✨ Member-only story

Build a Chatbot for Clinical Trials Across Multiple Data Sources

Unify the access to a knowledge graph, a vector database and the web in a LangChain chatbot

Sixing Huang · Following
8 min read · 6 hours ago

Listen Share ••• More

Clinical research plays a crucial role in advancing medical knowledge, improving patient care, and developing innovative treatments and therapies. It evaluates the safety and efficacy of medical interventions, including pharmaceuticals, medical devices, and behavioral interventions among human participants. Clinical research metadata is systematically uploaded to the widely recognized web portal, clinicaltrials.gov. This invaluable platform serves as a hub for medical professionals, enabling them to stay abreast of the latest advancements and developments within the pharmaceutical industry.

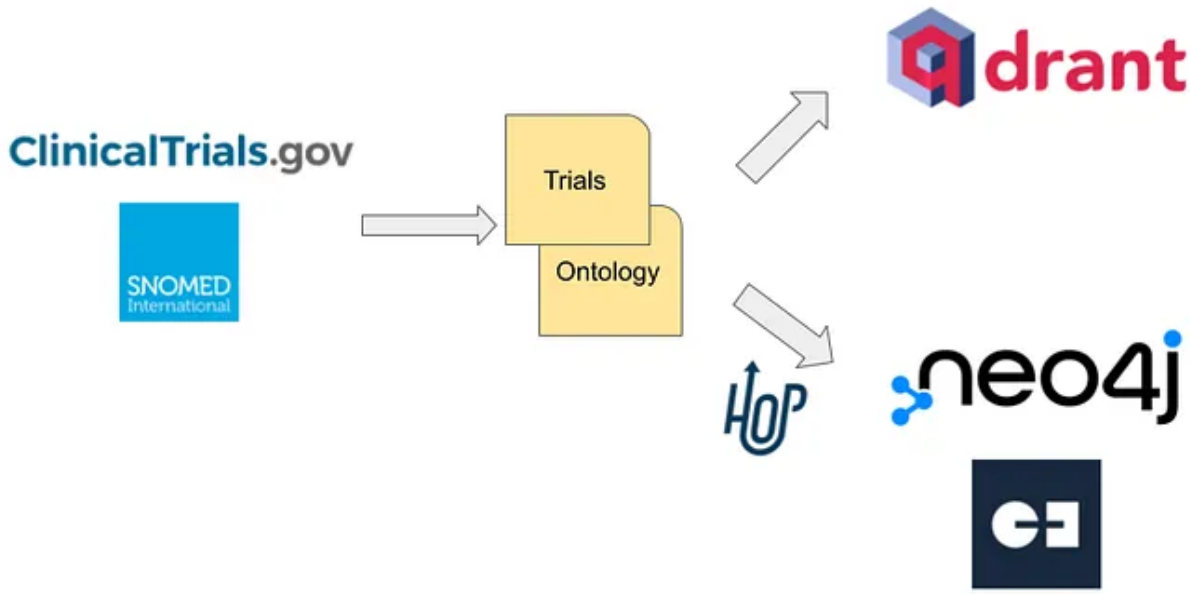


Figure 1. The architecture of the project Clinical Trials as Graphs and Vectors. Image by author.

However, navigating and searching through clinicaltrials.gov can be challenging and cumbersome. In order to enhance user experience, I undertook a project called *Clinical Trials as Graphs and Vectors*, where I restructured data from clinicaltrials.gov and SNOMED into a Neo4j knowledge graph and a Qdrant vector database (Figure 1). Users can analyze the graph with Cypher and search the vector database semantically. With these two databases, users can quickly see who are the competitors, how they design the trials, and what the results are.

However, this duo-database setup can be improved. First, the data was limited to clinicaltrials.gov and SNOMED. So when users ask outside questions, the solution will fail. Second, each data source has its own interface. When users want to use the vector search, they have to input their natural language questions in the vector database API. Similarly, they need to type Cypher into Neo4j if they want to search the graph. That will become a problem as we add more and more data sources to the stack.

This article is to tackle these issues. The result is a chatbot under the inspiration of Tomaz Bratanic's article *Integrating Neo4j into the LangChain ecosystem*. Powered by LangChain, it integrates the two data sources and adds web search into the mix. And it has the potential to include even more data sources such as SQL, NoSQL, and so on. Under the hood, the chatbot utilizes GPT-4. Faced with a natural language

question, the chatbot has the ability to search for answers from the relevant data source and generate comprehensive responses in complete sentences.

The code for this project is hosted on my GitHub repository here.

GitHub - dgg32/clinical_chatbot_public

Contribute to dgg32/clinical_chatbot_public development by creating an account on GitHub.

github.com

1. The architecture and data

The LangChain chatbot in this article is built on top of the two databases from *Clinical Trials as Graphs and Vectors*, namely a Neo4j knowledge graph and a free Qdrant vector database (Figure 2).

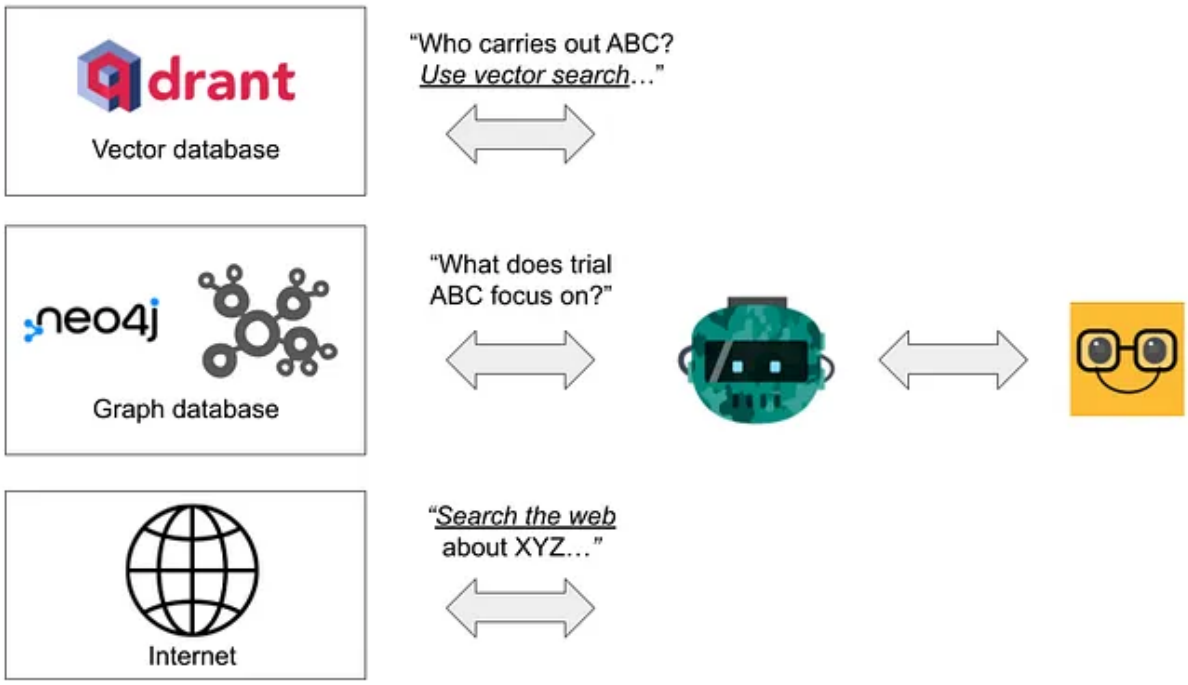


Figure 2. The LangChain architecture for the LangChain chatbot. Image by author.

