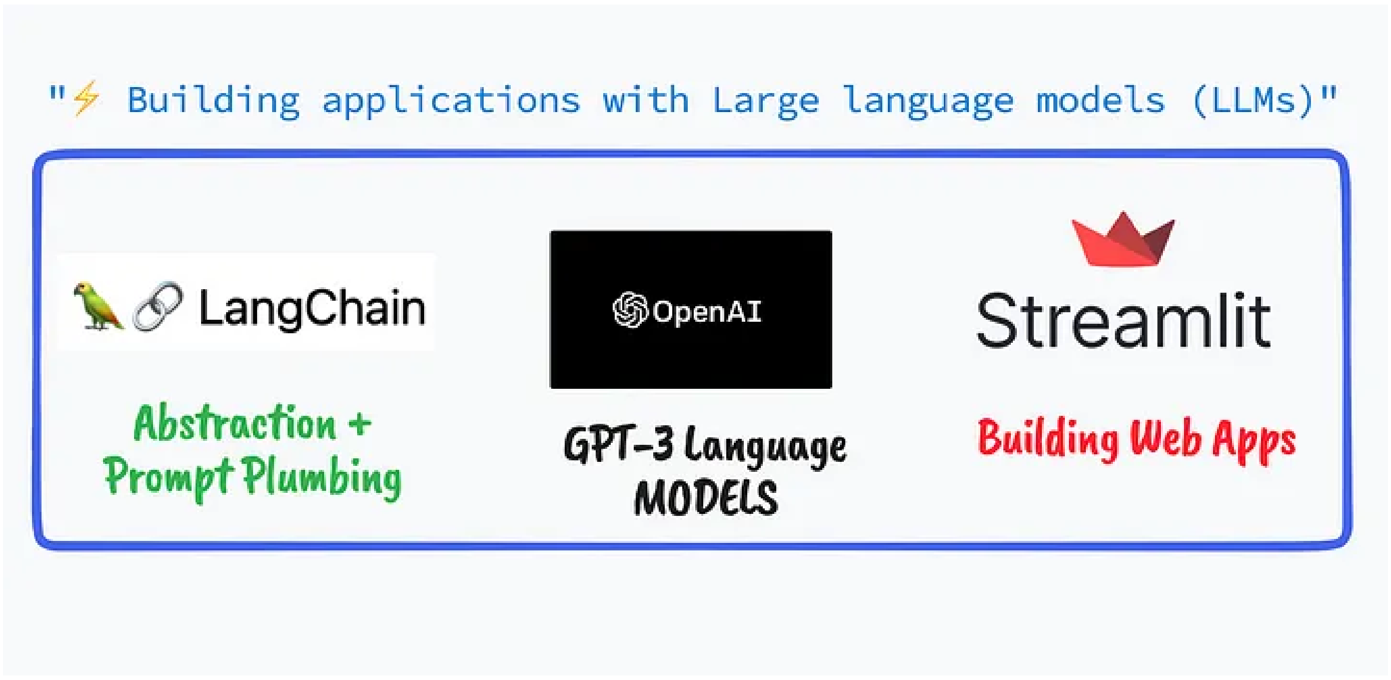
Getting started with LangChain — A powerful tool for working with Large Language Models

Avra · Follow
Published in Databutton
8 min read · Feb 25

Listen Share

Building a Web Application using OpenAI GPT3 Language model and LangChain's SimpleSequentialChain within a Streamlit front-end

🧑🏾‍💻 GitHub ⭐️| 🐦 Twitter | 📹 YouTube | ☕️ BuyMeaCoffee | Ko-fi💜


"⚡ Building applications with Large language models (LLMs)"
🦜🔗 LangChain
Abstraction + Prompt Plumbing
OpenAI
GPT-3 Language MODELS
Streamlit
Building Web Apps

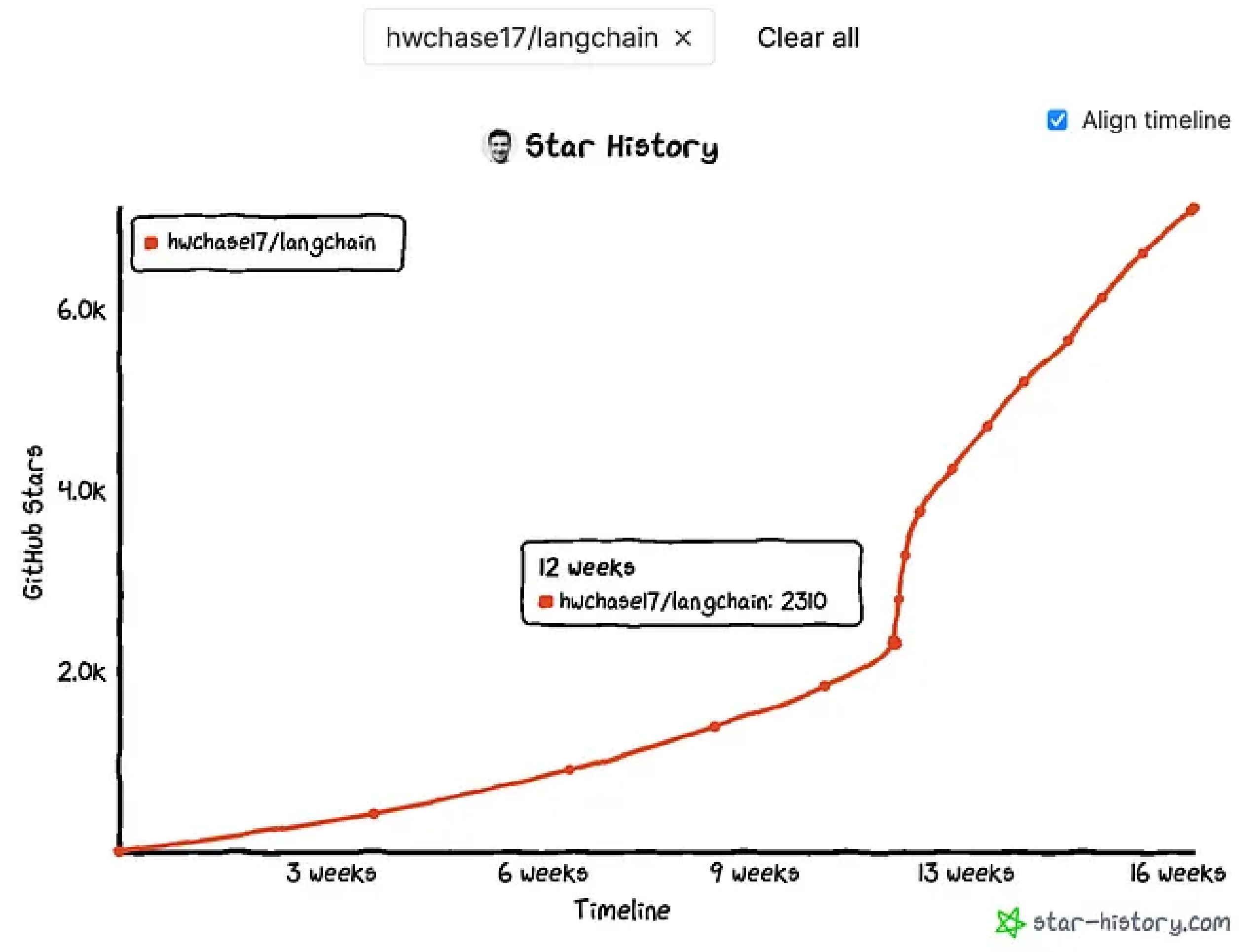
You can refer to the video as well, which will walk you through step-by-step,

Bonus : The tutorial video also showcases how we can build this entire web application within a single platform called databutton (No! they aren't sponsoring me , just loved using their product …) — where you don't even need to push to GitHub or set up your own environment to work with .

