Multi-agent workflows with Fetch.ai x Langchain

Multi-agent workflows are at the forefront of modern agent development; the idea that individualAgents can be utilized to create larger more complex services for people has created lots of excitement in the AI space. At Fetch.ai, we're building the agent communication layer which perfectly compliments Langchain libraries.

Agents representing smaller parts of a service, allow for many agents to represent a whole. Agents reduce technical requirements in projects, for example you wouldn't need to write a function to calculate the historical index of a stock price, an agent will already return that data for you.

Before we go any further please read over our introduction guide toAgents and Langchain

The system

Three agents make up a simple agent workflow:RequestAgent ,PDFQuestionAgent andPDFSplitAgent .

A variation of the followingModel class is passed between each agent:

class

DocumentUnderstanding (Model): pdf_path :

str question :

str The flow is the following:

- RequestAgent
- provides theDocumentUnderstanding
- object.
- PDFQuestionAgent
- upon receivingDocumentUnderstanding
- sends a request to the third agent to validate and split the document.
- PDFSplitAgent
- upon receiving the request from PDFQuestionAgent, returns a list of pages to it.
- PDFQuestionAgent
- upto receiving the message from PDFSplitAgent processes the pages to answer the question from RequestAgent, then returns that answer to this latter one.

In this example, we are hard coding the agents addresses, meaning we know them. To search for agents dynamically, take a look at theAlmanac .

Installation

Run the following:

poetry

init poetry

add

uagents

requests

langchain

openai

langchain-openai

faiss-cpu

validators Versions used for this example are:

[tool.poetry.dependencies] python = ">=3.10,<3.12" uagents = "0.12.0" requests = "^2.31.0" langchain = "^0.1.7" openai = "^1.12.0" langchain-openai = "^0.0.6" faiss-cpu = "^1.7.4"

Environment setup

export OPENAI_API_KEY="YOUR_OPENAI_API_KEY"

Agent 1 - RequestAgent: provides a question and a source

This is our simplest agent; this agent provides a link to a PDF and question to be answered from the document.

Self hosted request_agent.py from uagents import Agent , Context , Protocol , Model

class

DocumentUnderstanding (Model): pdf_path :

str question :

str

class

DocumentsResponse (Model): learnings :

str

agent

Agent (name = "find_in_pdf" , seed = "" , port = 8001 , endpoint = ["http://127.0.0.1:8001/submit"],)

print ("uAgent address: " , agent.address) summary_protocol =

Protocol ("Text Summarizer")

AGENT_2_FAISS

""

@agent . on_event ("startup") async

def

on_startup (ctx : Context): await ctx . send (AGENT_2_FAISS, DocumentUnderstanding (pdf_path = "./a.pdf" , question = "What is the story about?"),)

@agent . on_message (model = DocumentsResponse) async

def

document_load (ctx : Context ,

sender :

str ,

msg : DocumentsResponse): ctx . logger . info (msg.learnings)

agent . include (summary_protocol, publish_manifest = True) agent . run ()

Agent 2 - PDFQuestionAgent: takes a request and returns a result

The PDFQuestionAgent gets the PDF and the request from the first agent, but is unable to split the PDF. This second agent sends a request to the third agent to split the PDF. Once the pages from the PDF are returned, a FAISS similarity search is ran on the pages by the second agent.

Self hosted pdf_question_agent.py from langchain_community . vectorstores import FAISS from langchain_community . docstore . in_memory import InMemoryDocstore from langchain_openai import OpenAIEmbeddings from uagents import Agent , Context , Protocol , Model from langchain_core . documents import Document from typing import List import os import uuid import faiss

class

PDF_Request (Model): pdf_path :

str

class

DocumentUnderstanding (Model): pdf_path :

str question :

str

class

PagesResponse (Model): pages : List

class

DocumentsResponse (Model): learnings :

str

faiss_pdf_agent

Agent (name = "faiss_pdf_agent" , seed = "" , port = 8002 , endpoint = ["http://127.0.0.1:8002/submit"],)

print ("uAgent address: " , faiss_pdf_agent.address) faiss_protocol =

Protocol ("FAISS")

RequestAgent

"" PDF_splitter_address =

""

openai_api_key

os . environ ["OPENAI_API_KEY"] embeddings =

OpenAIEmbeddings (model = "text-embedding-3-large")

@faiss_pdf_agent . on_message (model = DocumentUnderstanding, replies = PDF_Request) async

def

document_load (ctx : Context ,

sender :

str ,

msg : DocumentUnderstanding): ctx . logger . info (msg) ctx . storage . set (str (ctx.session), { "question" : msg.question, "sender" : sender}) await ctx . send (PDF_splitter_address, PDF_Request (pdf_path = msg.pdf_path))

@faiss_pdf_agent . on_message (model = PagesResponse, replies = DocumentsResponse) async

def

document_understand (ctx : Context ,

sender :

str ,

msg : PagesResponse): index = faiss . IndexFlatL2 (len (embeddings. embed_query ("hello")))

vector_store

FAISS (embedding_function = embeddings, index = index, docstore = InMemoryDocstore (), index_to_docstore_id = {},)

documents

[] for page in msg . pages : documents . append (Document (page_content = page["page_content"], metadata = page["metadata"]))

uuids

[str (uuid. uuid4 ())

for _ in

range (len (documents))]

vector_store . add_documents (documents = documents, ids = uuids)

prev

ctx . storage . get (str (ctx.session))

results

vector_store . similarity_search (prev["question"], k = 2 ,)

if

len (results)

0 : await ctx . send (prev["sender"], DocumentsResponse (learnings = results[0].page_content))

faiss_pdf_agent . include (faiss_protocol, publish_manifest = True) faiss_pdf_agent . run () The core difference with agent two compared to other agents you have seen so far is that there are multipleon_message decorators. Your agent can have as many any number of message handlers as you want.

