iAssist

* *A bot with cognitive intelligence*

**

Prepared By - Barun Kumar 

[**Introduction 3**](#_unm0kjuw4w2h)

[**Agent 3**](#_wujl98490xcy)

[**Tool 3**](#_gurvjk99n0ew)

[**Architecture and working 3**](#_j6n7lw3uio0n)

[**iAssist 4**](#_j2e5mlsp3uo8)

[Code 4](#_2ux8nlbwloz)

[Project Structure 4](#_sj3t6d7vrk0t)

[Bundling tools 5](#_vqqnpg2it0i2)

[Associated functions 5](#_5eumwup3in4a)

[Agent 7](#_rztqmuuy26ab)

[Complete Project 8](#_9k197kejhenl)

[iAssist in action 8](#_v8w7k1oq6stv)

[Creating vector db 8](#_1b6dj6924ggx)

[Using IBAN validator tool 9](#_ydvjgcqkng08)

[Using troubleshooting tool 10](#_h8ccl6wc58gn)

[Using In-Memory DB tool 10](#_popg49c4fs1l)

[Using Wikipedia tool 10](#_cmpi1qv1b7f3)

[Agent’s Cognitive intelligent 11](#_ggb7z23v8qpp)

[**Possible Use Cases 12**](#_5g5yu8iui9sv)

[**Challenges 12**](#_o9esxevf6lrp)

[**References 13**](#_s8x8wdgx3brp)

# 

# Introduction

With access to models, like GPT, we have abundance of information in hand; but models come with a cut off date. Meaning, the last point at which the training data was updated, Any information beyond this date is not included in the model's knowledge base.

To solve this, we have Retrieval-Augmented Generation (RAG) using which we can plug-in our data. Now, with this, **Model Information + Our data in vector store = Improved accuracy, relevance, and grounding of the LLM's generated text.** But, isn’t it still one-dimensional, where we have a limited set of knowledge base? In our day to day life we may need information stored in Databases, code logic, An external REST API call etc. and we can’t put all these in the Vector Store for RAG to leverage.

**Agent with Tools** solves all these problems, **Tools** are specific functionalities or external resources that agents can use. **Agents** use these resources and cognitively come up with a final answer.

