# Build a LangChain RAG Agent

## Introduction

In this guide, we'll walk through how to create agents capable of answering questions based on any provided document using the Fetch.ai uAgents and ai_engine python libraries as well as openai and cohere. The aim is to assists you in building aLangChain Retrieval-Augmented Generation (RAG) Agent !

The process of creating RAG agents has been improved by offering a decentralized and modular framework through the uAgents and AI Engine libraries. These tools streamline the integration of AI models like OpenAI and Cohere, enabling more efficient and scalable development of intelligent agents. With enhanced interoperability, security, and resource management, this framework allows developers to quickly build and deploy sophisticated agents that can effectively answer questions based on any provided document, making the entire process faster and more robust.

ℹ Check out theAI Engine package(opens in a new tab) anduAgents(opens in a new tab) packages to download them and start integrating this tools within your Agents project!

Current version of the AI Engine package is0.6.0 . Current version of the uAgents package is0.17.1 . Let's dive into the LangChain RAG Agents development!

## Prerequisites

Make sure you have read the following resources before going on with this guide:

- Creating an agent
- Creating an interval task
- Communicating with other agents
- Agent Handlers
- Register in Almanac
- Almanac Contract
- Mailbox
- Utilising the Agentverse Mailroom service
- Protocols
- Agentverse Functions
- Register an Agent Function on the Agentverse

## API KEYs

Importantly, you will need two API keys to correctly go through this guide: one fromOpenAI and one fromCohere . Follow the steps provided below to obtain these keys:

### OpenAI API Key

1. Visit theOpenAI website(opens in a new tab)
2. .
3. Sign up or log in to your account.
4. Navigate to theAPI
5. section.
6. Generate or retrieve your API key.

### Cohere API Key

1. Visit theCohere website(opens in a new tab)
2. .
3. Sign up or log in to your account.
4. Go to theAPI Keys
5. section.
6. Copy an existing key or create a new one.: one from OpenAI and one from Cohere. Follow the steps below to obtain these keys.

## Project structure and overview

### Project structure

The project is structured as follows:

langchain-rag/ . ├── poetry.lock ├── pyproject.toml └── src ├── agents │ ├── langchain_rag_agent.py │ └── langchain_rag_user.py ├── main.py └── messages └── requests.py The source code directory (src ) contains the following directories and files:

- agents
- : Contains scripts for the LangChain agents.* langchain_rag_agent.py
-
    - : Script for the RAG agent (retrieves info, finds documents, generates answers).
-
    - langchain_rag_user.py: Script for the User agent (asks questions, handles responses).
- main.py
- : Starts both the RAG and user agents.
- messages
- : Defines custom message models.* requests.py
-
    - : Defines the RagRequest message model (question, URL, optional deep read).

### Environment variables

You'll need to set up environment variables for your project to run correctly. These variables include your API keys, which should be stored in a.env file within thesrc directory.

To do this, navigate to thesrc directory. Here, create and source the.env file. Within this one, add the following:

export COHERE_API_KEY="YOUR_COHERE_API_KEY" export OPENAI_API_KEY="YOUR_OPENAI_API_KEY"

## Langchain RAG setup

### Project dependencies

You'll need several Python packages for the project. These can be managed efficiently usingPoetry(opens in a new tab) . The following dependencies are needed for the correct development of the Langchain RAG Agent project:

[tool.poetry.dependencies] python = ">=3.10,<3.12" uagents = "*" requests = "^2.31.0" langchain = "^0.2.11" openai = "^1.12.0" langchain-openai = "^0.1.19" tiktoken = "^0.7.0" cohere = "^4.47" faiss-cpu = "^1.7.4" validators = "^0.22.0" uagents-ai-engine = "^0.4.0" unstructured = "^0.12.4"

## Messages Data Model

### RagRequest Model

We now need to define therequests.py file under themessages folder in the project.

windows echo .

    requests . py The script look like the following:

Self hosted requests.py from typing import Optional from uagents import Model , Field

class

RagRequest ( Model ): question :

str

=

Field ( description = "The question that the user wants to have an answer for." ) url :

str

=

Field (description = "The url of the docs where the answer is." ) deep_read : Optional [ str ]

=

Field ( description = "Specifies weather all nested pages referenced from the starting URL should be read or not. The value should be yes or no." , default = "no" , ) Here's a breakdown of theRagRequest message data Model:

- question (str)
- : This is the user's question that needs to be answered based on the provided website URL.
- url (str)
- : This is the URL of the website where the answer should be found.
- deep_read (Optional[str], default="no")
- : This optional field allows you to specify whether the RAG agent should follow and read nested pages (i.e., pages linked from the starting URL). Valid values are"yes"
- or"no"
- . Bydefault
- , the agent focuses only on the initial URL.