Introduction

What is LangChain?

LangChain is a powerful tool that can be used to work with Large Language Models (LLMs). LLMs are very general in nature, which means that while they can perform many tasks effectively, they may not be able to provide specific answers to questions or tasks that require deep domain knowledge or expertise. For example, imagine you want to use an LLM to answer questions about a specific field, like medicine or law. While the LLM may be able to answer general questions about the field, it may not be able to provide more detailed or nuanced answers that require specialized knowledge or expertise.

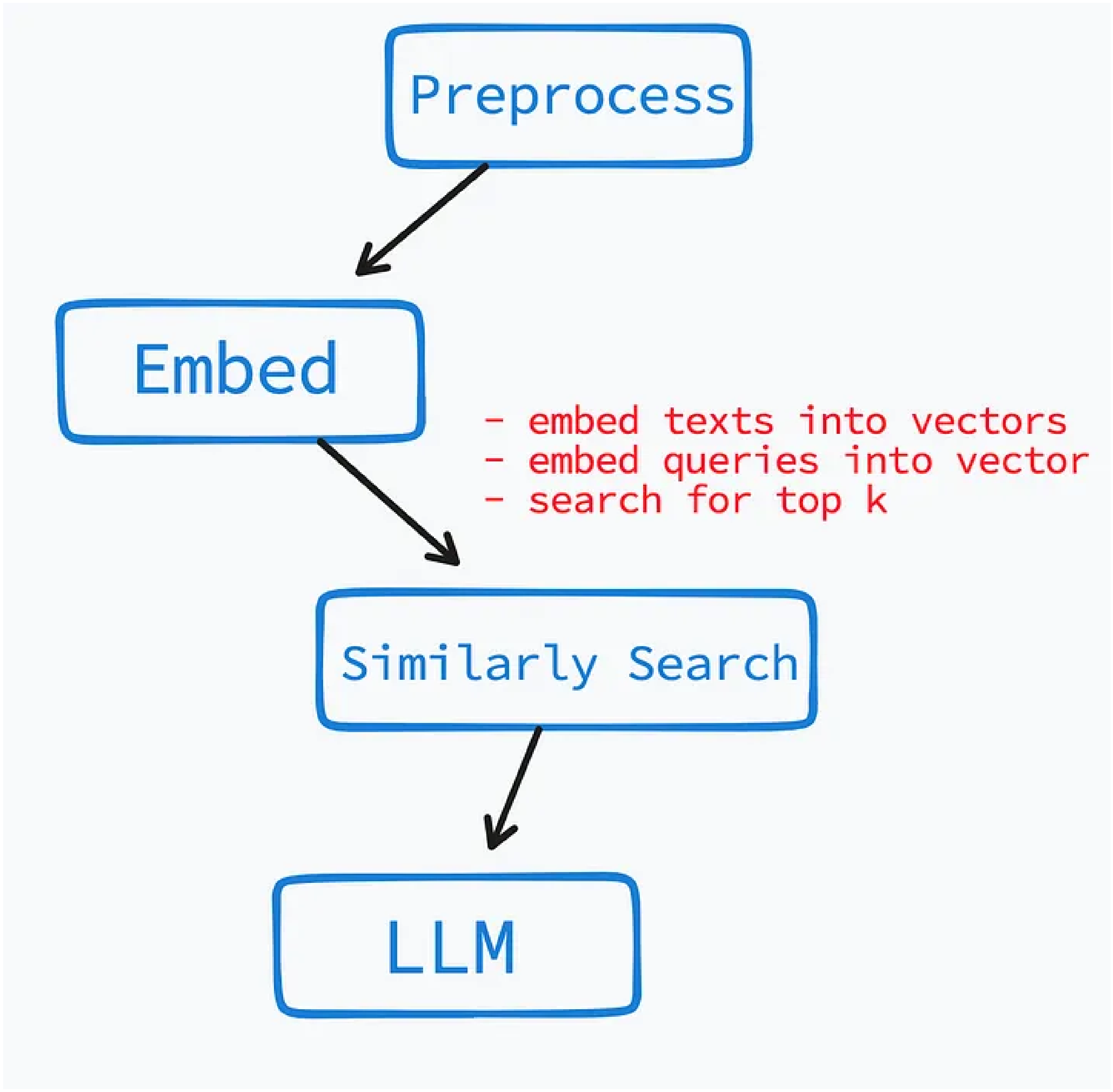


The growth of LangChain has been pretty quick, undoubtedly impressive!

Why we need LangChain?

To work around this limitation, LangChain offers a useful approach where the corpus of text is preprocessed by breaking it down into chunks or summaries, embedding them in a vector space, and searching for similar chunks when a question is asked. This pattern of preprocessing, real-time collecting, and interaction with the LLM is common and can be used in other scenarios as well, such as code and semantic search. LangChain provides an abstraction that simplifies the process of composing these pieces. This "prompt plumbing" is crucial and will become increasingly important as LLMs become more powerful and require more data to be provided at prompt time.

You can read more about general use-cases of LangChain over their documentation or their GitHub repo. Highly recommended to have broader perspective about this package.

