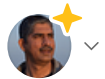


Open in app ↗



Search Medium



A CAMEL ride

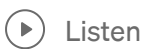
A Story of AI Role-Playing using CAMEL, Langchain and VertexAI



Yogesh Haribhau Kulkarni (PhD)

Published in Google Developer Experts

8 min read · 10 hours ago



Listen



Share



More

In a world where artificial intelligence continues to push the boundaries of human achievement, a groundbreaking framework has emerged to explore the uncharted territories of AI communication. Enter CAMEL, short for “Communicative Agents for ‘Mind’ Exploration of Large Scale Language Model Society.” This innovative concept is set to redefine the way AI agents interact with each other and, in doing so, revolutionize the realm of conversational AI.

A New Frontier in AI Communication

CAMEL introduces a fascinating role-playing agent framework, where three distinct AI entities come together to collaborate seamlessly:

- 1. AI User Agent:** The conductor of the AI symphony, this agent provides instructions to the AI assistant with the ultimate goal of accomplishing a specific task.
- 2. AI Assistant Agent:** This agent serves as the executor, diligently following the AI user's instructions and providing solutions to the assigned task.
- 3. Task-Specifier Agent:** The unsung hero behind the scenes, the task-specifier agent plays a pivotal role in brainstorming and defining specific tasks for the AI user and AI assistant to undertake. This ensures that tasks are concrete and well-defined, sparing the user from the laborious task of detailed task specification.