The data for the two databases came from clinicaltrials.gov. Different from the last project, here I embedded all the fields in the trial nodes and uploaded them to Qdrant. The new addition is the SerpAPI from LangChain. It allows the chatbot to look for answers on the internet.

It is worth mentioning that the chatbot has memory. In other words, it can conduct context-aware conversions. It uses GPT-4 to understand and formulate language in English and even Japanese.

2. Python code for the tools, agents, and chatbot

The code is divided into two scripts: `agents.py` and `chatbot.py`. The first one configures the data accesses, while the second one sets up the web interface of the chatbot.

The `agents.py` contains three tools and an agent. Each tool has access to one data source. Each tool calls a chain to fulfill its function. For example, the chain for Neo4j looks like this, which is written by Tomaz Bratanic. Notice that his code automatically gets the node and edge labels from the Neo4j graph schema. So it no longer requires any prompt training for the natural language-to-Cypher conversion. Kudos to Tomaz.

```
...
with open("config.yaml", "r") as stream:
    try:
        PARAM = yaml.safe_load(stream)
...

graph = Neo4jGraph(
    url="bolt://localhost:7687", username=PARAM["neo4j_username"], password=PAR
)

graph.refresh_schema()

chain_neo4j = GraphCypherQAChain.from_llm(
    ChatOpenAI(temperature=0), graph=graph, verbose=True, return_direct=True
)
```

And the one for SerpAPI is quite simple.

```
os.environ["SERPAPI_API_KEY"] = PARAM["serpapi_key"]

serpai = SerpAPIWrapper()
```

And this is the one for Qdrant.

```
loader = CSVLoader("for_neo4j/node_trial.tsv", source_column="NCT", csv_args={
    'delimiter': '\t'
})
docs = loader.load()

url = PARAM["qdrant_URL"]
api_key = PARAM["qdrant_API_KEY"]

embeddings = OpenAIEmbeddings()

qdrant = Qdrant.from_documents(
    docs,
    embeddings,
    url = url,
    prefer_grpc=True,
    api_key=api_key,
    collection_name="my_documents",
)

retriever = qdrant.as_retriever()

qa = RetrievalQA.from_chain_type(llm=ChatOpenAI(model_name='gpt-4', temperature
```

We can assemble the chains into tools and put them into an agent.

```
tools = [
    Tool(
        name="Neo4j_search",
        func=chain_neo4j.run,
        description="useful for when you need to answer questions about clinica
    ),
    Tool(
        name="Vector_search",
        func=qa.run,
        description="Utilize this tool when the user asks for similarity search
    ),
    Tool(
        name="Serp_search",
        description="Utilize this tool when the user asks for information of a
        func=serpai.run,
    )
]
```

```
agent_instructions = "Try 'Neo4j_search' tool first. If Neo4j returns good resu

memory = ConversationBufferMemory(memory_key="chat_history", return_messages=Tr

custom_agent = ConversationalChatAgent.from_llm_and_tools(llm=ChatOpenAI(model_

agent_executor = AgentExecutor.from_agent_and_tools(agent = custom_agent, tools
agent_executor.verbose = True

def ask_question(question):
    return agent_executor.run(question)
```

The descriptions and instructions set the tool order. `Neo4j_search` is the default tool. `Vector_search` and `Serp_search` are launched when the user explicitly asks the chatbot to do so. The conversational buffer memory keeps track of the chat history, enabling the chatbot to resolve pronouns and engage in other context-aware activities. Afterward, an AgentExecutor bundles the agent and the memory together. Finally, I created a function `ask_question` for the chatbot to call.

The chatbot code is based on Tomaz Bratanic's work. However, his code for chat history management is no longer needed because LangChain has taken that task over.

```
import streamlit as st
from streamlit_chat import message
import agents

st.set_page_config(layout="wide")
st.title("Clinical Research Chatbot")

def get_text():
    input_text = st.text_input(
        "Powered by clincialtrials.gov and the web", "", key="input")
    return input_text

user_input = get_text()

if 'generated' not in st.session_state:
    st.session_state['generated'] = []

if 'user_input' not in st.session_state:
    st.session_state['user_input'] = []

if user_input:
```

```
        res = agents.ask_question(user_input)

        st.session_state.user_input.append(user_input)
        st.session_state.generated.append(res)

    if st.session_state['generated']:
        size = len(st.session_state['generated'])
        # Display only the last four exchanges

        for i in range(max(size-4, 0), size):
            if st.session_state['user_input'][i]:

                message(st.session_state['user_input'][i],is_user=True, key=str(i)

            if st.session_state["generated"][i]:

                message(st.session_state["generated"][i], key=str(i))
```

4. Test

Let's test the chatbot now. As you can see in Figure 3, I tested the chatbot with four questions.

The first question explicitly asked the chatbot to use vector search in Qdrant for its answer. It is worth noting that there was a misspelling in the word "trials," where it was mistakenly typed as "trails." And yet, the chatbot was able to process the question and come back with the right answer.

