AI Sommelier Built with PaLM API and LangChain

Project walkthrough to build an LLM-powered application using Pa API, Pinecone as the vector database, and Streamlit as the interface.

Gabriel Cassimiro · Follow

5 min read · Sep 26, 2023

Image by Hermes Rivera at Unsplash

In this article, I aim to demonstrate how to use the PaLM API to build solutions using LangChain. I will build a solution using LangChain chains, Google Embeddings, Pinecone as the VectorDB, and Streamlit as the interface to interact with users.

The problem we want to tackle is personalized wine recommendations. I want to build an AI capable of recommending wines based on user preference and also using a database containing 130K wines.

The full code can be found here.

The data

The data used in this project is the dataset wine reviews, license. This dataset contains 130K reviews of wines. It also contains the country, region, variety, and winery of each wine. This is a sample of the data:

country	description	points	price	province	region_1	title	variety	winery
Italy	Aromas include tropical fruit, broom, brimston...	87	NaN	Sicily & Sardinia	Etna	Nicosia 2013 Vulkà Bianco (Etna)	White Blend	Nicosia
Portugal	This is ripe and fruity, a wine that is smooth...	87	15.0	Douro	NaN	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Portuguese Red	Quinta dos Avidagos
US	Tart and snappy, the flavors of lime flesh and...	87	14.0	Oregon	Willamette Valley	Rainstorm 2013 Pinot Gris (Willamette Valley)	Pinot Gris	Rainsto
US	Pineapple rind, lemon pith and orange blossom ...	87	13.0	Michigan	Lake Michigan Shore	St. Julian 2013 Reserve Late Harvest Riesling ...	Riesling	St. J
US	Much like the regular bottling from 2012, this...	87	65.0	Oregon	Willamette Valley	Sweet Cheeks 2012 Vintner's Reserve Wild Child...	Pinot Noir	Sweet Cheeks

Image by Author

The Problem

So, to tackle our problem we want to first understand the taste of the user, then search the database for the wines that may be a good recommendation, and decide which one is the final recommendation.

To get the user input we will use a simple questionnaire inside of Streamlit with the following questions:

1. Preferred Taste Profile;
2. Level of experience;
3. Red or White wine preference;
4. Favorite Flavours;
5. Pairing intent;
6. Open field to add any information about your taste.

All of these (except the last) have pre-selected categories. However, since we are using LLMs this isn't strictly necessary. We could leave all questions with free text input because the LLM is able to use it without the categories. However, to guide the user I chose to use categories.

The Architecture

The Architecture of the solution is shown below:

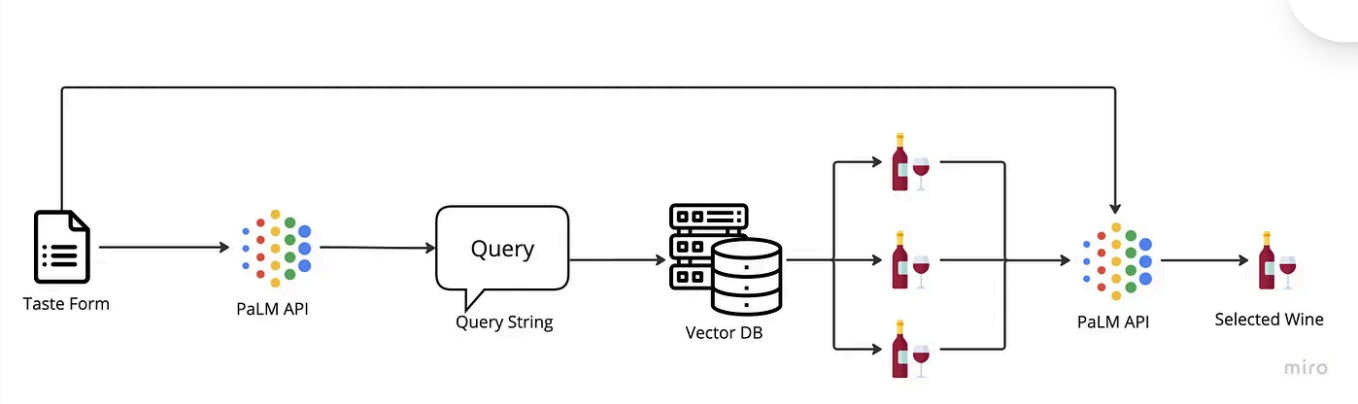


Image by Author

For the solution, we will have two calls to the LLM API. In the first one, we will pass the taste form information and ask for the LLM to generate a string query that summarizes the taste and will be used for similarity search on the vector DB. After that, we can find the most similar descriptions of the wines to the person's taste.

However, this is not enough for a final decision, because some characteristics like red vs. white are not taken into account by this simple similarity search. That is why we call the LLM again passing the original taste form and the top 3 most similar wines to the LLM and ask it to select the best one for it and explain the reasoning.

The Code

Let's start by looking at the chains used for the calls to the API.

First, if you do not know already what is LangChain, this is a brief definition:

> *LangChain is a framework for developing applications powered by language models.*

That is the official definition of LangChain. This framework was created recently and is already used as the industry standard for building tools powered by LLMs.

Chains are a core feature of LangChain and enable the integration of various components to form a unified application. They can format user input using a PromptTemplate and then forward it to an LLM. By linking multiple chains or combining them with other elements, we can create more complex chains.

This is the code for both of the chains:

```python
from langchain.prompts import ChatPromptTemplate
from langchain.chains import LLMChain, SequentialChain
from langchain.output_parsers import ResponseSchema, StructuredOutputParser, RetryWithEr


def build_query_chain(llm):

    query_string = ResponseSchema(
        name="query_string",
        description="The string used to query the vector database for wine options.",
    )

    output_parser = StructuredOutputParser.from_response_schemas(
        [query_string]
    )
    response_format = output_parser.get_format_instructions()

    prompt = ChatPromptTemplate.from_template(
        """You are an expert wine sommelier. Your goal is to select a wine from the data
        Take a breath and understand the following user preferences:
        taste: {taste}
        experience level:{experience}
        wine color: {wine_color}
        flavor: {flavor}
        pairing: {pairing}
        complement: {complement}

        Now create a string that will be used to do a similarity search on a vector data
        To build better queries for similarity search, ensure they are specific, utilize
        {response_format}
        """
    )

    small_chain = LLMChain(llm=llm, prompt=prompt, output_key="query")

    chain = SequentialChain(
        chains=[small_chain],
        input_variables=["taste", "experience","wine_color","flavor","pairing","compleme
        output_variables=["query"],
        verbose=False,
    )
    return chain, response_format, output_parser

def build_recommendation_chain(llm):

```

```
    recommendation = ResponseSchema(
        name="recommendation",
        description="The recommended wine Name.",
    )

    explanation = ResponseSchema(
        name="explanation",
        description="The explanation of the selected recommendation.",
    )

    output_parser = StructuredOutputParser.from_response_schemas(
        [explanation,recommendation]
    )
    response_format = output_parser.get_format_instructions()

    ## Issue and Summary Chain
    prompt = ChatPromptTemplate.from_template(
        """You are an expert wine sommelier. Your goal is to select a wine from the opti
        Take a breath and understand the following user preferences:
        taste: {taste}
        experience level:{experience}
        wine color: {wine_color}
        flavor: {flavor}
        pairing: {pairing}
        complement: {complement}

        Now take a breath and understand the wine options:
        Option 1:
        {wine_1}
        Option 2:
        {wine_2}
        Option 3:
        {wine_3}

        Now select the best wine to recommend to this user.

        {response_format}
        """
    )

    small_chain = LLMChain(llm=llm, prompt=prompt, output_key="recommendation")

    chain = SequentialChain(
        chains=[small_chain],
```

```
        input_variables = [
        "taste", "experience", "wine_color", "flavor", "pairing", "complement",
        "response_format", "wine_1", "wine_2", "wine_3"],
        output_variables=["recommendation"],
        verbose=False,
    )
    return chain, response_format, output_parser
```

build_chains.py hosted with ❤ by GitHub view raw

Here we have a couple of elements I would like to point out. First, we have the **Response Schema** and **Output Parser**. These are used to create a prom in which the LLM will always return the output with the same structure an the parser will process that output to be able to work back in the code.