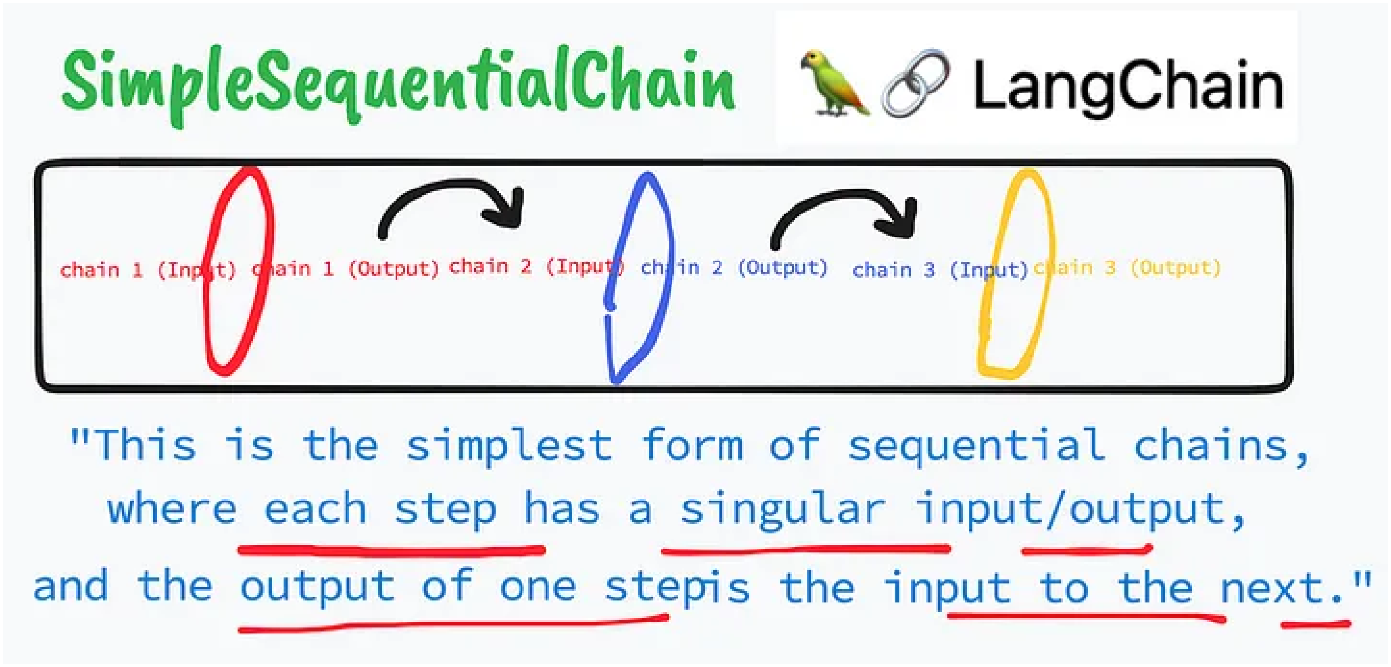


A general sketchy workflow while working with Large Language Models.

Attributes of LangChain (related to this blog post)

As the name suggests, one of the most powerful attributes (among many others!) which LangChain provides is to create Chains. Chains are an important feature of LangChain enable users to combine multiple components together to create a single, coherent application. One example of this is creating a chain that takes user input, formats it using a PromptTemplate, and then passes the formatted response to a Large Language Model (LLM) for processing.

In this blog post, we will primarily look into the abstraction capabilities of Sequential chains. Sequential chains, as the name suggests, execute their links in a sequential or step-by-step order. In other words, the output of one link is passed as the input to the next link in the chain. This means that the output of the first LLM becomes the input to the second LLM and so on, until the final output is generated. This approach allows users to create more complex and sophisticated models by combining the strengths of multiple LLMs.


SimpleSequentialChain
LangChain
chain 1 (Input)
chain 1 (Output)
chain 2 (Input)
chain 2 (Output)
chain 3 (Input)
chain 3 (Output)
"This is the simplest form of sequential chains, where each step has a singular input/output, and the output of one stepis the input to the next."

Visualizing Sequential Chain

Building a demo Web App with LangChain + OpenAI + Streamlit

Let's now try to implement this idea of LangChain in a real use-case and I'm certain that would help us to have a quick grasp !

But before! We need to install few dependencies — as we are going to build a Streamlit Web App using LangChain and OpenAI GPT-3 model.

```
# Dependencies to install

pip install streamlit
pip install langchain
pip install openai
```

- **Streamlit** is a popular Python library for building data science web apps
- **OpenAI** provides access to OpenAI's GPT-3 language model.

To know more about Streamlit — Open AI integration , make sure to check my other blog posts or video tutorials. How to obtain Open AI API keys ? Check this blog post!

Build the Web App

1. Importing Libraries

Here, we start with importing the necessary packages. We also import three classes from the `langchain` package: `LLMChain`, `SimpleSequentialChain`, and `PromptTemplate`. These classes are used to define and run our language model chains.

```
import streamlit as st # import the Streamlit library
from langchain.chains import LLMChain, SimpleSequentialChain # import LangChain l
from langchain.llms import OpenAI # import OpenAI model
from langchain.prompts import PromptTemplate # import PromptTemplate
```

2. Basic set up of the app (Header, subheader etc ...)

We set up the app, with few relevant informations, using simple Streamlit syntax,

```
# Set the title of the Streamlit app
st.title("✅ What's TRUE  : Using LangChain `SimpleSequentialChain`")

# Add a link to the Github repository that inspired this app
st.markdown("Inspired from [fact-checker](https://github.com/jagilley/fact-checke
```

3. Input widgets to interact with Front-end user (API KEY, Question widget ...)

The app also allows the user to enter their OpenAI API key, which will be used to access OpenAI's language model.

```
# If an API key has been provided, create an OpenAI language model instance
if API:
    llm = OpenAI(temperature=0.7, openai_api_key=API)
else:
    # If an API key hasn't been provided, display a warning message
    st.warning("Enter your OPENAI API-KEY. Get your OpenAI API key from [here](ht
```

Further we need to provide an input widget, which will allow our user to enter any questions .

```
# Add a text input box for the user's question
user_question = st.text_input(
    "Enter Your Question : ",
    placeholder = "Cyanobacteria can perform photosynthetsis , are they considere
)
```

3. The CHAINS are in work

Now , in order to generate a valid response to our user-end's question, we pass the questions, through couple of `SimpleSequentialChain` pipeline — as soon as the button `Tell me about it` is clicked!

```
# Generating the final answer to the user's question using all the chains
if st.button("Tell me about it", type="primary"):
    # Chain 1: Generating a rephrased version of the user's question
    template = """{question}\n\n"""
    prompt_template = PromptTemplate(input_variables=["question"], template=templa
    question_chain = LLMChain(llm=llm, prompt=prompt_template)

    # Chain 2: Generating assumptions made in the statement
    template = """Here is a statement:
        {statement}
        Make a bullet point list of the assumptions you made when producing the a
    prompt_template = PromptTemplate(input_variables=["statement"], template=temp
    assumptions_chain = LLMChain(llm=llm, prompt=prompt_template)
    assumptions_chain_seq = SimpleSequentialChain(
        chains=[question_chain, assumptions_chain], verbose=True
    )

    # Chain 3: Fact checking the assumptions
    template = """Here is a bullet point list of assertions:
    {assertions}
    For each assertion, determine whether it is true or false. If it is false, ex
    prompt_template = PromptTemplate(input_variables=["assertions"], template=tem
    fact_checker_chain = LLMChain(llm=llm, prompt=prompt_template)
    fact_checker_chain_seq = SimpleSequentialChain(
        chains=[question_chain, assumptions_chain, fact_checker_chain], verbose=T
    )

    # Final Chain: Generating the final answer to the user's question based on the
    template = """In light of the above facts, how would you answer the question
        user_question
    )
    template = """{facts}\n""" + template
    prompt_template = PromptTemplate(input_variables=["facts"], template=template
    answer_chain = LLMChain(llm=llm, prompt=prompt_template)
    overall_chain = SimpleSequentialChain(
        chains=[question_chain, assumptions_chain, fact_checker_chain, answer_cha
        verbose=True,
    )

    # Running all the chains on the user's question and displaying the final answe
    st.success(overall_chain.run(user_question))
```

The `SimpleSequentialChain` combines several chains of operations to run a pipeline. Here, we are using four different chains to build the fact-checker pipeline :

1. `question_chain` : This chain takes the user's question as input and returns it as output. This is the starting point for our pipeline. The template for this chain is just the user's question.