## Agents

### LangChain RAG Agent

This step involves setting up theLangChain Retrieval-Augmented Generation (RAG) Agent which is able to scrape web content, retrieve relevant documents, and generate answers to user queries based on that content. Let's create the file within theagents directory we created undersrc directory of our project and name itlangchain_rag_agent.py by using the following command:

windows echo .

> langchain_rag_agent . py The agent is defined as alocal agent with a Mailbox(opens in a new tab)and it is able to answer questions by fetching and summarizing information from a given website URL.

The script for this agent looks as follows:

Self hosted langchain_rag_agent.py import traceback from uagents import Agent , Context , Protocol import validators from messages . requests import RagRequest import os from langchain_openai import ChatOpenAI from langchain . prompts import ChatPromptTemplate from langchain_community . document_loaders import UnstructuredURLLoader import requests from bs4 import BeautifulSoup from urllib . parse import urlparse from langchain_openai import OpenAIEmbeddings from langchain_community . vectorstores import FAISS from langchain . retrievers import ContextualCompressionRetriever from langchain . retrievers . document_compressors import CohereRerank from ai_engine import UAgentResponse , UAgentResponseType import nltk from uagents . setup import fund_agent_if_low

nltk . download ( "punkt" ) nltk . download ( "averaged_perceptron_tagger" )

# LANGCHAIN_RAG_SEED

"YOUR_LANGCHAIN_RAG_SEED" AGENT_MAILBOX_KEY =

"YOUR_MAILBOX_KEY"

# agent

Agent ( name = "langchain-rag-agent" , seed = LANGCHAIN_RAG_SEED, mailbox = f " { AGENT_MAILBOX_KEY } @https://agentverse.ai" , )

fund_agent_if_low (agent.wallet. address ())

# docs_bot_protocol

Protocol ( "DocsBot" )

# PROMPT_TEMPLATE

""" Answer the question based only on the following context:

{context}

---

Answer the question based on the above context: {question} """

def

create_retriever ( ctx : Context ,

url :

str ,

deep_read :

bool ) -> ContextualCompressionRetriever: def

scrape ( site :

str ): if

not validators . url (site): ctx . logger . info ( f "Url { site } is not valid" ) return

## r

requests . get (site) soup =

BeautifulSoup (r.text, "html.parser" )

## parsed_url

urlparse (url) base_domain = parsed_url . scheme +

"://"

+ parsed_url . netloc

## link_array

soup . find_all ( "a" ) for link in link_array : href :

str

= link . get ( "href" , "" ) if

len (href)

==

0 : continue current_site =

f " { base_domain }{ href } "

if href . startswith ( "/" )

else href if ( ".php"

in current_site or

"#"

in current_site or

not current_site . startswith (url) or current_site in urls ) : continue urls . append (current_site) scrape (current_site)

## urls

[url] if deep_read : scrape (url) ctx . logger . info ( f "After deep scraping - urls to parse: { urls } " )

try : loader =

UnstructuredURLLoader (urls = urls) docs = loader . load_and_split () db = FAISS . from_documents (docs, OpenAIEmbeddings ()) compression_retriever =

ContextualCompressionRetriever ( base_compressor = CohereRerank (), base_retriever = db. as_retriever () ) return compression_retriever except

Exception

as exc : ctx . logger . error ( f "Error happened: { exc } " ) traceback . format_exception (exc)