Then we have the prompt built using the **Chat Prompt Template**. This allows us to pass variables into the prompt like the answers of the taste form. Lastly, we put everything back together with the **Sequential Chain**, defining the inputs and outputs.

Now we have to initialize the DB.

The Database

For the Vector Database, we are using Pinecone however you can easily change the one you want to use because we are using LangChain to interact with the DB.

To add the wines to the Database we need to first create an Embedding of each one and then upload them. We can easily perform this with the code below:

We create an instance of Document from LangChain containing the text and the Metadata. Then we use LangChain and Google PaLM Embeddings to upload to Pinecone. The embeddings have 768 dimensions.

For this implementation I am using the free tier from Pinecone, which allows for 1 free index of the standard resource.

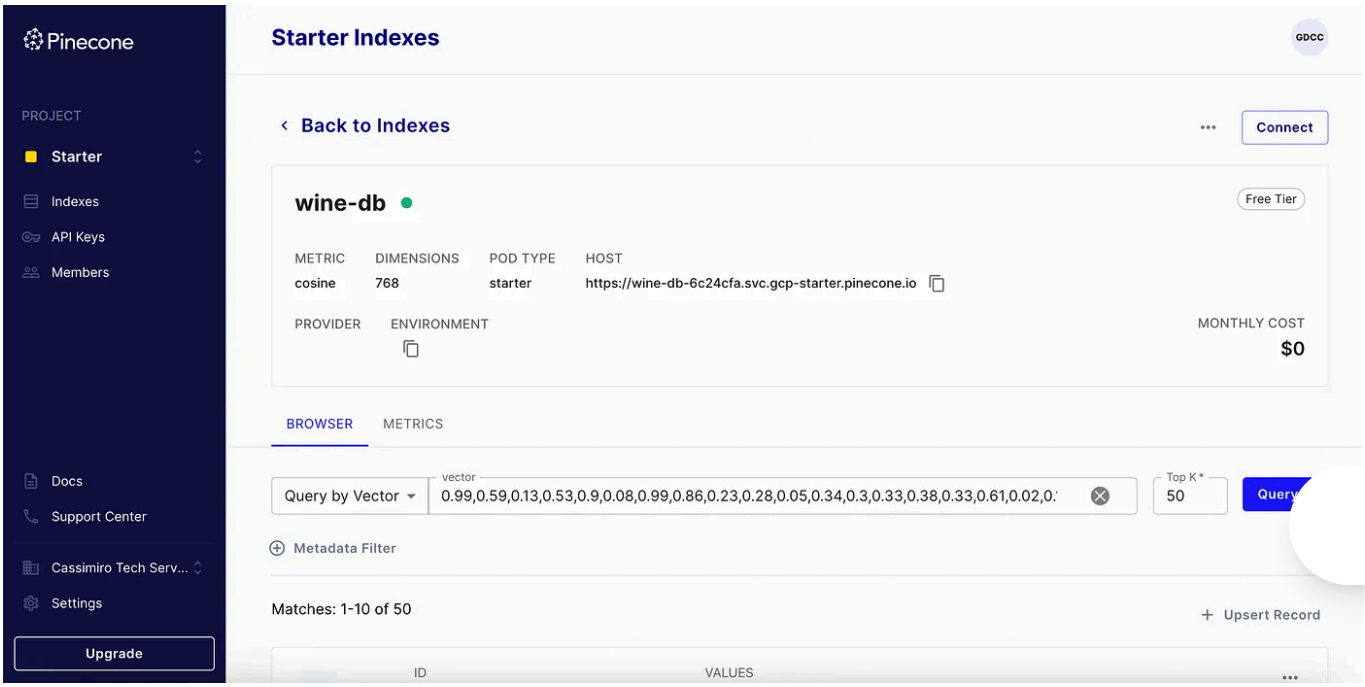


Image by Author

Now we just need to put everything together passing the input from the application to the chains and connect to the vector database.