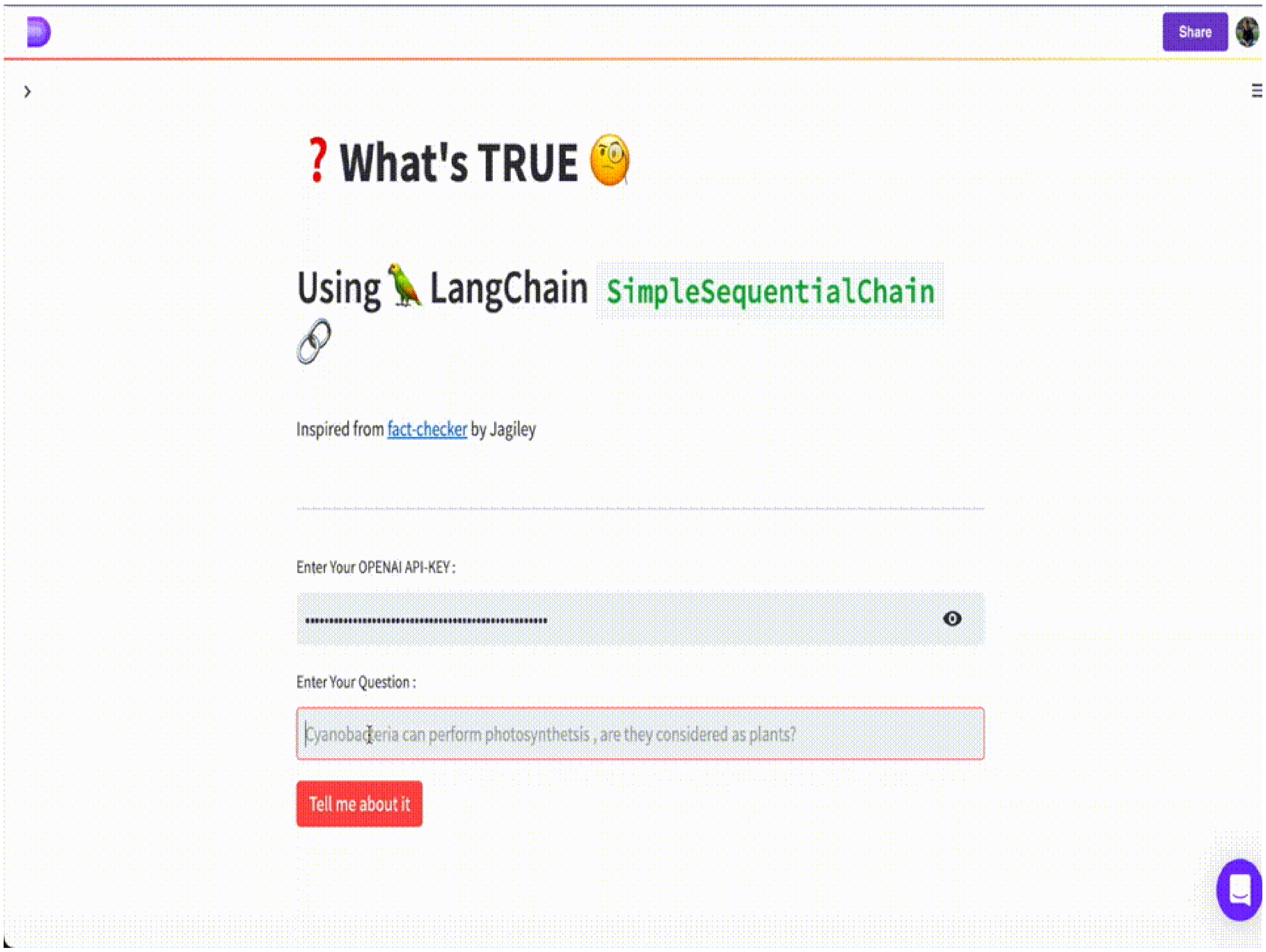
2. `assumptions_chain` : This chain takes the output from `question_chain` as input and produces a bullet-point list of assumptions based on a statement related to the question. The statement is generated using LangChain's `LLMChain` and the `OpenAI` model. The template for this chain asks the user to make a bullet-point list of the assumptions made when producing the statement.

3. `fact_checker_chain`: This chain takes the outputs from `question_chain` and `assumptions_chain` as inputs and produces a bullet-point list of assertions based on the question and assumptions. LangChain's `LLMChain` and the `OpenAI` model are used to generate the assertions. The template for this chain asks the user to determine whether each assertion is true or false, and to explain why if it is false.

4. `answer_chain`: This chain takes the outputs from `question_chain`, `assumptions_chain`, and `fact_checker_chain` as inputs and produces an answer to the user's question based on the facts generated by the previous chains. The template for this chain asks the user to answer the original question based on the generated facts.

Finally, we combine these chains into the `overall_chain`. The result is the answer to the user's question based on the facts generated by the previous chains.

Conclusion



Inspired from fact-checker by Jagiley

Enter Your OPENAI API-KEY :

Enter Your Question :

Cyanobacteria can perform photosynthetsis , are they considered as plants?

How the app works in Live ? Here's a quick demo.

We have successfully used the `SimpleSequentialChain` module that LangChain provides us and build a very simple yet significant use case to demostrate the abstraction ability of LangChain in form of `Chains` ⛓️ .

You can find the live app here.

👨🏾‍💻 GitHub ⭐️| 🐦 Twitter | 📹 YouTube | ☕️ BuyMeaCoffee | Ko-fi💜

Hi there ! I'm always on the lookout for sponsorship, affiliate links and writing/coding gigs to keep broadening my online contents. Any support, feedback and suggestions is very much appreciated ! Interested ? Drop an email here : avrab.yt@gmail.com

Also consider becoming my Patreon Member ? — you'll get access to exclusive content, codes, or videos beforehand, one-to-one web app development / relevant discussion, live-chat with me on specific videos and other perks. (FYI : Basic Tier is 50% cheaper than ChatGPT/monthly with benefits which an AI can't help with 😉)

Related Blogs

1. Summarizing Scientific Articles with OpenAI ✨ and Streamlit
2. Build Your Own Chatbot with openAI GPT-3 and Streamlit
3. How to 'stream' output in ChatGPT style while using openAI Completion method
4. ChatGPT helped me to built this Data Science Web App using Streamlit-Python

Recommended YouTube Playlists

1. OpenAI — Streamlit Web Apps
2. Streamlit-Python-Tutorials

Links , references and credits

1. LangChain Docs : https://langchain.readthedocs.io/en/latest/index.html
2. LangChain Prompt Template : https://langchain.readthedocs.io/en/latest/modules/prompts/getting_started.html#what-is-a-prompt-template