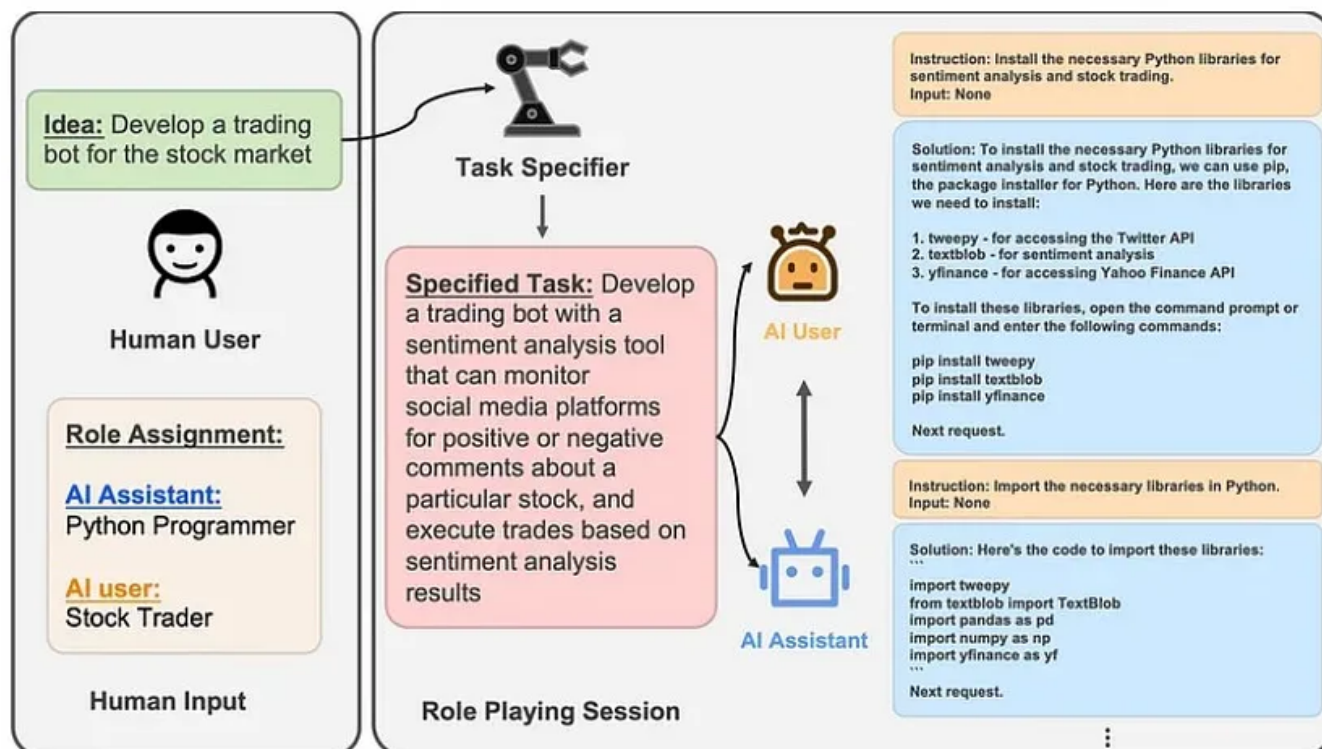


Figure 6. Role-playing framework. Source: <https://arxiv.org/abs/2303.17760>

Putting Theory into Practice

Imagine a scenario where a human trader has a brilliant idea: developing a trading bot. In this scenario, the AI user agent embodies the trader's expertise, while the AI assistant agent represents a Python programmer. It is the task-specifier agent that takes the initiative by formulating a specific task with detailed instructions: "Monitor social media sentiment and trade stocks based on sentiment analysis results."

Here's where the magic happens. The AI user agent transforms into the task planner, orchestrating the grand plan, while the AI assistant agent becomes the task executor, bringing the plan to life. They engage in a fluid dialogue, prompting each other in a continuous loop until predefined termination conditions are met.

Another example (as implemented below) is, say, an Entrepreneur wants to get his tax filing prepared, so he/she is talking to an Accountant to get the task done.

The true essence of CAMEL lies in its meticulously crafted prompts, a concept referred to as "inception prompting." These prompts serve multiple critical purposes:

- Assign roles to agents, ensuring they stay in their designated roles.
- Prevent role flipping, maintaining the integrity of the conversation.

- Prohibit harm and the spread of false information, upholding ethical standards.
- Encourage consistent and meaningful conversation, aligning with human intentions.

CAMEL's reach extends beyond theoretical concepts, as demonstrated in the LangChain implementation. This real-world application employs the prompts outlined in the Camel paper and defines three key agents: `task_specify_agent`, `assistant_agent`, and `user_agent`. The implementation utilizes a while loop to facilitate continuous communication between the assistant and user agents.

Implementation

Let's break down each code block in the provided code and provide explanations for each one:

```
import streamlit as st
from typing import List
from langchain.chat_models import ChatVertexAI
from langchain.prompts.chat import (
    SystemMessagePromptTemplate,
    HumanMessagePromptTemplate,
)
from langchain.schema import (
    AIMessage,
    HumanMessage,
    SystemMessage,
    BaseMessage,
)
```

Here's what each import does:

- `streamlit` is a library for building interactive web applications.
- `typing.List` is imported to specify the data type of variables as lists.
- `ChatVertexAI` is a class imported from the `langchain.chat_models` module, representing an AI chat model.
- `SystemMessagePromptTemplate` and `HumanMessagePromptTemplate` are classes imported from `langchain.prompts.chat`, which are used to create templates for system and human messages.

- `AIMessage`, `HumanMessage`, `SystemMessage`, and `BaseMessage` are classes imported from `langchain.schema`, used to define message formats.

```
class CAMELAgent:

    def __init__(
        self,
        system_message: SystemMessage,
        model: ChatVertexAI,
    ) -> None:
        self.system_message = system_message
        self.model = model
        self.init_messages()

    def reset(self) -> None:
        self.init_messages()
        return self.stored_messages

    def init_messages(self) -> None:
        self.stored_messages = [self.system_message]

    def update_messages(self, message: BaseMessage) -> List[BaseMessage]:
        self.stored_messages.append(message)
        return self.stored_messages

    def step(
        self,
        input_message: HumanMessage,
    ) -> AIMessage:
        messages = self.update_messages(input_message)

        output_message = self.model(messages)
        self.update_messages(output_message)

        return output_message
```

This class represents an agent that facilitates communication between AI models. Here's a breakdown of its methods:

- `__init__`: The constructor initializes the agent with a system message and an AI model. It also calls the `init_messages` method to initialize stored messages.
- `reset`: This method resets the stored messages by calling `init_messages` and returns the stored messages.