The App

The App will be the main control of the flow. It is also where we will create all the resources we need: the form, the PaLM API Connection, the Pinecone connection and the final visualization.

This code can be divided in three main parts:

1. The form to get the user input;
2. The interaction of the chains and the vector database;
3. The display of the final recommendation.

These parts are executed inside the main script but for a larger application should be modularized.

This is an example of using the tool:

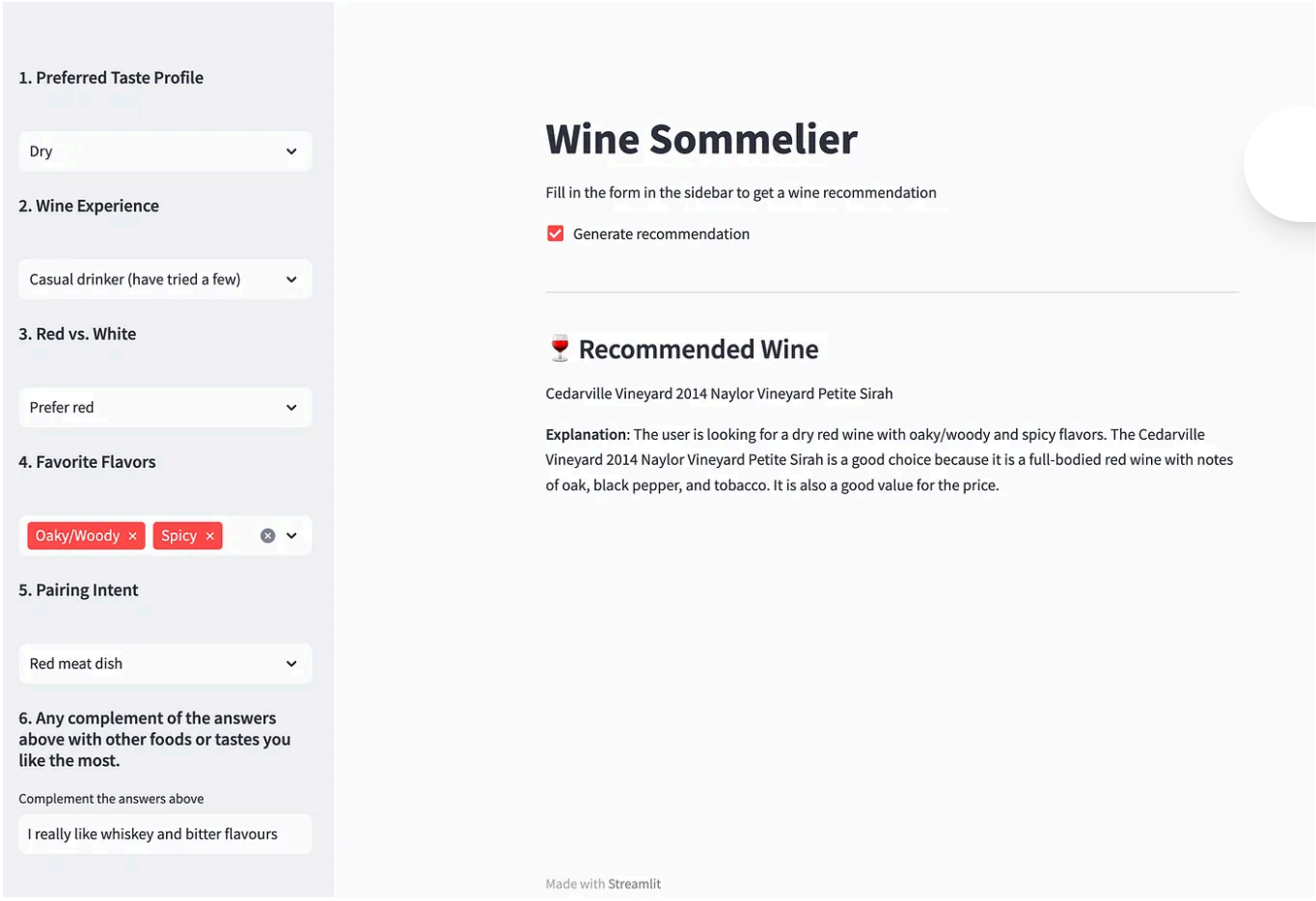


Image by Author

So this is the final product. All we have to do now is deploy this application. With this in mind, the first thing we need is to allow the user to choose which LLM to use and have an input for the API key.