3. LangChain SequentialChain Doc : https://langchain.readthedocs.io/en/latest/modules/chains/getting_started.html#combine-chains-with-the-sequentialchain

4. LangChain GitHub Repo : https://github.com/hwchase17/langchain

5. Inspiration from the fast-checker repo : https://github.com/jagilley/fact-checker

6. Streamlit : https://streamlit.io/

7. DataButton : https://www.databutton.io/

8. Open AI document

Thank you ! See you in the next blog , stay tuned 🤖

OpenAI Gpt 3 Langchain Streamlit Large Language Models

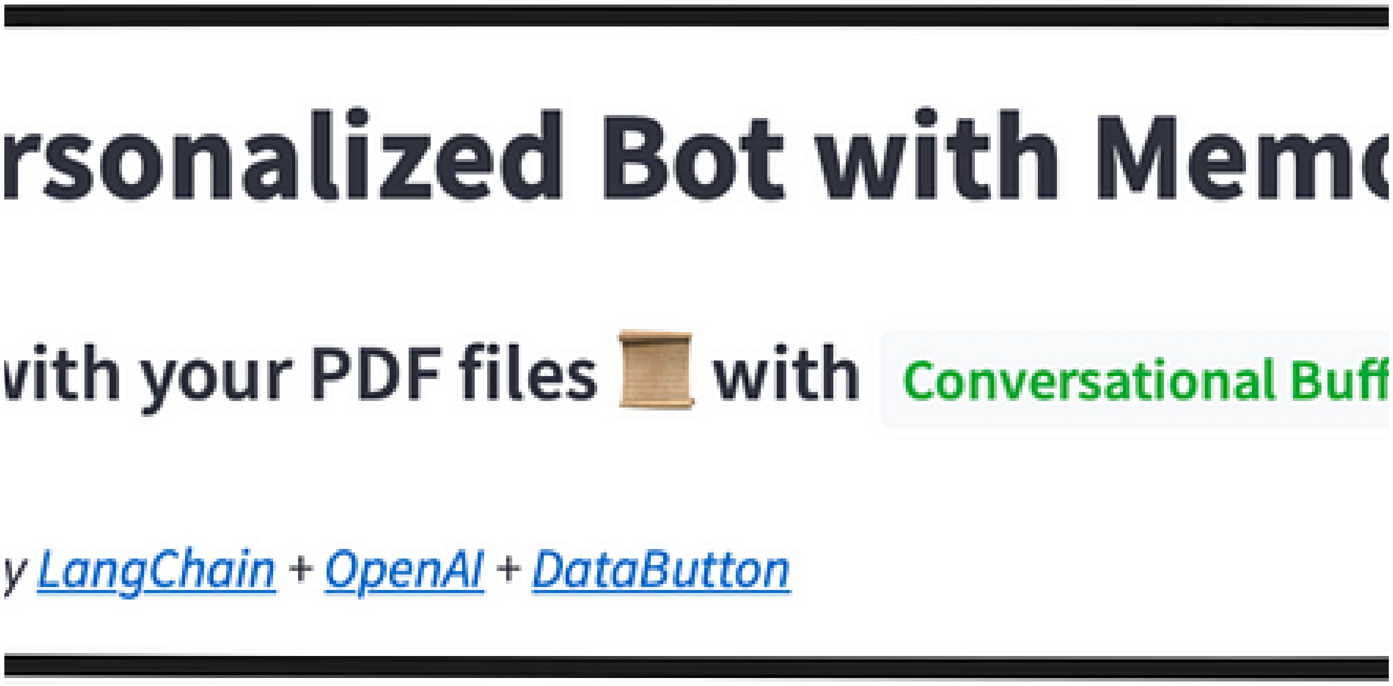
Follow

Written by Avra

790 Followers · Writer for Databutton

Self-taught programmer. PhD candidate, biophysicist and data science enthusiast. Youtube channel - https://www.youtube.com/c/avra_b

More from Avra and Databutton


rsonalized Bot with Memo
vith your PDF files with Conversational Buff
y LangChain + OpenAI + DataButton

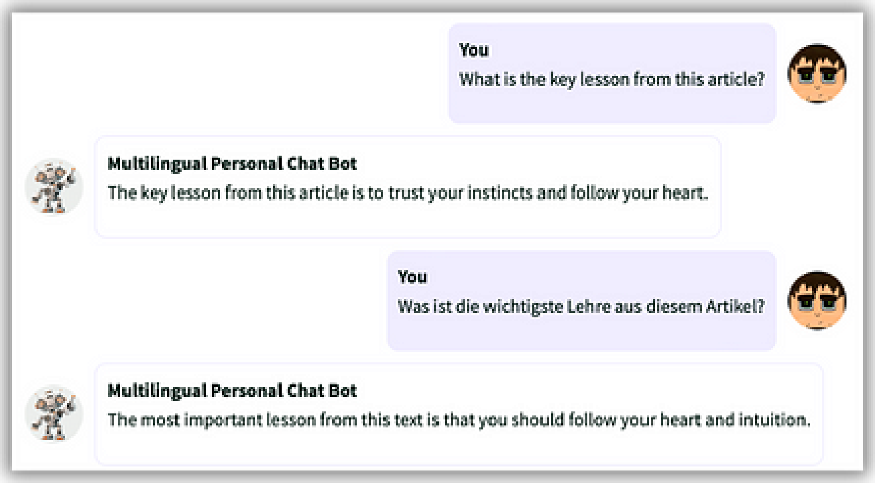
Avra

How to Build a Personalized PDF Chat Bot with Conversational Memory

🧠 Memory to your Personal Chat Bot 🤖— powered by ChatGPT API, LangChainAI and Databutton

10 min read · Apr 16

283 4



You
What is the key lesson from this article?

Multilingual Personal Chat Bot
The key lesson from this article is to trust your instincts and follow your heart.

You
Was ist die wichtigste Lehre aus diesem Artikel?

Multilingual Personal Chat Bot
The most important lesson from this text is that you should follow your heart and intuition.