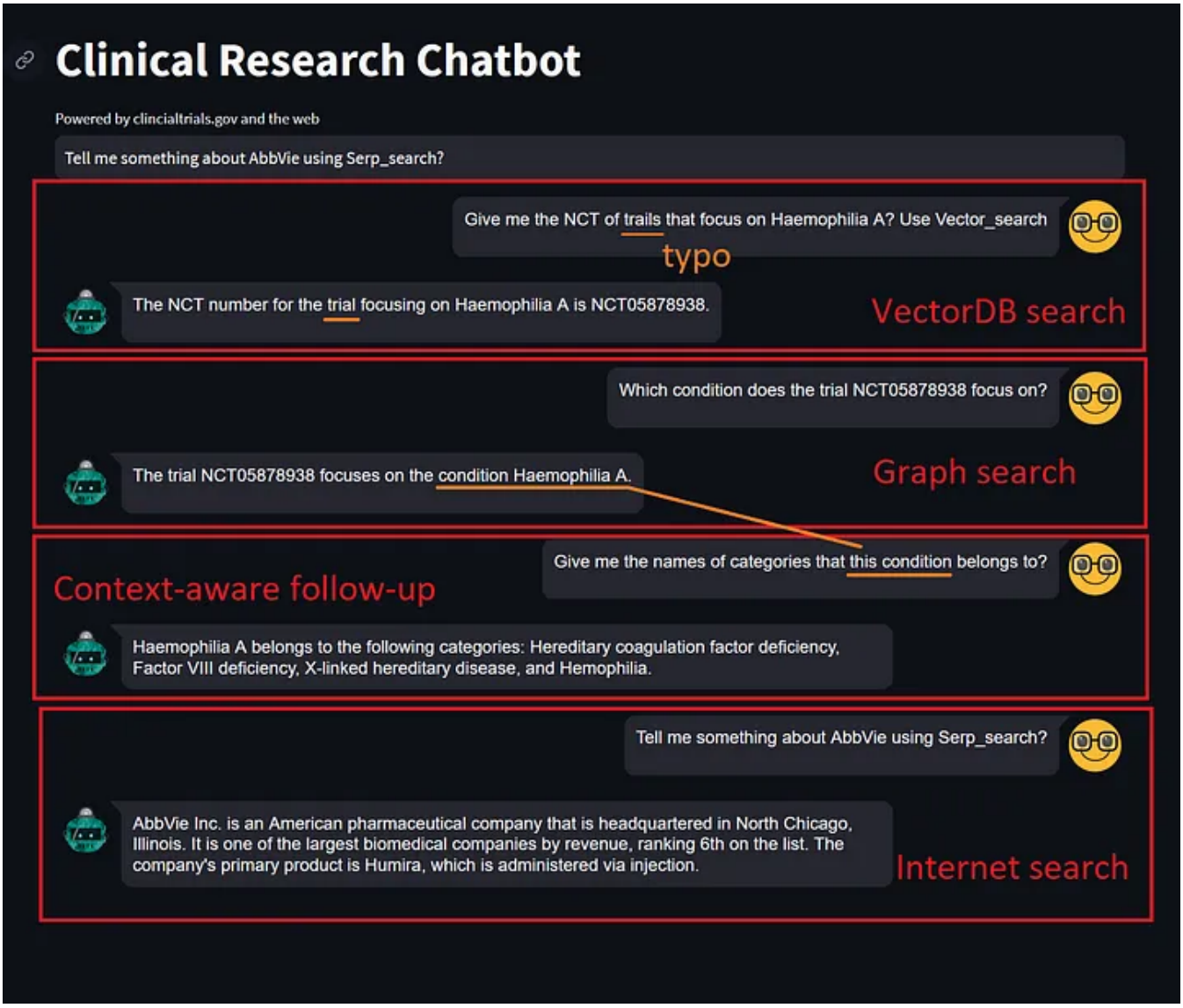


Figure 3. Four test rounds for the chatbot. Image by author.

The second question tested its ability in Cypher search. The chatbot aced that one easily. The third question tested the chatbot's context-awareness. I deliberately said "this condition", which referred to Haemophilia A from the previous round. And the chatbot was aware of that and assembled the right answer from the Neo4j knowledge graph.

Because clinicaltrials.gov does not contain information about drug companies and many other things, the final question requested a description of the pharmaceutical company AbbVie from the internet. The chatbot was competent for that task, too. Further investigation indicated that its answer came from the English Wikipedia.