@docs_bot_protocol . on_message (model = RagRequest, replies = {UAgentResponse}) async

def

answer_question ( ctx : Context ,

sender :

str ,

msg : RagRequest): ctx . logger . info ( f "Received message from { sender } , session: { ctx.session } " ) ctx . logger . info ( f "input url: { msg.url } , question: { msg.question } , is deep scraping: { msg.deep_read } " )

# parsed_url

urlparse (msg.url) if

not parsed_url . scheme or

not parsed_url . netloc : ctx . logger . error ( "invalid input url" ) await ctx . send ( sender, UAgentResponse ( message = "Input url is not valid" , type = UAgentResponseType.FINAL, ), ) return base_domain = parsed_url . scheme +

"://"

+ parsed_url . netloc ctx . logger . info ( f "Base domain: { base_domain } " )

# retriever

create_retriever (ctx, url = msg.url, deep_read = msg.deep_read ==

"yes" )

# compressed_docs

retriever . get_relevant_documents (msg.question) context_text =

"\n\n---\n\n" . join ([doc.page_content for doc in compressed_docs]) prompt_template = ChatPromptTemplate . from_template (PROMPT_TEMPLATE) prompt = prompt_template . format (context = context_text, question = msg.question)

# model

ChatOpenAI (model = "gpt-4o-mini" ) response = model . predict (prompt) ctx . logger . info ( f "Response: { response } " ) await ctx . send ( sender, UAgentResponse (message = response, type = UAgentResponseType.FINAL) )

agent . include (docs_bot_protocol, publish_manifest = True )

if

**name**

==

"**main**" : agent . run () In order to correctly run this code, you need to provide thename ,seed ,mailbox ,LANGCHAIN_RAG_SEED ,PROMPT_TEMPLATE andAGENT_MAILBOX_KEY parameters to correctly run this example.

The agent fetches and processes information from specified URLs to answer users' questions. Let's explore the script above in more detail. Initially, multiple modules are imported, including tools for making HTTP requests, parsing HTML, and handling Natural Language Processing (NLP). The agent usesOpenAI's GPT-4o-mini model for generating answers based on retrieved documents. We then up an agent calledlangchain_rag_agent() with a uniqueseed andmailbox address. Check out theMailbox guide for additional info on Agent Mailboxes.

To ensure the agent has sufficient funds to operate, thefund_agent_if_low() function is invoked.

Next, theDocsBot protocol is defined using theProtocol class of theuagents library to handle message interactions. This protocol uses a predefined prompt template to structure the questions and context for the language model.

Thecreate_retriever() function is responsible for fetching and parsing web pages. If necessary, it can perform deep scraping, which involves gathering all linked pages within the same domain. The function validates URLs, retrieves HTML content, and usesBeautifulSoup to parse the HTML and find links. The documents are then loaded and split using LangChain'sUnstructuredURLLoader , indexed withFAISS , and compressed withCohere , creating a retriever that can extract relevant information.

Theanswer_question() function, decorated with the.on_message() handler , is triggered when the agent receives a message matching theRagRequest message data model. The function validates the input URL and creates a retriever to fetch relevant documents based on the question. The context alongside with the question are then used to create a prompt for the language model (ChatOpenAI ), which generates the final answer.

The generated answer is then sent back to the user.

Finally, the agent is set to include theDocsBot protocol and is then run, which starts the agent and allows it to receive and process incoming messages.

### LangChain User Agent

We are now ready to define theLangChain User Agent which interacts with the LangChain RAG agent defined previously to ask a question based on documents found at specified URLs. The user agent sends a request to the RAG agent to retrieve and process information from the web page and then handle the response. The agent is defined as alocal agent with an endpoint(opens in a new tab) .

