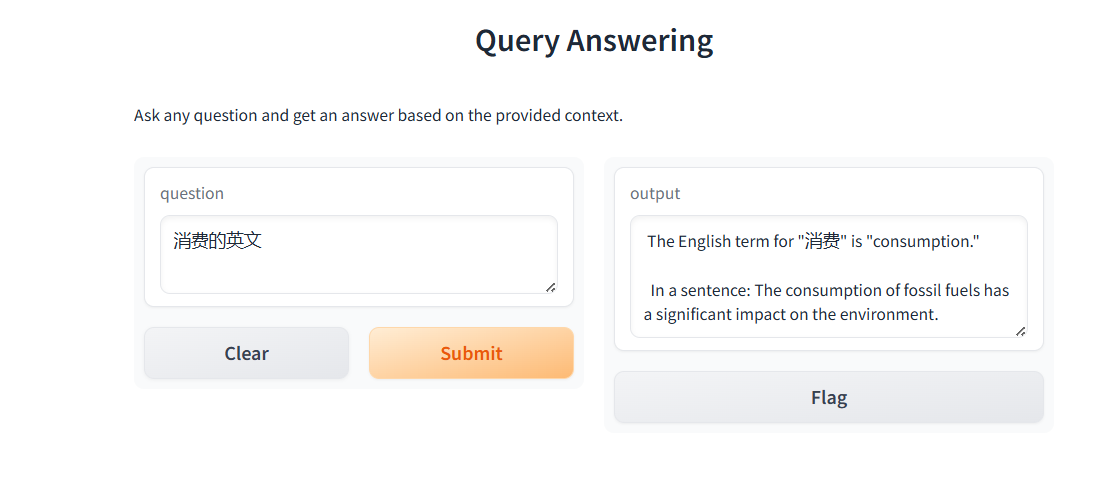
* 应用场景展示： 描述对话机器人的具体应用场景，如客户服务、教育辅导等。

Retrieval-augmented Generation (RAG)的文字形式的智能对话机器人，可用于帮助学生在学习英语的过程中提高对于单词的记忆能力。同时也能够提供个性化学习建议，配合易于使用的界面，可以使得学习过程更加便捷和高效，提高对于英语考试的信心。

* 功能演示： 列出并展示实现的主要功能，附上UI页面截图，直观展示项目成果。

**单词检索（目前仅支持部分c开头单词）**





**提供建议**



#### 问题与解决方案：

问题分析：在下载python时出现报错

WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by ……

WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=None)) after connection broken by ……

WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=None)) after connection broken by ……

WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) after connection broken by ……

WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=None)) after connection broken by ……

ERROR: Could not find a version that satisfies the requirement XXX (from v ersions: none)

ERROR: No matching distribution found for XXX

解决措施：一条命令直接永久设置，不然每次都要-i：pip config set global.index-url <https://pypi.tuna.tsinghua.edu.cn/simple>如果还是不行，更换阿里源

# 问题分析：AttributeError: module ‘gradio‘ has no attribute ‘outputs‘

解决措施：排除代码错误后，是版本问题，pip install gradio==3.50.2

问题分析：Exception: [401] Unauthorized

RequestID: d4cc467b-307209

Please check or regenerate your API key.

解决措施：更换 API key ，网址：[NVIDIA NIM | llama-3\_1-405b-instruct](https://build.nvidia.com/meta/llama-3_1-405b-instruct?snippet_tab=Python)

更多问题解决方案请参考：

https://blog.csdn.net/babylon3000/article/details/141217141?csdn\_share\_tail=%7B%22type%22%3A%22blog%22%2C%22rType%22%3A%22article%22%2C%22rId%22%3A%22141217141%22%2C%22source%22%3A%22babylon3000%22%7D

#### 项目总结与展望：

此项目目前仅实现了基本的文字对话模式，数据库有限，难以实现更复杂的功能，后期希望针对具体需求加入多模态，AI agent,语音等功能，满足更为丰富的需求。

#### 附件与参考资料

原文链接：<https://blog.csdn.net/oZuoShen123/article/details/139240311>

[视频：五分钟内构建 RAG 驱动的聊天机器人 - NVIDIA 技术博客](https://developer.nvidia.com/zh-cn/blog/video-build-a-rag-powered-chatbot-in-five-minutes/)

[全网最最实用--教你用gradio搭网页、部署模型等（代码粘贴即用、不讲虚的）\_gradio 局域网访问-CSDN博客](https://blog.csdn.net/m0_56243424/article/details/140298602)

[python-jupyter实现OpenAi语音对话聊天\_通过python ai的方式进行语音聊天-CSDN博客](https://blog.csdn.net/dfBeautifulLive/article/details/134291088)

[2024 NVIDIA开发者社区夏令营环境配置指南(Win & Mac)\_csdn 2024nvidia开发者-CSDN博客](https://blog.csdn.net/kunhe0512/article/details/140910139)

[列出项目报告中引用的所有附件和参考资料。]