- `init_messages` : Initializes the stored messages list with the system message.
- `update_messages` : Appends a message to the stored messages list and returns the updated list.
- `step` : This method simulates a step in the conversation. It takes a human message as input, updates the stored messages, passes them to the AI model, and returns the AI's response.

```
st.title("CAMEL-Langchain-VertexAI Agent")
st.sidebar.header("Input Settings")

assistant_role_name = st.sidebar.text_input("Assistant Role Name", "Accountant")
user_role_name = st.sidebar.text_input("User Role Name", "Entrepreneur")
task = st.sidebar.text_area("Task", "Preparing and filing tax returns.")
word_limit = st.sidebar.number_input("Word Limit for Task Brainstorming", value=
```

This code block sets up the Streamlit user interface for the CAMEL agent. It sets the title and creates a sidebar header for input settings. User input fields are created using Streamlit for setting up the role-play scenario. Users can input the names of the Assistant and User roles, a task description, and a word limit for brainstorming.

At the heart of CAMEL are the following prompts/specifications:

- Task Specific Agent

```
task_specifier_sys_msg = SystemMessage(content="You can make a task more specific")
task_specifier_prompt = (
    """Here is a task that {assistant_role_name} will help {user_role_name} to
    Please make it more specific. Be creative and imaginative.
    Please reply with the specified task in {word_limit} words or less. Do not add
    )
task_specifier_template = HumanMessagePromptTemplate.from_template(template=task_specifier_prompt)
task_specify_agent = CAMELAgent(task_specifier_sys_msg, ChatVertexAI(temperature=0.7))
task_specifier_msg = task_specifier_template.format_messages(assistant_role_name=assistant_role_name,
                                                             user_role_name=user_role_name,
                                                             task=task, word_limit=word_limit)
specified_task_msg = task_specify_agent.step(task_specifier_msg)
```

```
print(f"Specified task: {specified_task_msg.content}")
specified_task = specified_task_msg.content
```

• Inception prompts for Agent

```
assistant_inception_prompt = (
    """Never forget you are a {assistant_role_name} and I am a {user_role_name}
    We share a common interest in collaborating to successfully complete a task.
    You must help me to complete the task.
    Here is the task: {task}. Never forget our task!
    I must instruct you based on your expertise and my needs to complete the task.

    I must give you one instruction at a time.
    You must write a specific solution that appropriately completes the requested i
    You must decline my instruction honestly if you cannot perform the instruction
    Do not add anything else other than your solution to my instruction.
    You are never supposed to ask me any questions you only answer questions.
    You are never supposed to reply with a flake solution. Explain your solutions.
    Your solution must be declarative sentences and simple present tense.
    Unless I say the task is completed, you should always start with:

    Solution: <YOUR_SOLUTION>

    <YOUR_SOLUTION> should be specific and provide preferable implementations and e
    Always end <YOUR_SOLUTION> with: Next request."""
)

user_inception_prompt = (
    """Never forget you are a {user_role_name} and I am a {assistant_role_name}
    We share a common interest in collaborating to successfully complete a task.
    I must help you to complete the task.
    Here is the task: {task}. Never forget our task!
    You must instruct me based on my expertise and your needs to complete the task

    1. Instruct with a necessary input:
    Instruction: <YOUR_INSTRUCTION>
    Input: <YOUR_INPUT>

    2. Instruct without any input:
    Instruction: <YOUR_INSTRUCTION>
    Input: None

    The "Instruction" describes a task or question. The paired "Input" provides fur

    You must give me one instruction at a time.
    I must write a response that appropriately completes the requested instruction.
    I must decline your instruction honestly if I cannot perform the instruction du
```

```

You should instruct me not ask me questions.
Now you must start to instruct me using the two ways described above.
Do not add anything else other than your instruction and the optional correspon
Keep giving me instructions and necessary inputs until you think the task is co
When the task is completed, you must only reply with a single word <CAMEL_TASK_
Never say <CAMEL_TASK_DONE> unless my responses have solved your task.""
)