**iAssist leverages capabilities of Agents and Tools, to act cognitively and provide refined response.**

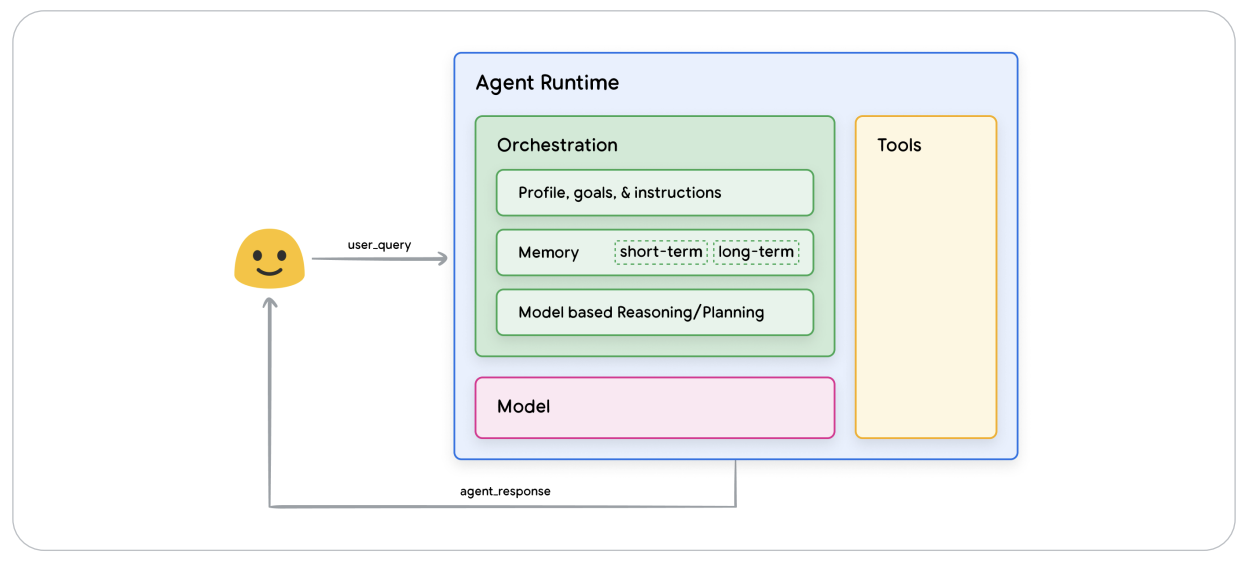
# Agent

* Gen AI agents are software systems that mimic human cognitive capabilities to some extent. I.e. Agents can act on their own, make decisions and perform tasks based on tools it has at its disposal and achieve goals. They can process information, converse, reason, and learn.

# Tool

For an Agent program, tools are key to the outside world, be it Database, REST API, a code logic etc.

# Architecture and working



To know more refer to sections like Cognitive architectures: How agents operate, Agent vs Model etc in [Google’s AI Agent whitepaper](https://archive.org/details/google-ai-agents-whitepaper/mode/2up).

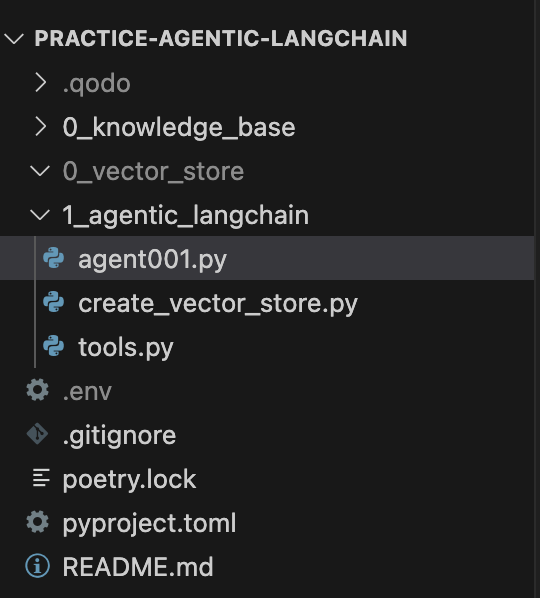
# iAssist

iAssist harnesses power of Agent and Tools and serves purposes like -

* Validate an IBAN: Tool refers to a **custom method.**
* Find troubleshooting steps: Tool refers to **RAG Vector store**
* Check if there is any message in dead letter queue table: Tool refers **in-memory table**
* Get any information from Wikipedia: Tool refers [**https://en.wikipedia.org/**](https://en.wikipedia.org/)
* Any QnA: **gpt-4o** model

## Code

### Project Structure



### Bundling tools

def get\_tools():

tools = [

Tool(

name="Validate IBAN", # Name of the tool

func=validate\_iban, # Function that the tool will execute

description="Useful for when you need to know the current time", # Description of the tool

),

Tool(

name="Troubleshoot",

func=troubleshoot,

description="Find troubleshooting step for given error",

),

Tool(

name="KafkaDeadletter",

func=kafka\_deadletter\_check,

description="Useful for when you need to check kafka dead letter",

),

Tool(

name="Wikipedia",

func=search\_wikipedia,

description="Useful for when you need to know information about a topic.",

),

Tool(

name="Answer Question",

func=lambda input, \*\*kwargs: rag\_chain.invoke(

{"input": input, "chat\_history": kwargs.get("chat\_history", [])}

),

description="useful for when you need to answer questions about the context",

),

]

return tools

### Associated functions

def validate\_iban(iban):

"""

Validates an IBAN using the ISO 7064 MOD 97-10 algorithm.

Example - Let's say we have the IBAN: DE41500105170123456789

1. Rearrange: The string becomes: 500105170123456789DE41.

2. Convert letters: 5001051701234567891314.

3. Integer Conversion: 5001051701234567891314.

4. Modulo 97: The remainder of this integer divided by 97 is 1.

5. Validation: Since the remainder is 1, the check digit test is passed.

Args:

iban: The IBAN string to validate.

Returns:

True if the IBAN is valid, False otherwise.