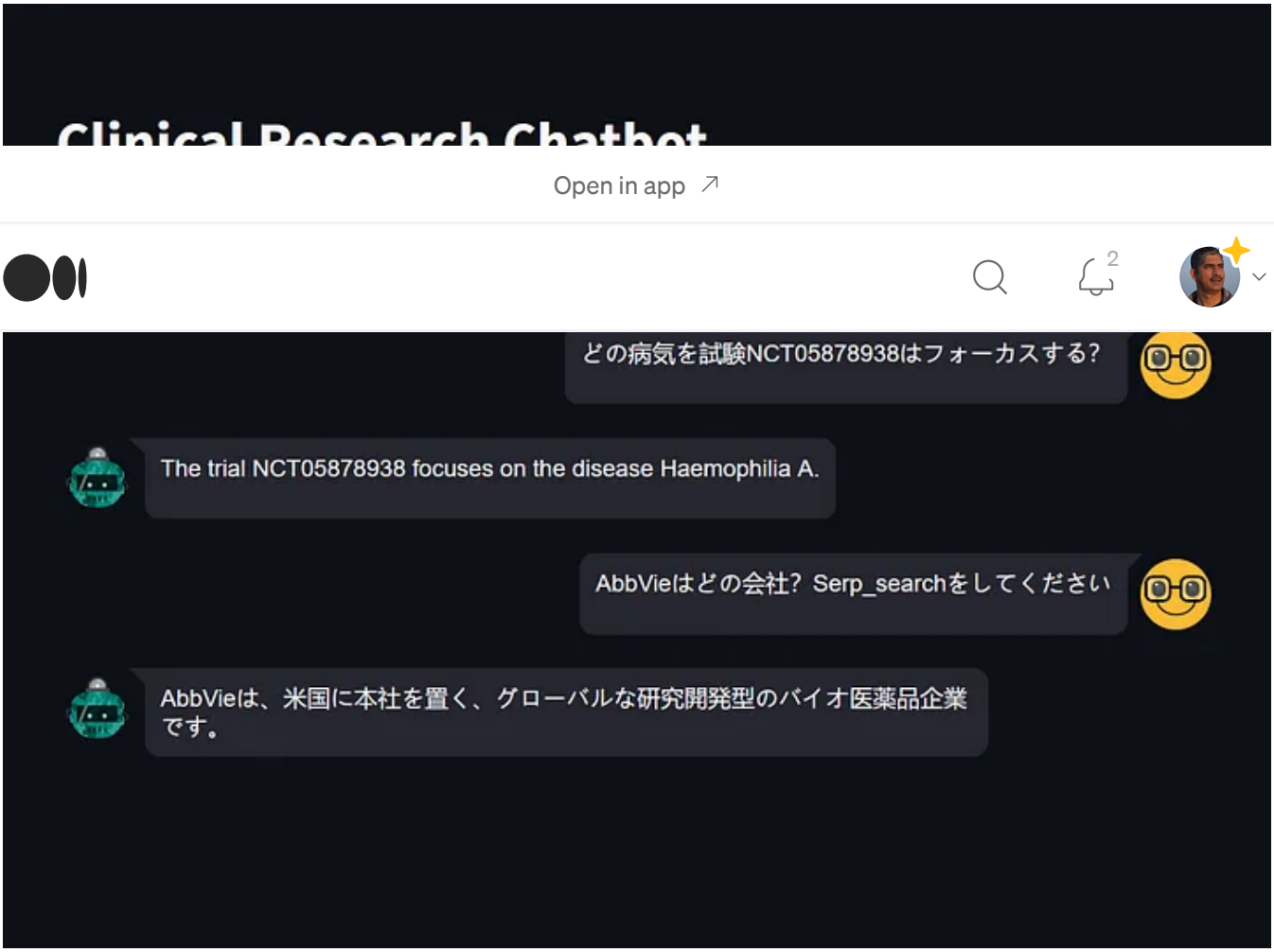


Figure 4. Two Japanese test rounds for the chatbot. Image by author.

Next, I tested the chatbot with some Japanese questions (Figure 4). The good news is that it was able to understand the questions and returned factually correct answers. But only the one from the web search was delivered in Japanese, while the `Neo4j_search` result was in English.

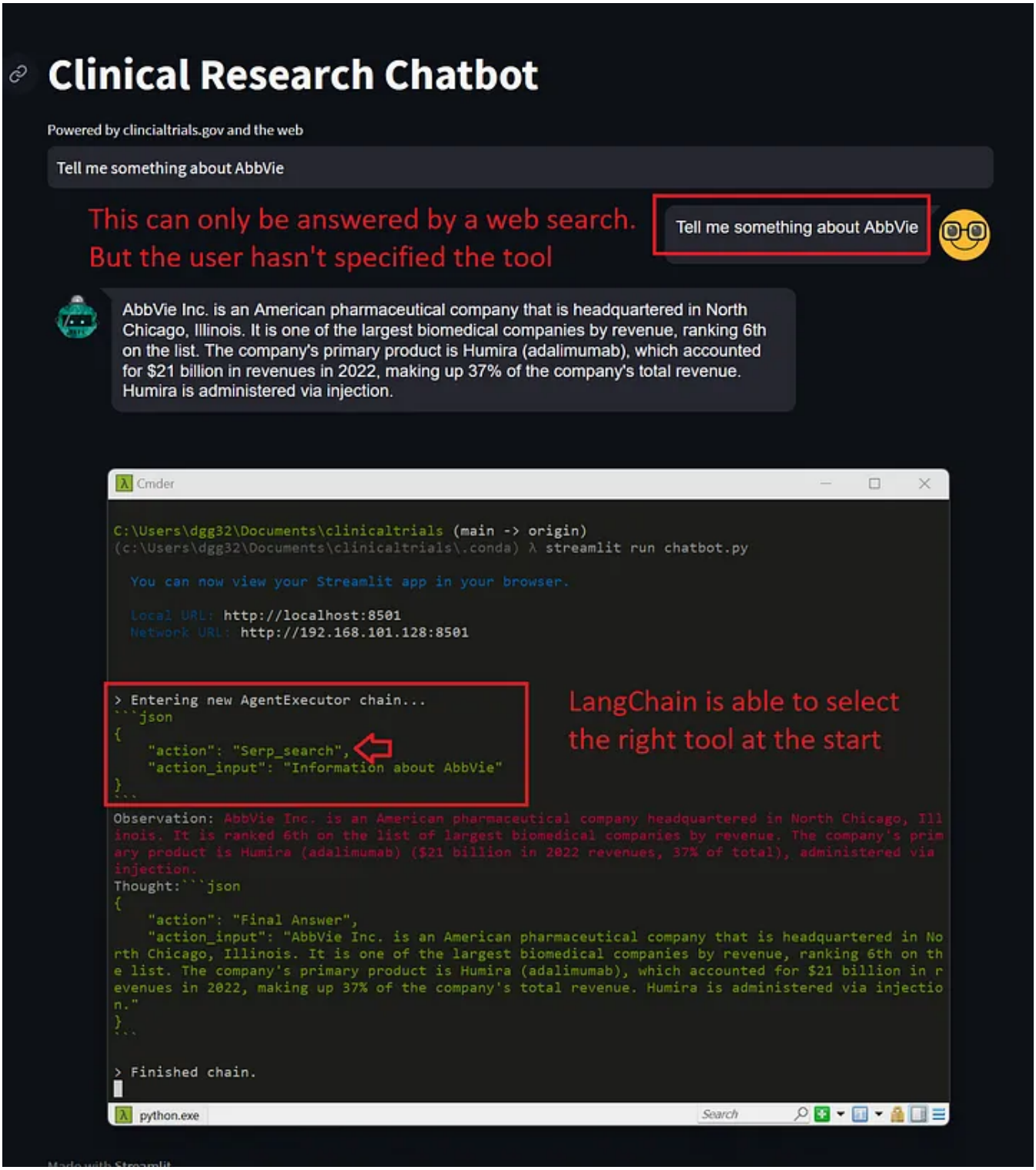


Figure 5. Two Japanese test rounds for the chatbot. Image by author.

Finally, what happens if the user asks an outside question without giving a tool hint? Will the chatbot waste its time searching the result first in Neo4j? I tested that scenario with the question, "*Tell me something about AbbVie*". This question can only be answered by a web search, and I intentionally did not specify the tool. And yet, the chatbot was able to select the right tool at the start and answer the question.

Conclusion

In this short article, I have demonstrated to you a simple chatbot that can query multiple data sources, including a local Neo4j knowledge graph, a cloud vector database, and the internet. One of the challenges that many enterprises are facing is that their data is stored in separate departments, in many formats (SQL, NoSQL, and PDF), and on multiple platforms (on-prem and cloud). Their top priority is to integrate them so that data consumers can access them all with a unified interface. The chatbot in this article is one possible solution (Figure 5).