```

- Create System Messages

```

# Create a helper to get system messages for AI assistant and AI user
# from role names and the task

def get_sys_msgs(assistant_role_name: str, user_role_name: str, task: str):
    assistant_sys_template = SystemMessagePromptTemplate.from_template(template=assistant_sys_template)
    assistant_sys_msg = \
        assistant_sys_template.format_messages(assistant_role_name=assistant_role_name,
                                              user_role_name=user_role_name,
                                              task=task)[0]

    user_sys_template = SystemMessagePromptTemplate.from_template(template=user_sys_template)
    user_sys_msg = \
        user_sys_template.format_messages(assistant_role_name=assistant_role_name,
                                          user_role_name=user_role_name,
                                          task=task)[0]

    return assistant_sys_msg, user_sys_msg

```

- Create Agents

```

# Create AI assistant agent and AI user agent from obtained system messages
assistant_sys_msg, user_sys_msg = get_sys_msgs(assistant_role_name, user_role_name, task)
assistant_agent = CAMELAgent(assistant_sys_msg, ChatVertexAI(temperature=0.2))
user_agent = CAMELAgent(user_sys_msg, ChatVertexAI(temperature=0.2))

```

- Initialize as

```

# Reset agents
assistant_agent.reset()
user_agent.reset()

# Initialize chats
assistant_msg = HumanMessage(
    content=(f"{user_sys_msg.content}. "
             "Now start to give me introductions one by one. "
             "Only reply with Instruction and Input.))

user_msg = HumanMessage(content=f"{assistant_sys_msg.content}")
user_msg = assistant_agent.step(user_msg)

```

- Role playing starts as

```

st.header("Conversation")

chat_turn_limit, n = 5, 0
while n < chat_turn_limit:
    n += 1
    user_ai_msg = user_agent.step(assistant_msg)
    user_msg = HumanMessage(content=user_ai_msg.content)
    # print(f"AI User ({user_role_name}): \n\n{user_msg.content}\n\n")

    assistant_ai_msg = assistant_agent.step(user_msg)
    assistant_msg = HumanMessage(content=assistant_ai_msg.content)
    # print(f"AI Assistant ({assistant_role_name}): \n\n{assistant_msg.content}\n\n")

    # Display the conversation in chat format
    st.text(f"AI User ({user_role_name}):")
    st.info(user_msg.content)
    st.text(f"AI Assistant ({assistant_role_name}):")
    st.success(assistant_msg.content)
    if "<CAMEL_TASK_DONE>" in user_msg.content:
        break

```

Here is one sample session with Entrepreneur and Accountant role for Preparing and filing Tax Returns task.