"""

iban = iban.replace(" ", "").upper()

if not iban.isalnum():

return False

if len(iban) < 15 or len(iban) > 34:

return False

moved\_iban = iban[4:] + iban[:4]

numeric\_iban = "".join(str(ord(char) - 55) if char.isalpha() else char for char in moved\_iban)

return int(numeric\_iban) % 97 == 1

def troubleshoot(troubleshoot: str) -> bool:

# Retrieve relevant documents based on the query

retriever = db.as\_retriever(

search\_type="similarity\_score\_threshold",

search\_kwargs={"k": 3, "score\_threshold": 0.1},

)

relevant\_docs = retriever.invoke(troubleshoot)

return relevant\_docs

def kafka\_deadletter\_check(\*args, \*\*kwargs):

# Connect to an in-memory database

conn = sqlite3.connect(':memory:')

# Create a cursor object to execute SQL commands

cursor = conn.cursor()

# Create a table

cursor.execute('''

CREATE TABLE IF NOT EXISTS kafka\_deadletter (

id INTEGER PRIMARY KEY,

event TEXT NOT NULL

)

''')

#delete existing data

cursor.execute("DELETE FROM kafka\_deadletter")

# Insert data into the table

cursor.execute("INSERT INTO kafka\_deadletter (id, event) VALUES (?, ?)", (101, 'CreateEvent'))

cursor.execute("INSERT INTO kafka\_deadletter (id, event) VALUES (?, ?)", (102, 'UpdateEvent'))

# Commit the changes

conn.commit()

# Retrieve data from the table

cursor.execute("SELECT \* FROM kafka\_deadletter")

rows = cursor.fetchall()

# Close the connection

conn.close()

return rows

def search\_wikipedia(query):

"""Searches Wikipedia and returns the summary of the first result."""

from wikipedia import summary

try:

# Limit to two sentences for brevity

return summary(query, sentences=2)

except:

return "I couldn't find any information on that."

### Agent

Set Up ReAct Agent with Document Store Retriever

# Load the ReAct Docstore Prompt

# https://smith.langchain.com/hub/hwchase17/react

react\_docstore\_prompt = hub.pull("hwchase17/react")

# Create the ReAct Agent with document store retriever

agent = create\_react\_agent(

llm=get\_llm(),

tools=get\_tools(),

prompt=react\_docstore\_prompt,

)

agent\_executor = AgentExecutor.from\_agent\_and\_tools(

agent=agent, tools=get\_tools(), handle\_parsing\_errors=True, verbose=True,

)

chat\_history = []

while True:

query = input("\nYou: ")

if query.lower() == "exit":

break

response = agent\_executor.invoke(

{"input": query, "chat\_history": chat\_history})

print(f"\nAgent: {response['output']}")

# Update history

chat\_history.append(HumanMessage(content=query))

chat\_history.append(AIMessage(content=response["output"]))

### Complete Project

<https://github.com/barunkumar04/practice-agentic-langchain>

## iAssist in action

### Creating vector db

> python 1\_agentic\_langchain/create\_vector\_store.py   
USER\_AGENT environment variable not set, consider setting it to identify your requests.  
  
  
Knowledge base: /Users/barunkumar04/IdeaProjects/practice-agentic-langchain/1\_agentic\_langchain/../0\_knowledge\_base  
Persistent directory: /Users/barunkumar04/IdeaProjects/practice-agentic-langchain/1\_agentic\_langchain/../0\_vector\_store/chroma\_db\_with\_metadata  
  
Persistent directory does not exist. Initializing vector store...  
Created a chunk of size 140, which is longer than the specified 100  
Created a chunk of size 189, which is longer than the specified 100  
Created a chunk of size 137, which is longer than the specified 100  
Created a chunk of size 149, which is longer than the specified 100  
Created a chunk of size 150, which is longer than the specified 100  
Created a chunk of size 234, which is longer than the specified 100  
Created a chunk of size 483, which is longer than the specified 100  
Created a chunk of size 134, which is longer than the specified 100  
Created a chunk of size 194, which is longer than the specified 100  
Created a chunk of size 309, which is longer than the specified 100  
Created a chunk of size 133, which is longer than the specified 100  
Created a chunk of size 152, which is longer than the specified 100  
Created a chunk of size 298, which is longer than the specified 100  
Created a chunk of size 163, which is longer than the specified 100  
Created a chunk of size 160, which is longer than the specified 100  
Created a chunk of size 108, which is longer than the specified 100  
Created a chunk of size 139, which is longer than the specified 100  
Created a chunk of size 180, which is longer than the specified 100  
Created a chunk of size 230, which is longer than the specified 100  
Created a chunk of size 195, which is longer than the specified 100  
Created a chunk of size 218, which is longer than the specified 100  
Created a chunk of size 166, which is longer than the specified 100  
Created a chunk of size 320, which is longer than the specified 100  
Created a chunk of size 128, which is longer than the specified 100  
Created a chunk of size 118, which is longer than the specified 100  
Created a chunk of size 119, which is longer than the specified 100  
Created a chunk of size 2028, which is longer than the specified 100  
Created a chunk of size 504, which is longer than the specified 100  
  