Avra in Databutton

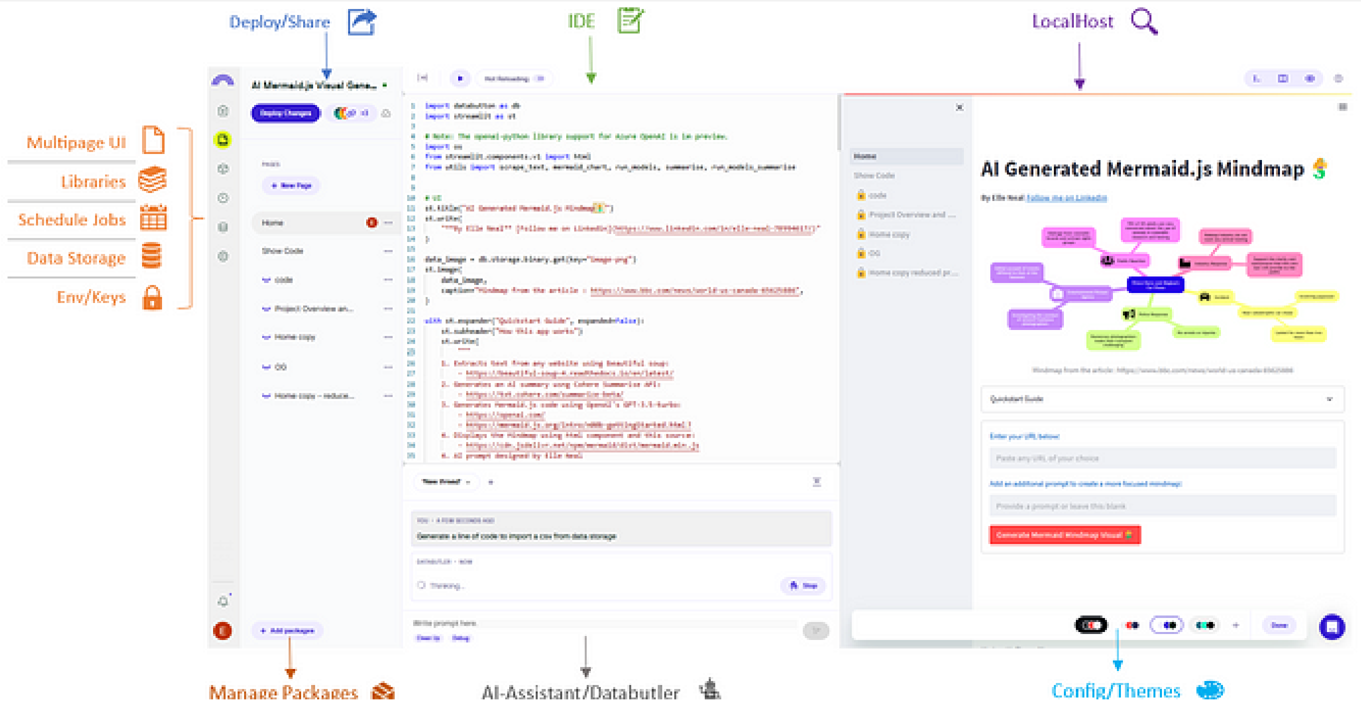
Multilingual Chat Bot for Personal Documents — using Cohere's Multilingual Models & LangChain

Building a multilingual chat bot using Cohere, LangChain, and Databutton

7 min read · 5 days ago

Open in app ↗ Sign up Sign In

Search Medium



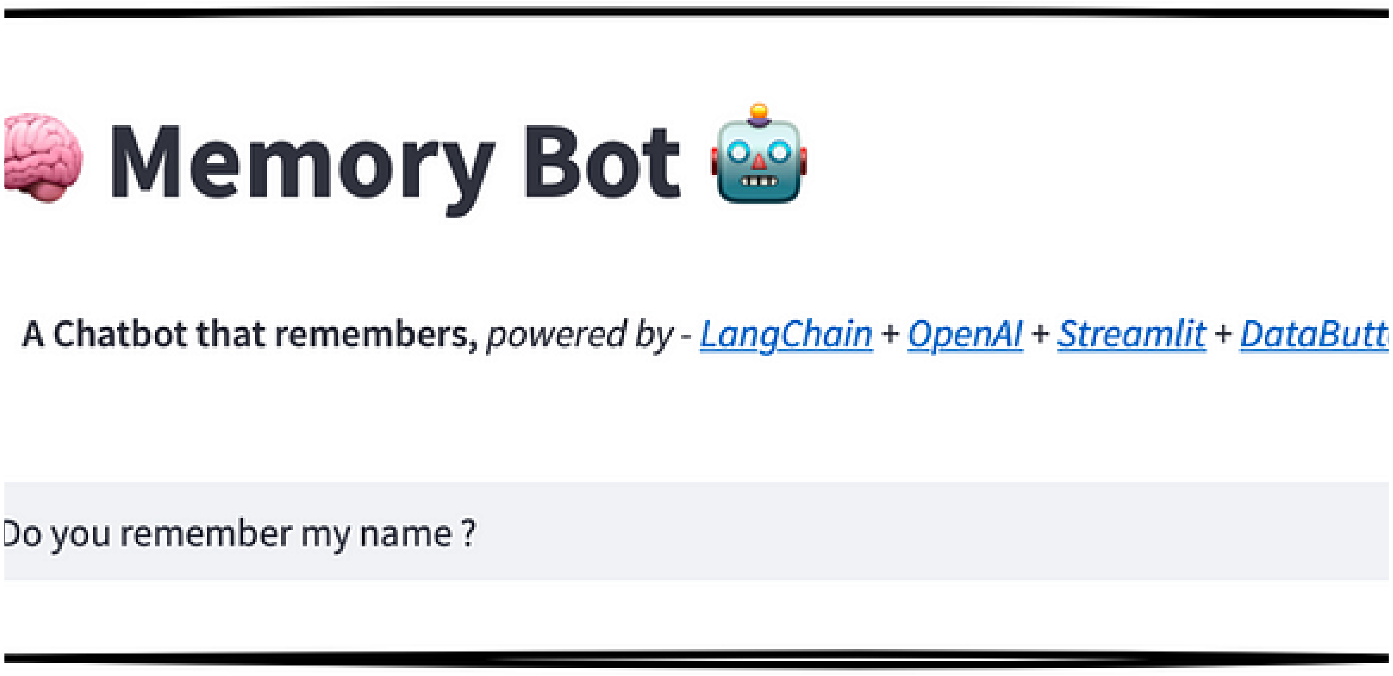
Elle Neal in Databutton

Beyond Streamlit: Databutton's Revolution in AI App Development

🚀Discover how Databutton is transforming the landscape of AI app development, providing an all-in-one online workspace.

6 min read · Jun 5

181


🧠 Memory Bot 🤖
A Chatbot that remembers, powered by - LangChain + OpenAI + Streamlit + DataButt
Do you remember my name ?

Avra

How to build a Chatbot with ChatGPT API and a Conversational Memory in Python

🧠 Memory Bot 🤖—An easy up-to-date implementation of ChatGPT API, the GPT-3.5-Turbo model, with LangChain AI's 🦜 — ConversationChain…