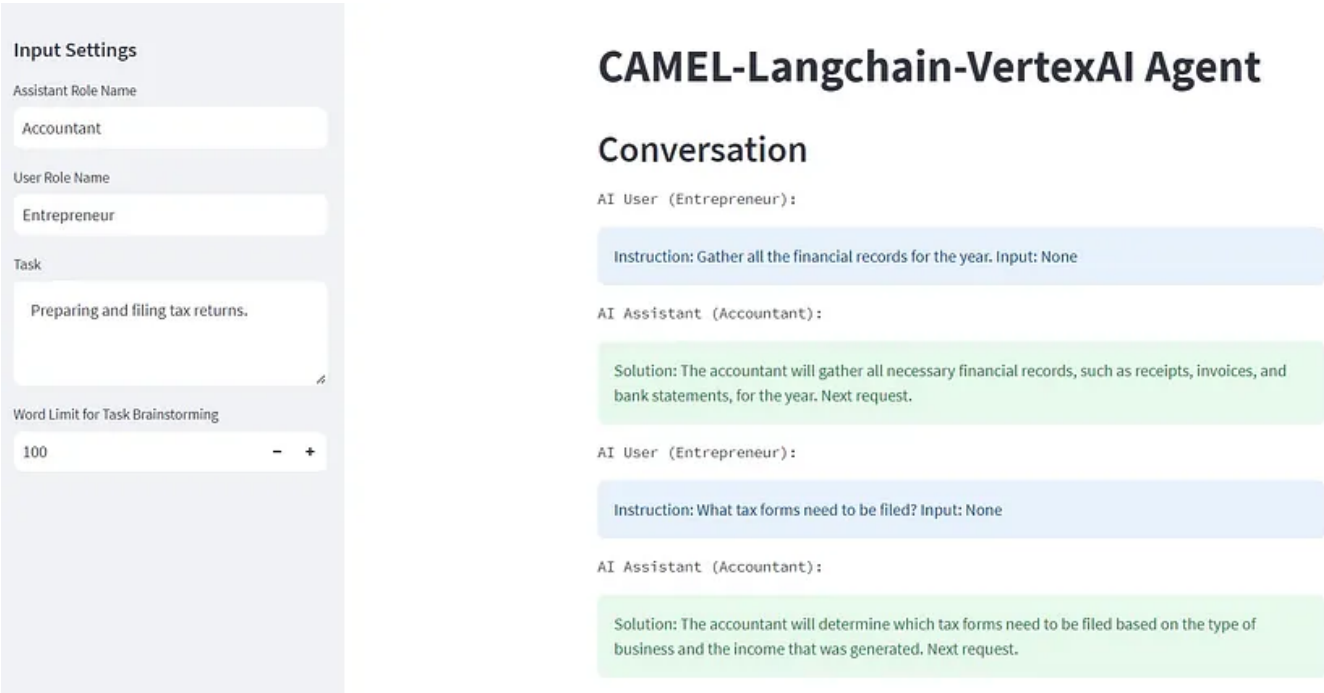


Image from code by Author

CAMEL isn’t just a concept; it’s a doorway to a new era of AI exploration. This role-playing framework empowers AI agents to communicate, collaborate, and learn like never before. It provides a unique window into the “cognitive” processes of AI, paving the way for innovative research and development.

Join the CAMEL-AI.org open-source community on a journey to unlock the minds of AI. Discover the behaviors, capabilities, and potential risks of autonomous and communicative agents. Through shared knowledge and collaboration, we’re poised to shape the future of AI in ways we’ve only dared to dream.

Explore CAMEL further at [Project website](#) and dive into the nitty-gritty details in the [Arxiv paper](#). The open-source implementation can be found [here](#).

In the world of AI, the possibilities are boundless, and with CAMEL, we’re venturing into uncharted territory. Welcome to the future of AI communication and cooperation.

References

<p>CAMEL Role-Playing Autonomous Cooperative Agents 🐦 🔗</p> <p>Langchain</p> <p>This is a langchain implementation of paper Communicative Agents for "Mind" Exploration of Large Scale Language Model...</p> <p>python.langchain.com</p>	
--	--

CAMEL-AI

<https://github.com/camel-ai/camel>

www.camel-ai.org

GitHub - camel-ai/camel: 🐪 CAMEL: Communicative Agents for "Mind" Exploration of Large Scale...