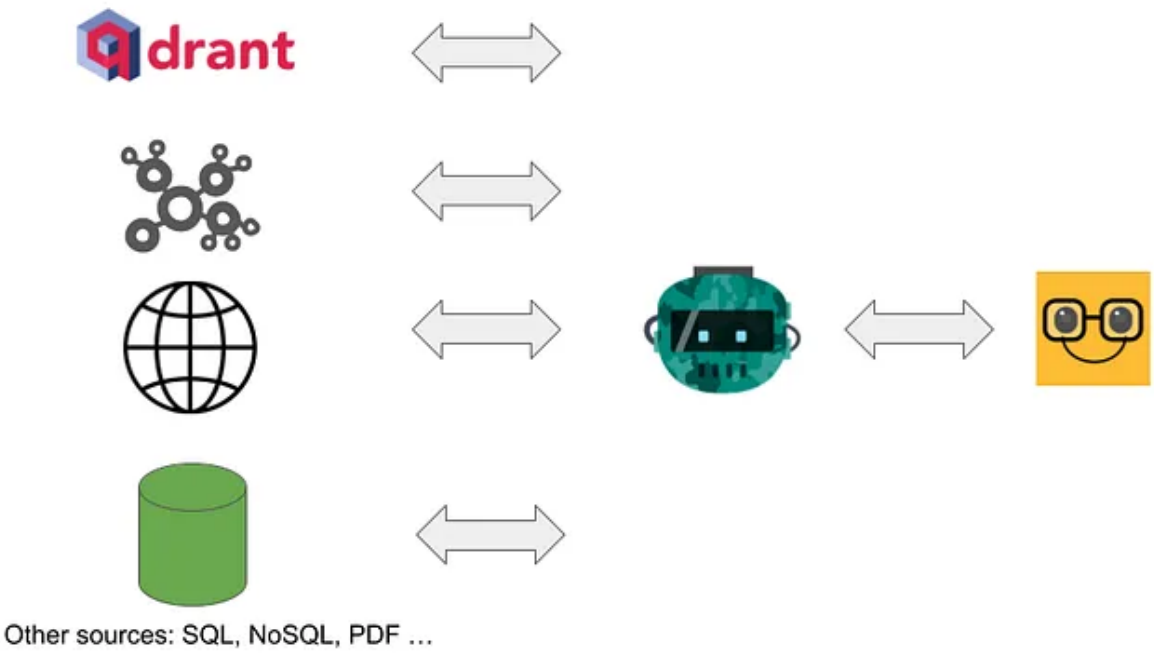


Figure 6. The LangChain architecture in this article can be extended to include many more data sources. Image by author.

There are some clear advantages of using a chatbot as the interface across all the data sources. Users are no longer forced to choose one standard format or platform for the integration. That is, no ETL is required. That cuts costs and saves time. And users can enjoy all the advantages that each data technology brings, such as the fuzzy search of the vector database, or the efficient aggregation provided by SQL. As a result, it will encourage users to try new data technologies (such as graph databases), because they no longer need to worry about ETL. Simply plug the new data source into the chatbot, and it is good to go.

An alternative architecture is data fabric. In that case, a knowledge graph is created as a data access layer. It acts as a giant index and hides all the underlying data

sources. It will fetch answers from the appropriate data sources. And the chatbot only talks to the graph (Figure 5).

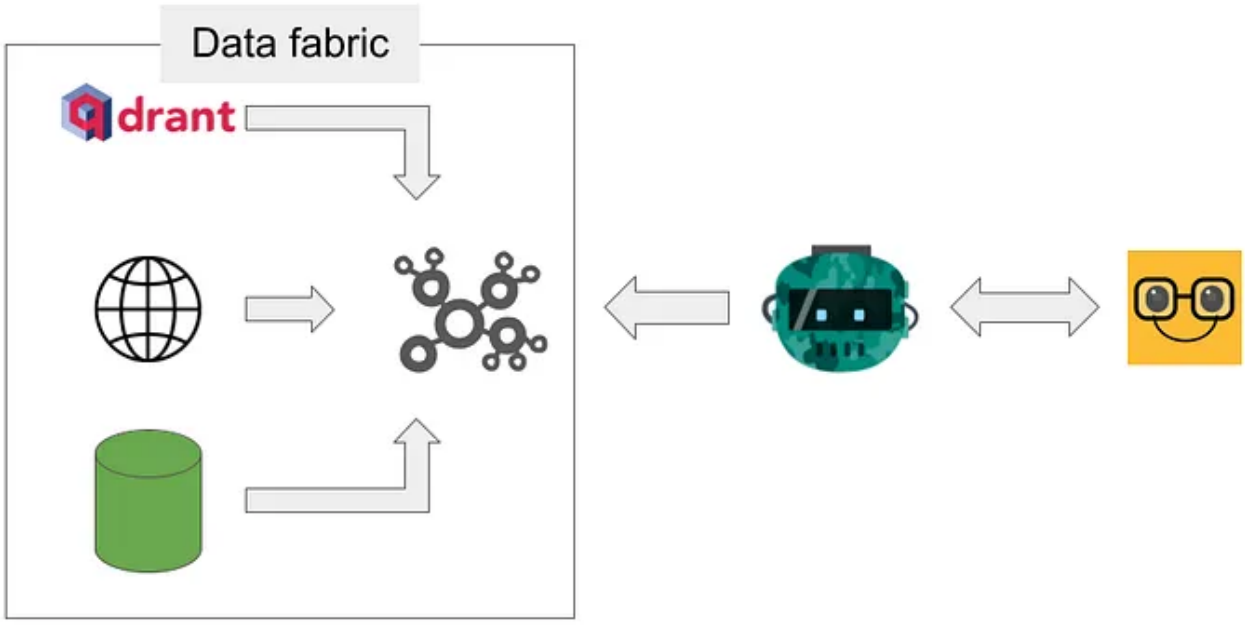


Figure 7. Data fabric is an alternative architecture for a chatbot across many data sources. Image by author.

The key difference between the LangChain architecture and the data fabric is where the data source routing occurs. In the LangChain architecture, LangChain chooses the appropriate data source based on the user inputs or its best judgment. In data fabric, the data addresses are encoded in the graph. There are pros and cons. In my opinion, the LangChain architecture is easier because programmers just need to configure the chatbot. In comparison, data fabric requires the maintenance of a large index graph with many moving parts.

The multiple data sources also make the chatbot more accurate and intelligent. On the one hand, they can build a consensus answer to enhance the trustworthiness of the chatbot. On the other hand, especially with the built-in web search capacity, the chatbot is less likely to answer “I don’t know”.

Certainly, this project is just a proof of concept. It has lots of room for improvement. For example, we can build a streaming pipeline between clinicaltrials.gov and our data sources. Finally, we can integrate data visualization, such as Gemini Data, into the chatbot.