The final version can be seen here.

Conclusion

Using LangChain allows us to build quick and in a modular manner. For the problem of recommending wines based on a database we were able to use Chains with two prompts that helped to understand the users taste, find some options to suggest and make the final decision with an explanation.

An easy way to interact with the user is through an interface such as Streamlit, making it fast to develop and deploy.

Thanks for reading.

If you like the content and want to support me, you can buy me a coffee:

Gabriel Cassimiro is a Data Scientist sharing free content to the community

I love supporting creators!

www.buymeacoffee.com

Here are a few other articles you might be interested in:

Async calls for Chains with Langchain

How to make Langchain chains work with Async calls to LLMs, speeding up the time it takes to run a sequential long...

towardsdatascience.com

Solving Unity Environment with Deep Reinforcement Learning

End to End Project with code of a PyTorch implementation of Deep Reinforcement Learning Agent.

towardsdatascience.com

Llm Langchain Pinecone Palm Palm Api

More from the list: "Langchain."

Curated by Shubham Rathod

Matheus Ferreira
AI Food Advisor with LangChain and Pinecone
5 min read · Oct 13, 2023

Jamesev
Playlist Recommender based on Spotify,...
3 min read · Jun 25, 2023

Emil Azadia
Using Agents LangChain
8 min read · Ma

View list

Written by Gabriel Cassimiro

Follow

208 Followers

Solving and creating problems with AI. Google Developer Expert in ML

More from Gabriel Cassimiro

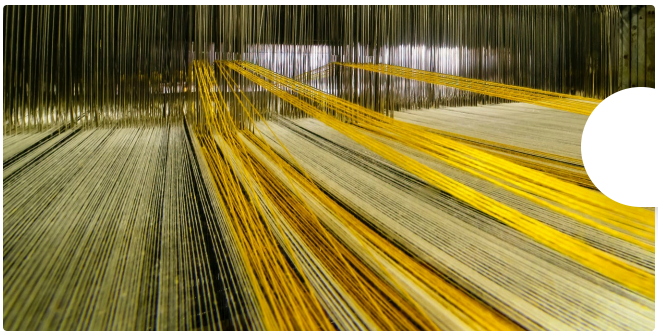
Gabriel Cassimiro in Towards Data Science

LLM Output Parsing Function Calling vs. LangChain

How to consistently parse outputs from LLMs using Function Calling of Open AI API and...

11 min read · Sep 21, 2023

271 1



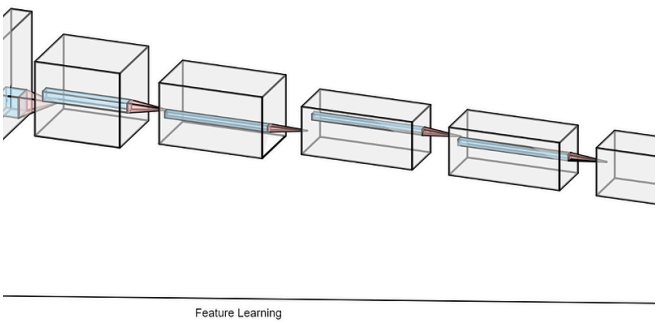
Gabriel Cassimiro in Towards Data Science

Async calls for Chains with Langchain

How to make Langchain chains work with Async calls to LLMs, speeding up the time it...