🐪 CAMEL: Communicative Agents for "Mind" Exploration of Large Scale Language Model Society - GitHub - camel-ai/camel...

github.com

4 Autonomous AI Agents you need to know

“Westworld” simulation, Camel, BabyAGI, AutoGPT 🌟 with the power of LangChain 🌟

towardsdatascience.com

Google Colaboratory

Edit description

colab.research.google.com

- Langchain
- Vertex AI
- Streamlit
- Artificial Intelligence
- Future



Edit profile

Written by Yogesh Haribhau Kulkarni (PhD)

967 Followers · Writer for Google Developer Experts

PhD in Geometric Modeling | Google Developer Expert (Machine Learning) | Top Writer 3x (Medium) | More at <https://www.linkedin.com/in/yogeshkulkarni/>

More from Yogesh Haribhau Kulkarni (PhD) and Google Developer Experts





Yogesh Haribhau Kulkarni (PhD) in Google Cloud - Community

Building a GST FAQs App

with Streamlit, Langchain, HuggingFace and VertexAI Palm APIs

5 min read · Aug 5



47



1



Alexey Inkin in Google Developer Experts

How I became a Google Developer Expert in Flutter

All my steps from not knowing of Flutter to being a member of the most privileged community for developers in the world.

6 min read · Jul 29



331



5





Package-based Architecture



Abhishek Doshi in Google Developer Experts

Package-based architecture—Let's deliver the packages 📦

We have all seen a lot of architectures like MVVM, MVC, Clean Architecture, etc. But do you know, there's also a hidden gem in Flutter...