Langchain Neo4j Qdrant Chatbots Gemini Data

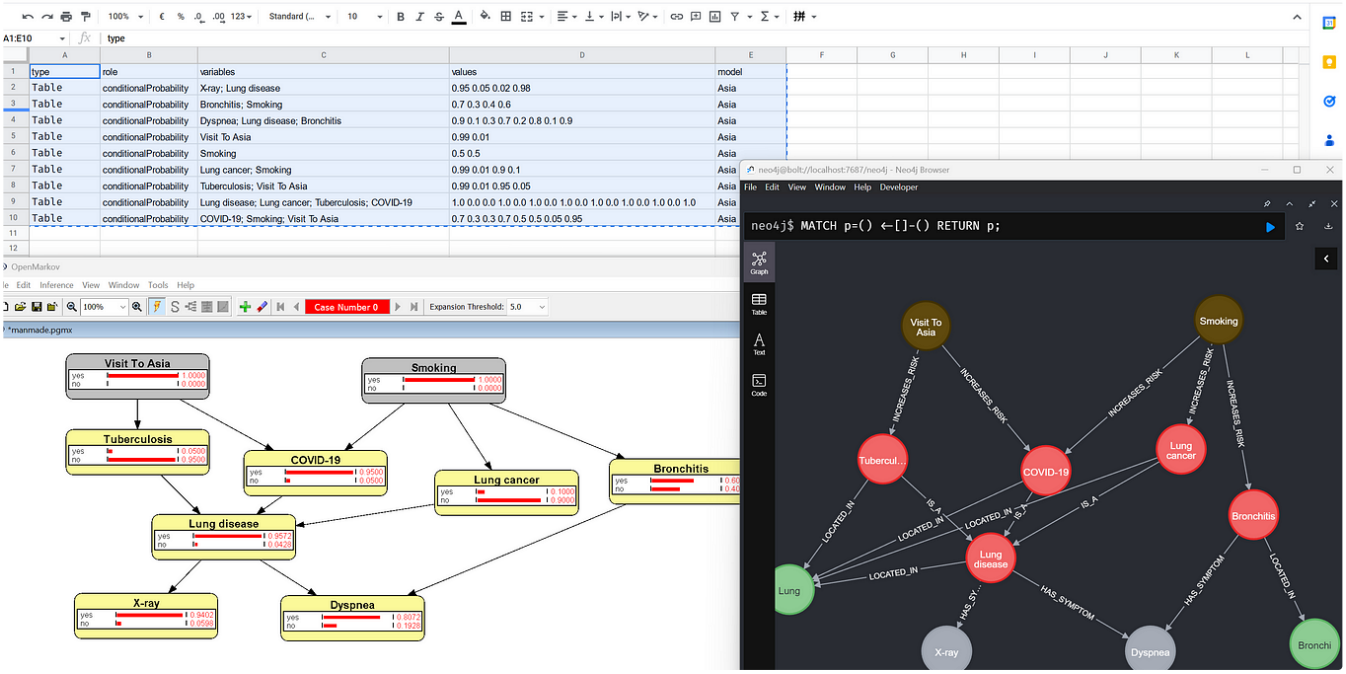
Following

Written by Sixing Huang

924 Followers

A Neo4j Ninja, German bioinformatician. I like to try things: Cloud, ML, satellite imagery, Japanese, plants, and travel the world.

More from Sixing Huang



Sixing Huang in Geek Culture

How to Build a Bayesian Knowledge Graph

Store data in Google Sheets, visualize the knowledge graph in Neo4j, and do Bayesian reasoning with OpenMarkov