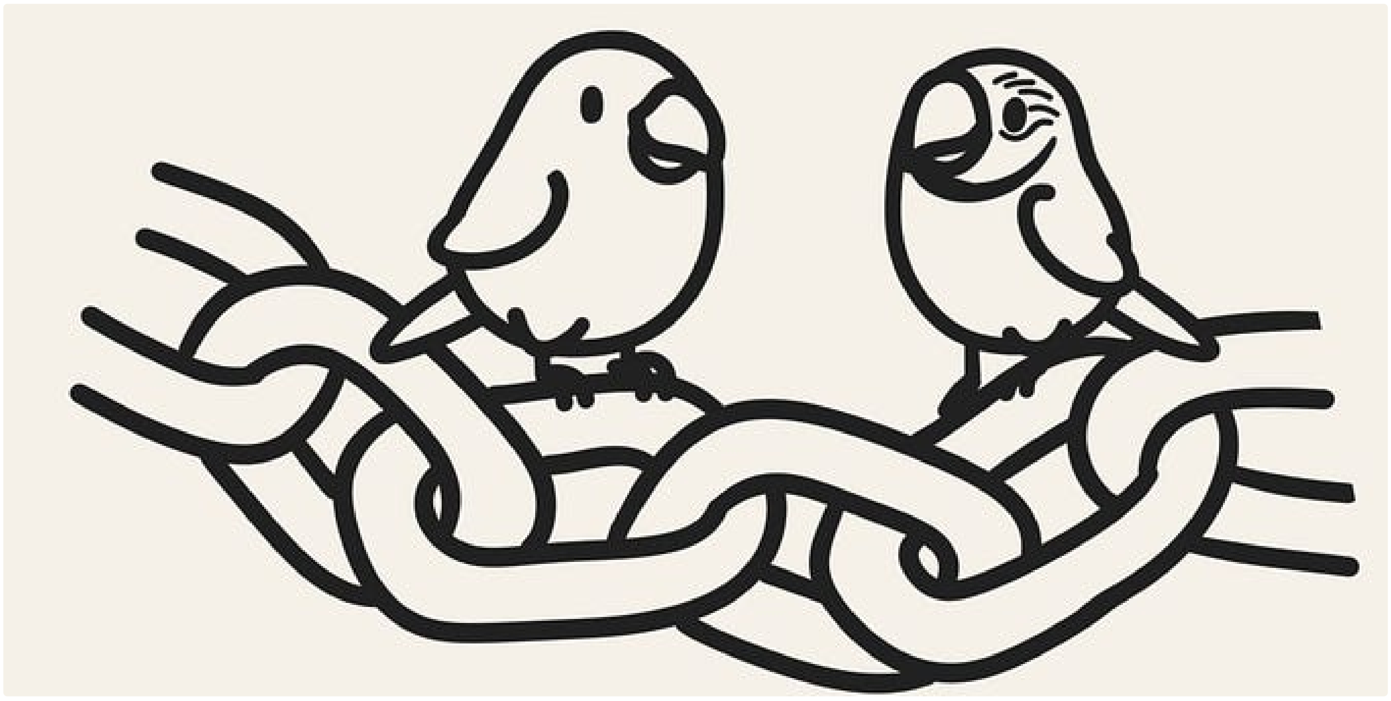
11 min read · Mar 18

358 6

See all from Avra

See all from Databutton

Recommended from Medium



Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner's Guide to Building LLM-Powered Applications

A LangChain tutorial to build anything with large language models in Python

· 12 min read · Apr 25

2.8K 19

Wei-Meng Lee in Level Up Coding

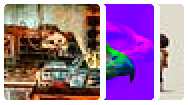
Training Your Own LLM using privateGPT

Learn how to train your own language model without exposing your private data to the provider

· 8 min read · May 19

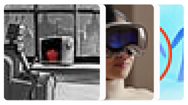
1K 9

Lists



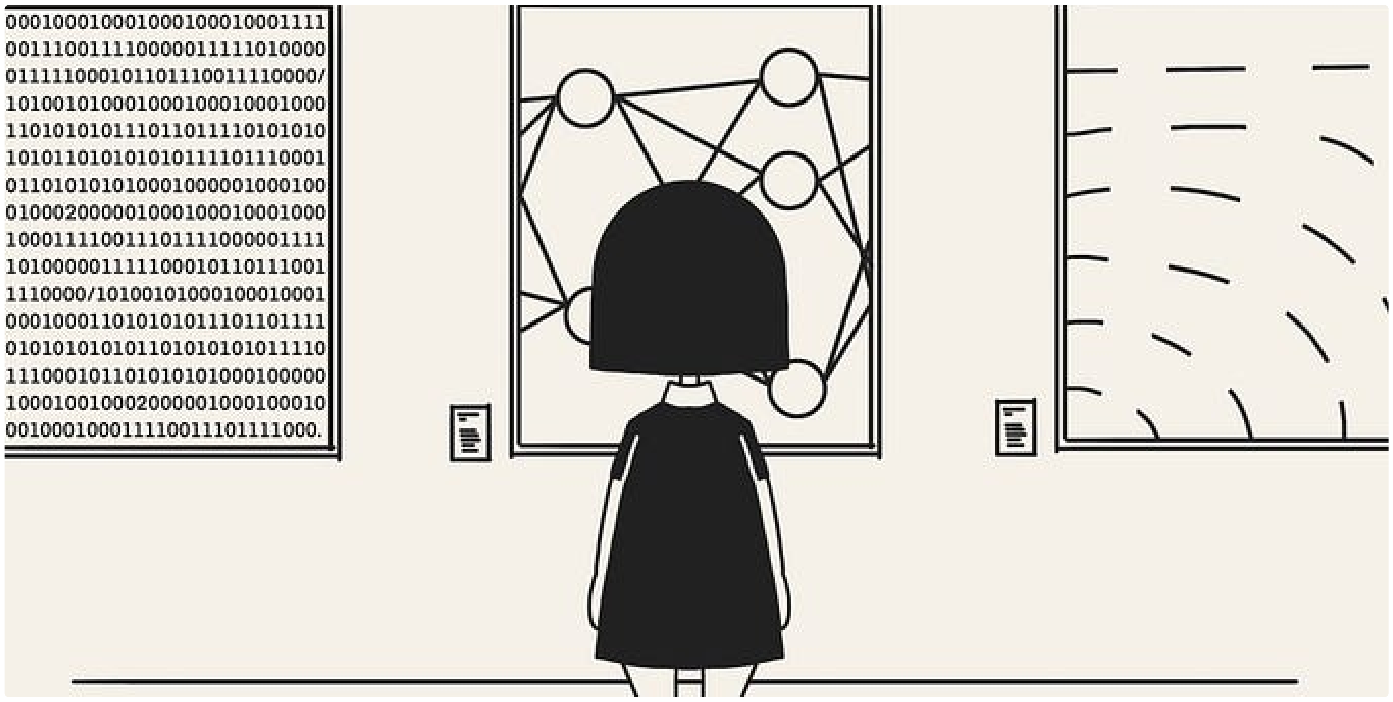
What is ChatGPT?