Create a file for this agent:

windows echo .

langchain_rag_user . py The script looks as follows:

Self hosted langchain_rag_user.py from uagents import Agent , Context , Protocol from messages . requests import RagRequest from ai_engine import UAgentResponse from uagents . setup import fund_agent_if_low

# QUESTION

"How to install uagents using pip" URL =

"https://fetch.ai/docs/guides/agents/installing-uagent" DEEP_READ = ( "no" )

# RAG_AGENT_ADDRESS

"YOUR_LANGCHAIN_RAG_AGENT_ADDRESS"

# user

Agent ( name = "langchain_rag_user" , port = 8000 , endpoint = [ "http://127.0.0.1:8000/submit" ], ) fund_agent_if_low (user.wallet. address ()) rag_user =

Protocol ( "LangChain RAG user" )

@rag_user . on_interval ( 60 , messages = RagRequest) async

def

ask_question ( ctx : Context): ctx . logger . info ( f "Asking RAG agent to answer { QUESTION } based on document located at { URL } , reading nested pages too: { DEEP_READ } " ) await ctx . send ( RAG_AGENT_ADDRESS, RagRequest (question = QUESTION, url = URL, deep_read = DEEP_READ) )

@rag_user . on_message (model = UAgentResponse) async

def

handle_data ( ctx : Context ,

sender :

str ,

data : UAgentResponse): ctx . logger . info ( f "Got response from RAG agent: { data.message } " )

user . include (rag_user)

if

**name**

==

"**main**" : rag_user . run () Remember that you need to provide theQUESTION ,URL ,DEEP_READ ,RAG_AGENT_ADDRESS ,name ,seed andendpoint parameters to correctly run this code.

Here, we have created the User Agent which interacts with the LangChain RAG Agent we previously defined. The User Agent periodically asks the RAG Agent a predefined question and then handles the response.

After importing all required modules and classes, a specificQUESTION is provided:"How to install uagents using pip" . We also provide theURL for the webpage where the relevant information can be found to answer the question. TheDEEP_READ variable is set tono thus indicating that the agent should only read the main page and not follow and read any nested or linked pages.

TheRAG_AGENT_ADDRESS variable holds the address of the RAG agent, which is responsible for retrieving and processing the webpage content to answer the user's question.

We are now ready to define the User Agent. We create thelangchain_rag_user is created with a specificport andendpoint , which is where it will communicate with the RAG Agent. The agent's wallet is funded if needed using thefund_agent_if_low() function.

We then proceed and define theprotocol . A protocol namedLangChain RAG user is defined using theProtocol class of theuagents library to handle the agent's interactions. This protocol contains two important functions:

1. Theask_question()
2. function is decorated with the.on_interval()
3. handler
4. and it is set to run at 60-second intervals. This function sends aRagRequest
5. message to the LangChain RAG agent, asking it to answer theQUESTION
6. based on the specifiedURL
7. . It then logs theQUESTION
8. andURL
9. details for debugging purposes.
10. Thehandle_data()
11. function is decorated with the.on_message()
12. handler and it handles incoming messages from the LangChain RAG agent. When a response is received, it logs the response message.

Finally, the LangChain RAG user protocol is then included using the.include() method into the agent which is then run by callingrag_user.run() in the main block.

## Main script

We are now ready to define the main script for our project. In thesrc folder of our project we create a Python script namedmain.py using the following command:

windows echo .

    main . py We then define the code within this one which looks like the one provided here below:

Self hosted main.py from uagents import Bureau from agents . langchain_rag_agent import agent from agents . langchain_rag_user import user

if

**name**

==

"**main**" : bureau =

Bureau (endpoint = "http://127.0.0.1:8000/submit" , port = 8000 ) print ( f "Adding RAG agent to Bureau: { agent.address } " ) bureau . add (agent) print ( f "Adding user agent to Bureau: { user.address } " ) bureau . add (user) bureau . run () Now, both of agents are set up, and you will need toconnect your local agents to the Agentverse so for them to be retrievable for communication and interaction with any other registered agent on the Fetch Network. This way, thelangchain_rag_agent will be connected to the Agentverse using aMailbox and afterward aAgent Function(opens in a new tab) will be created andregistered(opens in a new tab) for this agent.

Thelangchain_rag_user agent is used as a testing agent for the RAG agent being registered on the Agentverse and made subsequently available on DeltaV for queries.

ℹ Head over to the followingguide for a better understanding of local agent registration on the Agentverse.

## Expected output

The expected output for this example should be similar to the following one where we questioned theRAG Agent with the questionHow to install uagents using pip from theRAG user agent:

Last updated on October 18, 2024

## Was this page helpful?

## You can also leave detailed feedbackon Github

Hosted agent REST endpoints

On This Page