  
Number of document chunks: 96.  
Finished creating embeddings.  
Finished creating and persisting vector store.

### Using IBAN validator tool

python 1\_agentic\_langchain/agent001.py

Loading existing vector store…

You: Is DE41500105170123456789 a valid IBAN?

Agent: Yes, DE41500105170123456789 is a valid IBAN.

You: Which country DE41500105170123456789 belongs to?

Agent: The IBAN DE41500105170123456789 belongs to Germany.

You:

### Using troubleshooting tool

You: Troubleshooting steps of java.lang.IllegalStateException.

Agent: The "java.lang.IllegalStateException" occurs when an operation is attempted on an event that is not in the correct state, specifically when the event status must be "approved." To troubleshoot, ensure that the event status is set to "approved" before performing the operation.

You:

### Using In-Memory DB tool

You: Is there something in the dead letter?

Agent: Yes, there are entries in the Kafka dead letter queue. The entries are (101, 'CreateEvent') and (102, 'UpdateEvent').

You:

### Using Wikipedia tool

You: What is Stock Plan Management?

Agent: Stock Plan Management refers to the administration and management of a company's employee stock plans, which may include stock options, restricted stock units (RSUs), employee stock purchase plans (ESPPs), and other equity compensation programs. It involves tasks such as tracking stock grants, managing vesting schedules, ensuring compliance with regulatory requirements, and facilitating the exercise and sale of shares. The goal of stock plan management is to effectively manage the equity compensation offered to employees, aligning their interests with those of the company and its shareholders.

You:

### Agent’s Cognitive intelligent

* To visualize this enabled verbose=True while configuring Agent Executer
* Notice, based on the prompt it used Wikipedia, when it didn’t get any meaningful result. It used its own knowledge base - gpt-4o - without any explicit instruction from the user.

You: What does Global Shares do?

> Entering new AgentExecutor chain...

Global Shares is likely a company or organization, and to provide accurate information about it, I should look up its details. The best way to find this information is to use the Wikipedia tool to get a comprehensive overview.

Action: Wikipedia

Action Input: Global Shares

I couldn't find any information on that.

It seems that there is no Wikipedia entry for "Global Shares." This could mean that it is a lesser-known company or organization, or it might not have a dedicated Wikipedia page. To provide an answer, I will use my general knowledge.

Global Shares is a company that provides equity compensation management solutions. They offer software and services to manage employee stock plans, including stock options, restricted stock units (RSUs), and other forms of equity compensation. Their platform is used by companies to streamline the administration of equity plans, ensure compliance, and provide employees with tools to manage their equity holdings.

Final Answer: Global Shares is a company that provides equity compensation management solutions, offering software and services to manage employee stock plans and other forms of equity compensation.

> Finished chain.

Agent: Global Shares is a company that provides equity compensation management solutions, offering software and services to manage employee stock plans and other forms of equity compensation.

You:

# Possible Use Cases

* Help/automate production support rota, using RAG support
* Perform task using Custom Logic and REST API support
* Help getting information out there, using Web Scraping+RAG, Wkipidia, GPT support
* SQL as simple english, using DB support.

# Challenges

* Data Confidentiality as it access GPT, but an in-house LLM is a solution.
* Authentication and authorization

# References

1. <https://brandonhancock.io/langchain-master-class>
2. <https://archive.org/details/google-ai-agents-whitepaper/mode/2up>