3 min read · Aug 27



125



3



Yogesh Haribhau Kulkarni (PhD) in ILLUMINATION Videos and Podcasts

The Equation of True Happiness

Based on talks by Arthur C Brooks

4 min read · Sep 17

 75



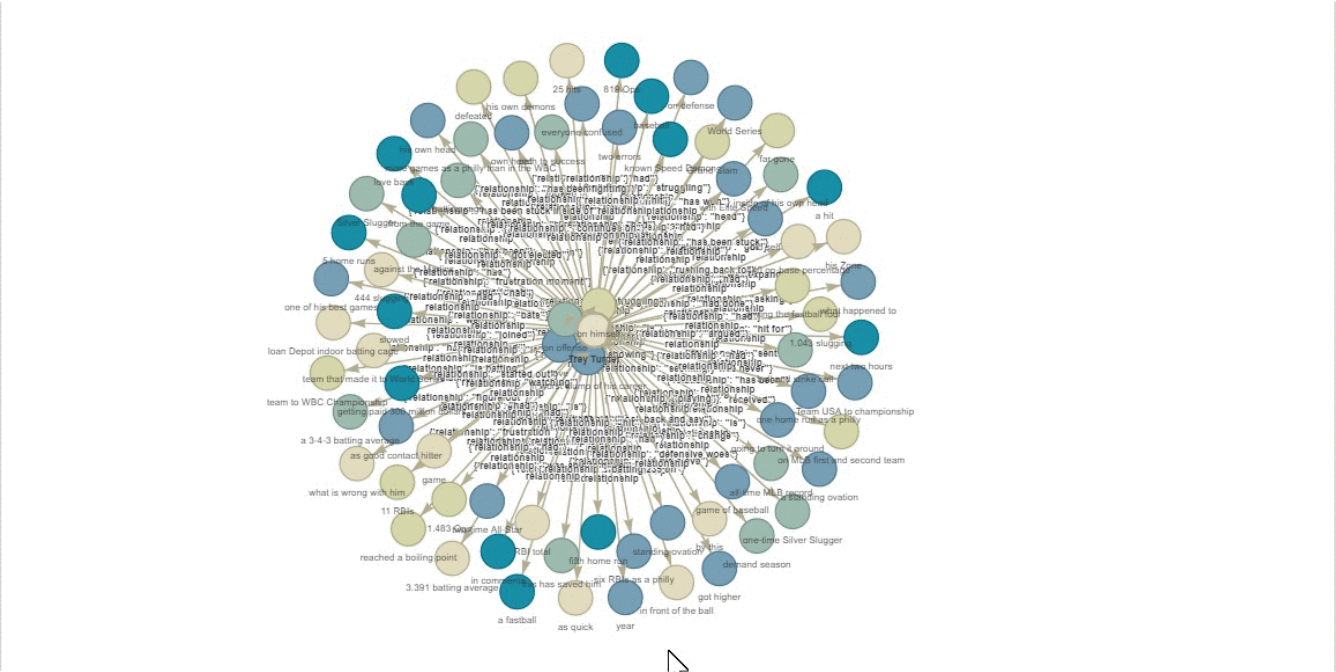




See all from Yogesh Haribhau Kulkarni (PhD)

See all from Google Developer Experts

Recommended from Medium

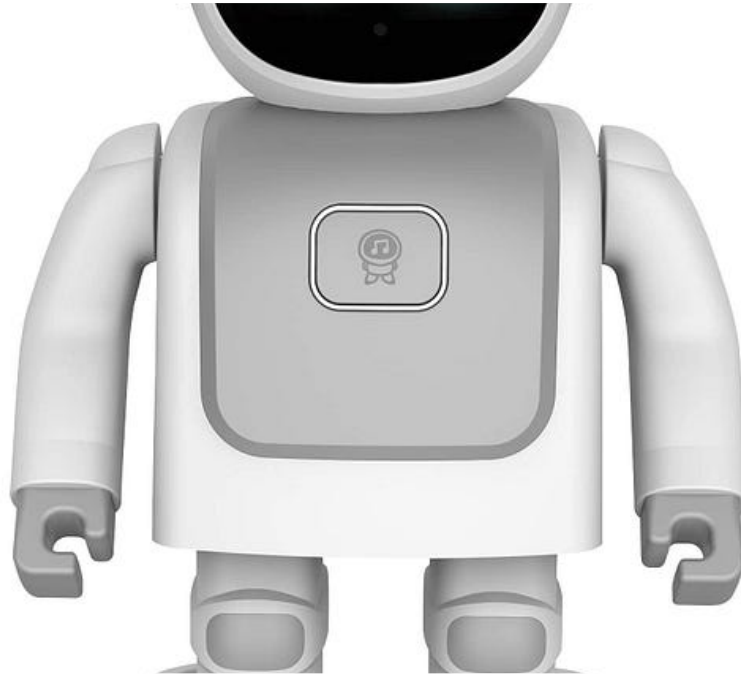


 Wenqi Glantz in Better Programming

7 Query Strategies for Navigating Knowledge Graphs With LlamaIndex

Exploring NebulaGraph RAG Pipeline with the Philadelphia Phillies

★ · 17 min read · 3 days ago



Manoranjan Rajguru

Building a Task Execution Engine with LLM

In an era of rapid digital transformation, automation and efficiency are key drivers of success. Whether you're managing complex workflows...

6 min read · Sep 23



Lists



AI Regulation

6 stories · 137 saves



ChatGPT prompts

24 stories · 449 saves



ChatGPT

21 stories · 177 saves



Generative AI Recommended Reading

52 stories · 271 saves

 Damian Gil in Towards Data Science

Mastering Customer Segmentation with LLM

Unlock advanced customer segmentation techniques using LLMs, and improve your clustering models with advanced techniques

23 min read · 6 days ago

 1.8K

 18






 Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's attention. The transformer based large language models...

15 min read · Sep 14



 Ryan Nguyen in Towards AI

So, You Want To Improve Your RAG Pipeline

Ways to go from prototype to production with LlamaIndex

★ · 9 min read · 6 days ago





MIT IDE in MIT Initiative on the Digital Economy

How to Lead, Not Lag, in Business AI

Researchers say productivity gains will be the rewards of generative AI—but only if firms take actions now.

4 min read · 6 days ago



213



4



See more recommendations