· 10 min read · Feb 5

168 2

Sixing Huang

Clinical Trials as Graphs and Vectors

A search engine powered by Neo4j, Gemini Data, and Qdrant

✨ · 9 min read · Jul 10

14

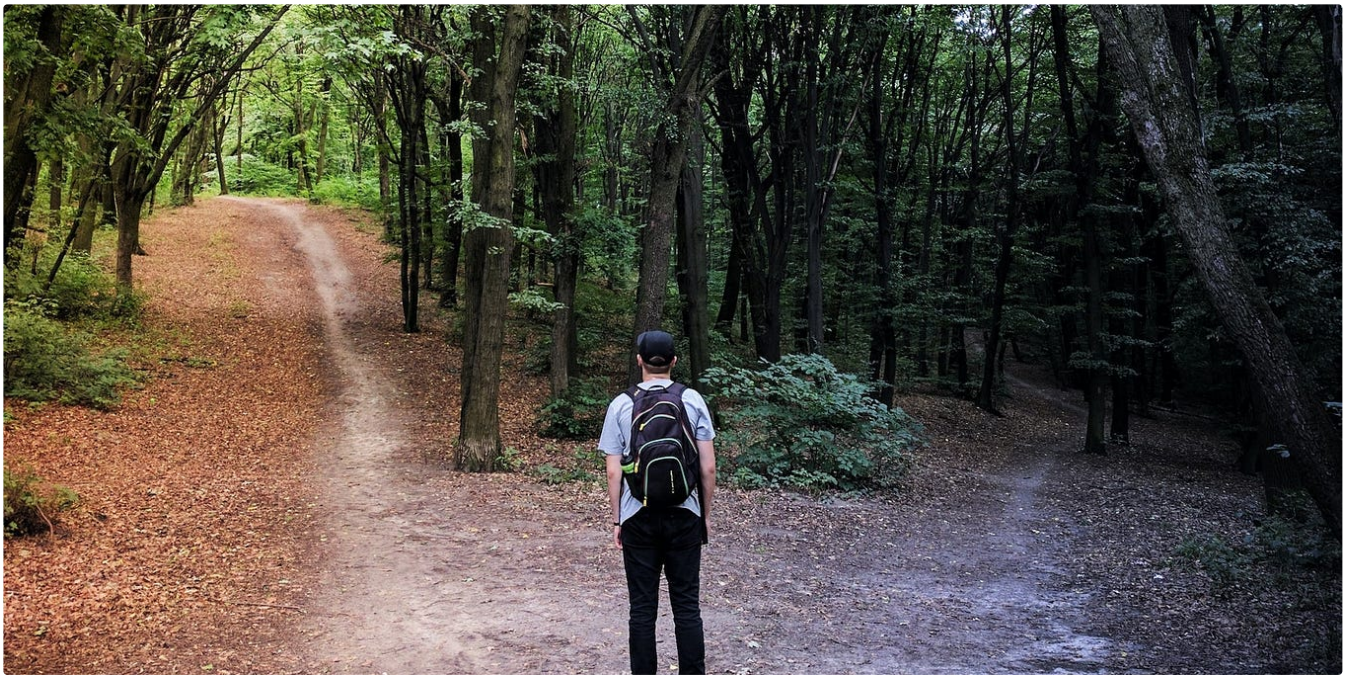
Sixing Huang in CodeX

How to do CI/CD in BitBucket Pipeline

A Simple but Complete Walkthrough with Python

5 min read · Nov 28, 2021

164



Sixing Huang in Geek Culture

Calculate Cost-Benefit in a Bayesian Knowledge Graph

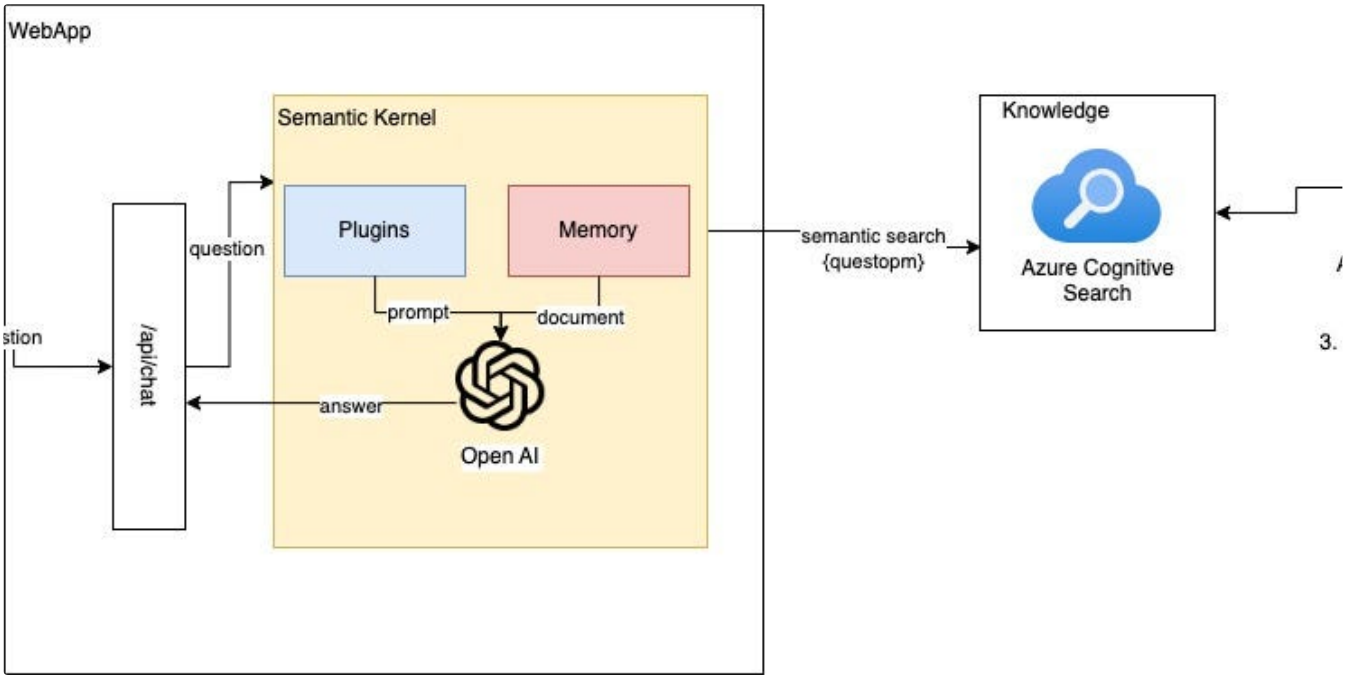
Store data in Google Sheets, view the knowledge graph in Neo4j, and calculate cost-benefit with OpenMarkov

✦ · 13 min read · Mar 28

83 1

See all from Sixing Huang

Recommended from Medium

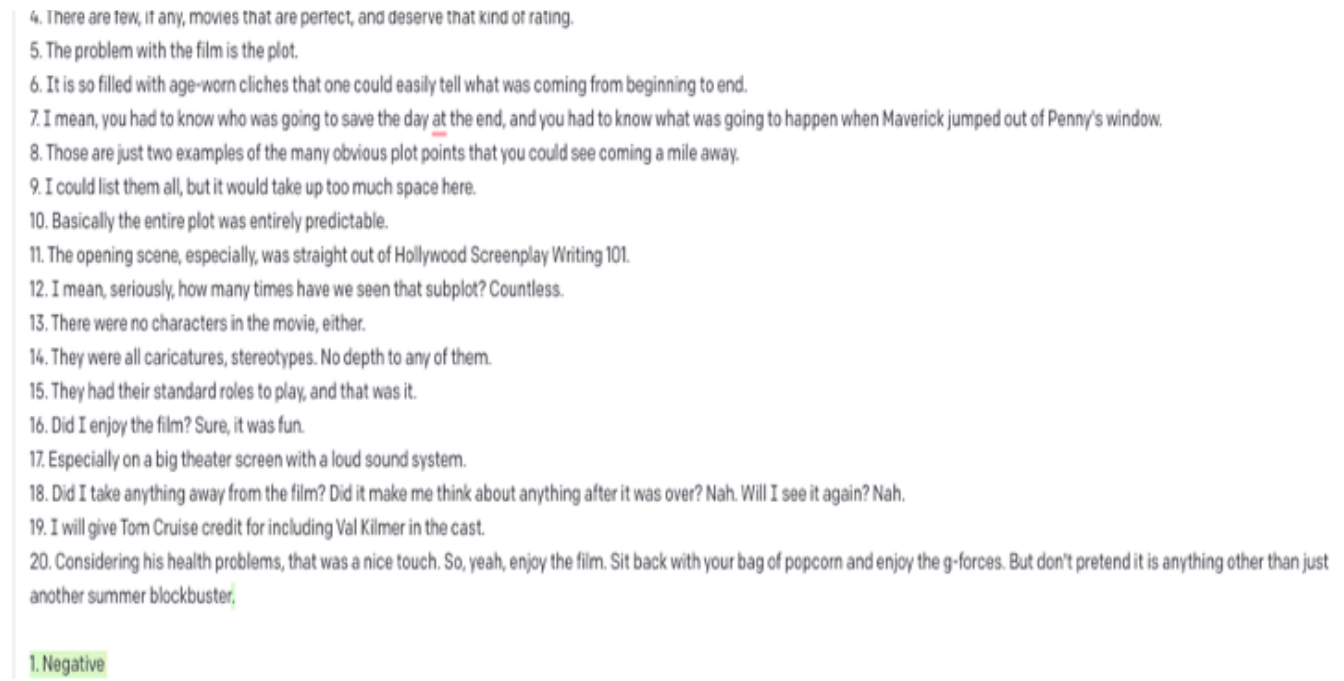


Akshay Kokane

Unleashing the Power of Semantic Kernel and Azure Cognitive Search: A Step-by-Step Guide to...