PDFQuestionAgent also has every request/response model to communicate withRequestAgent andPDFSplitAgent .

Agent 3 - PDFSplitAgent: validates the PDF, loads and returns the split

The PDFSplitAgent receives the PDF, and splits the document using thelangchain_community document loaderPyPDFLoader . It then returns an array of pages to the second agent.

Self hosted pdf_split_agent.py from langchain_community . document_loaders import PyPDFLoader from uagents import Agent , Context , Protocol , Model from typing import List

class

PDF_Request (Model): pdf_path :

str

class

PagesResponse (Model): pages : List

pdf_loader_agent

Agent (name = "pdf_loader_agent" , seed = "" , port = 8003 , endpoint = ["http://127.0.0.1:8003/submit"],)

print ("uAgent address: " , pdf_loader_agent.address) pdf_loader_protocol =

Protocol ("Text Summarizer")

@pdf_loader_agent . on_message (model = PDF_Request, replies = PagesResponse) async

def

document_load (ctx : Context ,

sender :

str ,

msg : PDF_Request): loader =

PyPDFLoader (msg.pdf_path) pages = loader . load_and_split () await ctx . send (sender, PagesResponse (pages = pages))

pdf_loader_agent . include (pdf_loader_protocol, publish_manifest = True) pdf_loader_agent . run ()

Run the agents

We need to run the agents backwards so that we can generate their addresses and the update the other agents with their addresses respectively.

Let's runPDFSplitAgent , and updatePDFQuestionAgent with its address:

Run:poetry run python pdf_split_agent.py

Updatepdf_question_agent.py script by filling thePDF_splitter_address field with the address of the third agent.

Runpoetry run python pdf_question_agent.py

Updaterequest_agent.py script by filling theAGENT_2_FAISS field with the address of the second agent.

Runpoetry run python request_agent.py

Add the address of the first agent in the dedicated fieldRequestAgent within the script for the second agent,pdf_question_agent.py .

Expected Output

- request_agent.py
- :
- uAgent address: agent1qf4au6rzaauxhy2jze6v85rspgvredx9m42p0e0cukz0hv4dh2sqjuhujpp
- INFO: [find_in_pdf]: Manifest published successfully: Text Summarizer
- INFO: [find_in_pdf]: Registering on almanac contract...
- INFO: [find_in_pdf]: Registering on almanac contract...complete
- INFO: [find_in_pdf]: Starting server on http://0.0.0.0:8001 (Press CTRL+C to quit)
- INFO: [find_in_pdf]: GuidesAI AgentsGetting StartedWhat's an Agent?
- Beginner Python
- Agents - uAgents Framework
- Introduction
- Agents are autonomous software programs that can commuicate with eachother, they're
- designed to solve problems alone, or as part of a multi agent system. Standardised
- communication protocols allows agents to communicate eciently, aiding negotiation and
- problem resolution. Unlike microservices or centralized systems, agents are reactive and self
- reasoning. Agents are built using the uAgents framework, and with additional libraries are
- party to the AI Engine.
- Agents oer intercontevity for larger systems, where centralised systmes may have some
- fault tolerance, multi agent systemsd allow for plug and play design; agents can be replaced
- or upgraded in a live system without downtime. Agents can be built to only care for one
- process or rule. In your systems, agents may have a totally independant network.
- Agents aren't just agents, you can wrap LLMs and other models to create self reasoning, self
- learning chat interfaces or services. This, is a simple as 12 lines of code link
- pdf_question_agent.py
- :
- INFO:faiss.loader:Loading faiss with AVX2 support.
- INFO:faiss.loader:Successfully loaded faiss with AVX2 support.
- uAgent address: agent1qt89fz44fp0nxvkpgfts4lm566lj8gs7qnlh7yz3lwz5f5scp7nrkcpt3qe
- INFO: [faiss_pdf_agent]: Manifest published successfully: FAISS
- INFO: [faiss_pdf_agent]: Registration on Almanac API successful
- INFO: [faiss_pdf_agent]: Registering on almanac contract...
- INFO: [faiss_pdf_agent]: Registering on almanac contract...complete
- INFO: [faiss_pdf_agent]: Starting server on http://0.0.0.0:8002 (Press CTRL+C to quit)
- INFO: [faiss_pdf_agent]: pdf_path='./a.pdf' question='What is the story about?'
- INFO:httpx:HTTP Request: POST https://api.openai.com/v1/embeddings "HTTP/1.1 200 OK"

- INFO:httpx:HTTP Request: POST https://api.openai.com/v1/embeddings "HTTP/1.1 200 OK"
- INFO:httpx:HTTP Request: POST https://api.openai.com/v1/embeddings "HTTP/1.1 200 OK"
- pdf_split_agent.py
- :
- uAgent address: agent1qf4au6rzaauxhy2jze6v85rspgvredx9m42p0e0cukz0hv4dh2sqjuhujpp
- INFO: [pdf_loader_agent]: Manifest published successfully: Text Summarizer
- INFO: [pdf_loader_agent]: Registering on almanac contract...
- INFO: [pdf_loader_agent]: Registering on almanac contract...complete
- INFO: [pdf_loader_agent]: Starting server on http://0.0.0.0:8003 (Press CTRL+C to quit)

Last updated on October 17, 2024

Was this page helpful?

You can also leave detailed feedbackon Github

Getting started with Fetch.ai x Langchain Getting started with Fetch.ai and Swarm

On This Page