9 stories · 94 saves



Staff Picks

347 stories · 107 saves



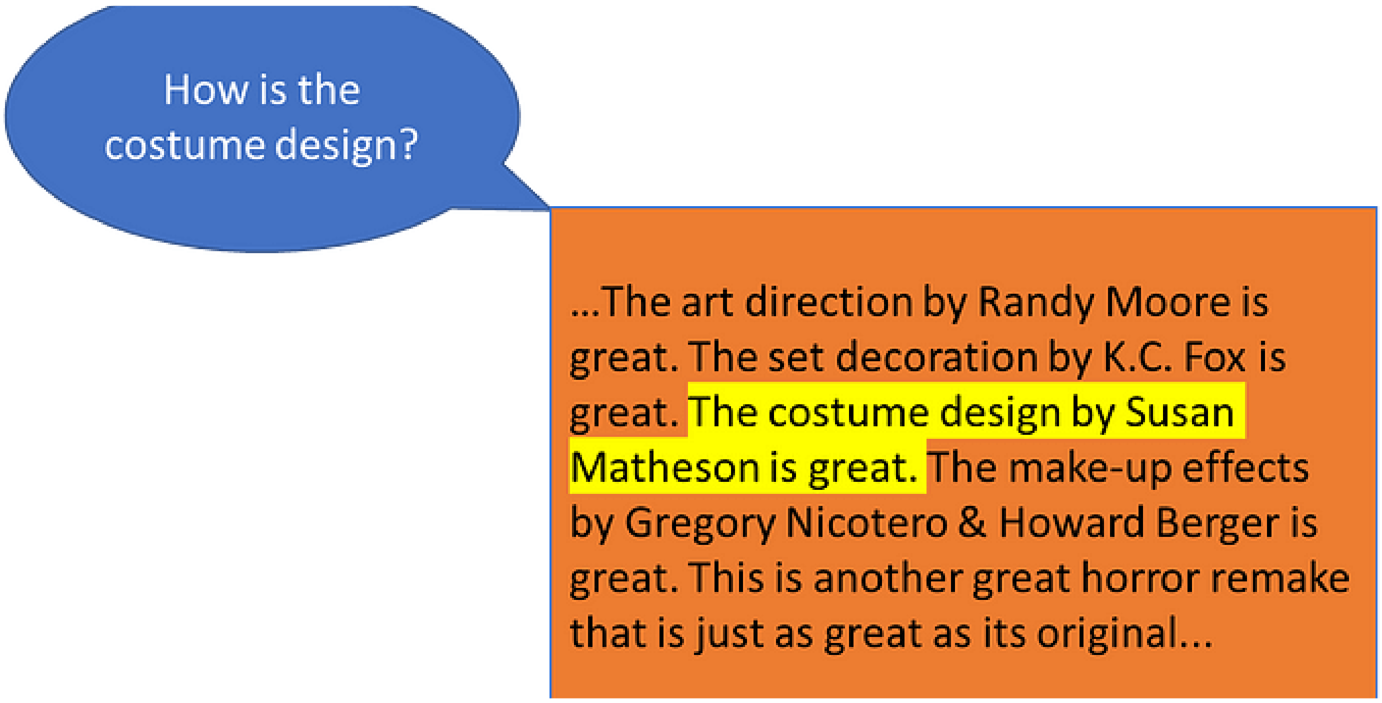
Leonie Monigatti in Towards Data Science

10 Exciting Project Ideas Using Large Language Models (LLMs) for Your Portfolio

Learn how to build apps and showcase your skills with large language models (LLMs). Get started today!

· 11 min read · May 15

1.2K 7


How is the costume design?
...The art direction by Randy Moore is great. The set decoration by K.C. Fox is great. The costume design by Susan Matheson is great. The make-up effects by Gregory Nicotero & Howard Berger is great. This is another great horror remake that is just as great as its original...

Skanda Vivek in Towards Data Science

Fine-Tune Transformer Models For Question Answering On Custom Data

A tutorial on fine-tuning the Hugging Face RoBERTa QA Model on custom data and obtaining significant performance boosts

· 5 min read · Dec 15, 2022

378 3

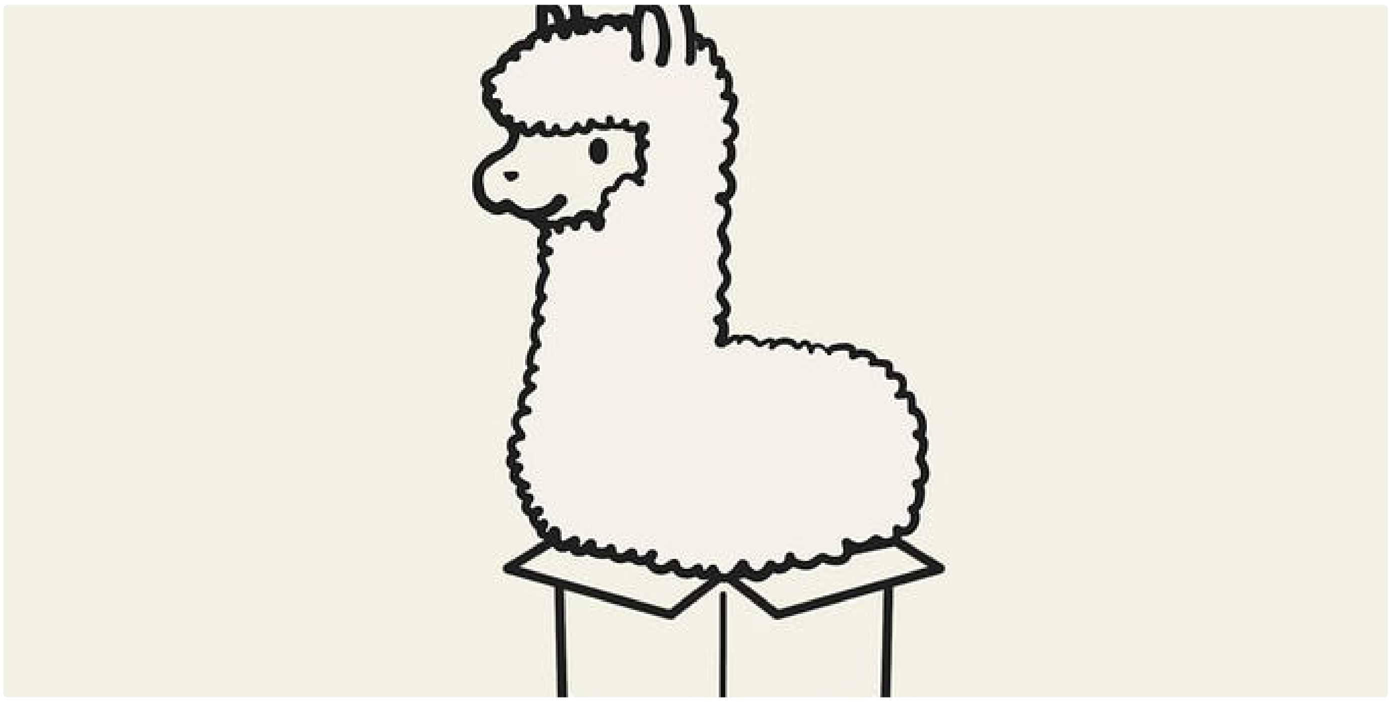
Gabe Araujo, M.Sc. in Level Up Coding

🐼Introducing PandasAI: The Generative AI Python Library 🐼

Pandas AI is an additional Python library that enhances Pandas, the widely-used data analysis and manipulation tool, by incorporating…

· 9 min read · May 17

992 11



Leonie Monigatti

Understanding LLMOps: Large Language Model Operations

How LLMs are changing the way we build AI-powered products and the landscape of MLOps

· 12 min read · May 2

636 6

See more recommendations