Transform Your Internal Data into Conversational Brilliance with Cutting-Edge Techniques

5 min read · Jun 30


4. There are few, if any, movies that are perfect, and deserve that kind of rating.
5. The problem with the film is the plot.
6. It is so filled with age-worn cliches that one could easily tell what was coming from beginning to end.
7. I mean, you had to know who was going to save the day at the end, and you had to know what was going to happen when Maverick jumped out of Penny's window.
8. Those are just two examples of the many obvious plot points that you could see coming a mile away.
9. I could list them all, but it would take up too much space here.
10. Basically the entire plot was entirely predictable.
11. The opening scene, especially, was straight out of Hollywood Screenplay Writing 101.
12. I mean, seriously, how many times have we seen that subplot? Countless.
13. There were no characters in the movie, either.
14. They were all caricatures, stereotypes. No depth to any of them.
15. They had their standard roles to play, and that was it.
16. Did I enjoy the film? Sure, it was fun.
17. Especially on a big theater screen with a loud sound system.
18. Did I take anything away from the film? Did it make me think about anything after it was over? Nah. Will I see it again? Nah.
19. I will give Tom Cruise credit for including Val Kilmer in the cast.
20. Considering his health problems, that was a nice touch. So, yeah, enjoy the film. Sit back with your bag of popcorn and enjoy the g-forces. But don't pretend it is anything other than just another summer blockbuster.

1. Negative

Shivam Solanki in Towards Generative AI

OpenAI vs Watson NLP: Sentiment Analysis

Exploring the Pros and Cons of OpenAI and Watson NLP for Sentiment Analysis Tasks

4 min read · Jul 11

40

Lists

Now in AI: Handpicked by Better Programming

260 stories · 54 saves

Natural Language Processing

439 stories · 81 saves

Generative AI Recommended Reading

52 stories · 106 saves

marketing

117 stories · 40 saves

Veronika Smilga in DeepPavlov

Creating Assistants with DeepPavlov Dream. Part 4: Document-based Question Answering with LLMs

Learn how to use or customize Document-based Question Answering distribution of Dream chatbot.

8 min read · Jul 10

3

Ignacio de Gregorio

Microsoft Just Showed us the Future of ChatGPT with LongNet

Let's talk about Billions

· 8 min read · Jul 20

2.2K 33

CUSTOMER DETAILS			
Billing		**Delivery**	
Nick Bert	P:0401 320 816	Nick Bert	M:0401 320 816
134 Barker Street	M:0401 320 816	134 Barker Street	
NEW FARM Queensland 4005	Account#: WW-833332	NEW FARM Queensland 4005	
Australia		Australia	

DESCRIPTION:	QTY:	UNIT PRICE: (INC TAX)	TOTAL: (EX TAX)	TOTAL: (INC TAX)
Adreno Dive Trip - Wreck of the Marietta Dal & Smiths Rock Double Dive Trip (ID: 179285, AD-DiveMariettaSmiths,)	1	$210.00	$190.91	**$210.00**
Scuba Kit Hire (Tank, Weights, BCD, Regulator & Computer) (ID: 172605, AD-SCUBA-SET,)	1	$70.00	$63.64	**$70.00**
Adreno Wetsuit Hire	1	$30.00	$27.27	**$30.00**

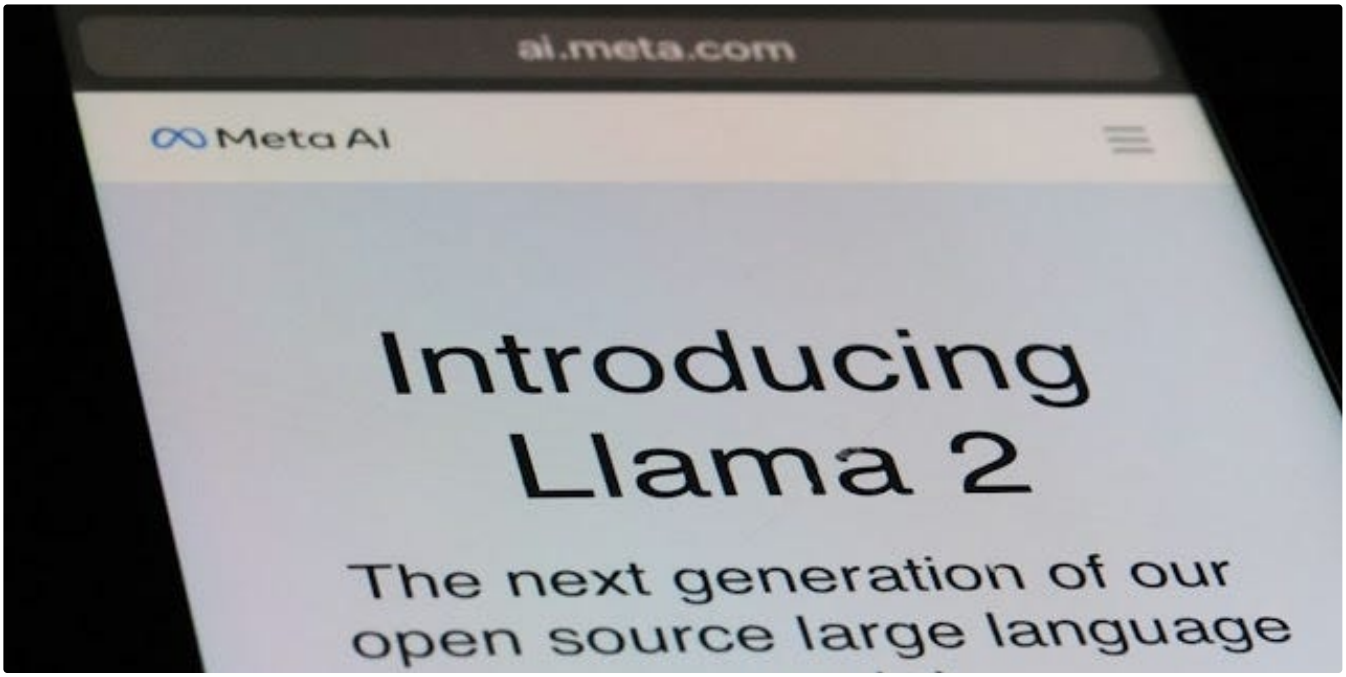
in box

Use of Generative A.I. to achieve better OCR results than traditional tools

As generative AI advances, models like ChatGPT have proven adept at extracting text from unstructured data. These models can grasp the...

3 min read · Apr 12

70



Kh Mohammed

LLaMA 2: how to use Meta's extremely powerful open-source chatbot now

Meta launched LLaMA 2, an extremely powerful open AI language model that challenges its competitors. It encourages researchers to build and...

3 min read · Jul 20

7