6 min read · Jul 10, 2023

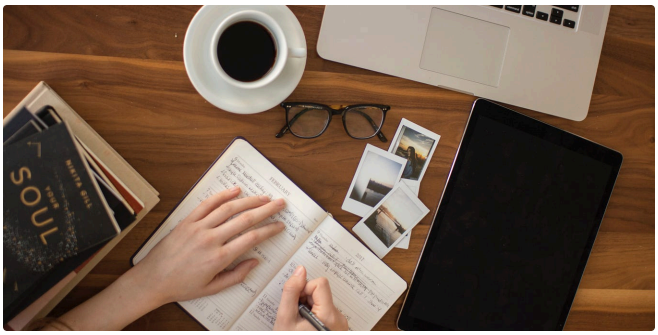
380 4



Gabriel Cassimiro in Towards Data Science

Transfer Learning with VGG16 and Keras

How to use a state-of-the-art trained NN to solve your image classification problem



Gabriel Cassimiro in Towards Data Science

How to prepare for the GCP Professional Machine Learning...

Courses review, study tips, and how I did it

4 min read · Jun 16, 2021

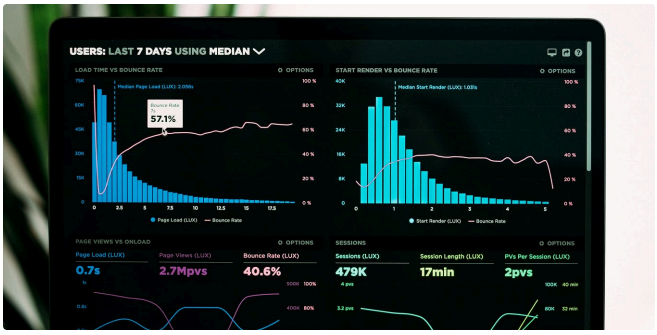
253 1

8 min read · Jan 10, 2022

300 3

See all from Gabriel Cassimiro

Recommended from Medium



Chandan Durgia

Evaluation of RAG (Retrieval-Augmented Generation)...

Quantifying the accuracy and relevance of the RAG output

9 min read · Feb 21, 2024

91 1

Supriyo Roy Banerjee

Text Summarization with Generative Models on Vertex AI

Overview

8 min read · Oct 11, 2023

Lists

Natural Language Processing

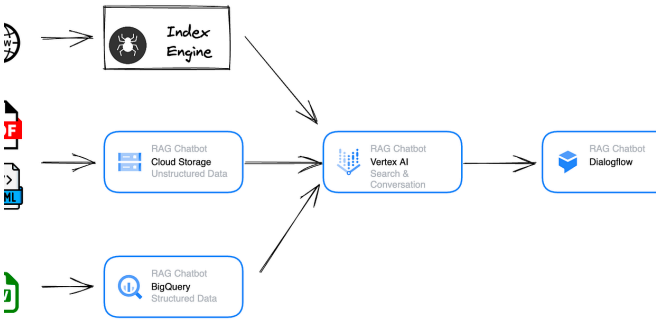
1250 stories · 731 saves

ChatGPT prompts

44 stories · 1198 saves

Staff Picks

594 stories · 790 saves



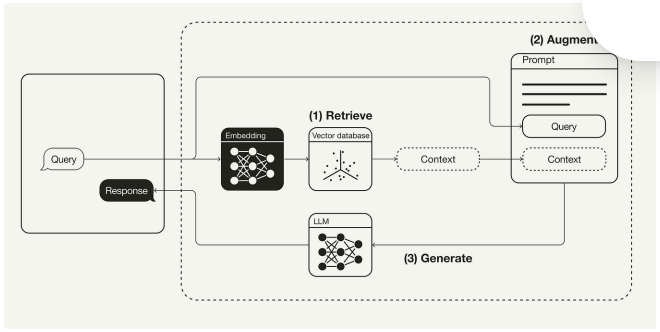
Euclidean AI in Towards AI

Use Google Vertex AI Search & Conversation to Build RAG Chatbot

Google has recently released their managed RAG (Retrieval Augmented Generator)...

6 min read · Oct 25, 2023

37 5



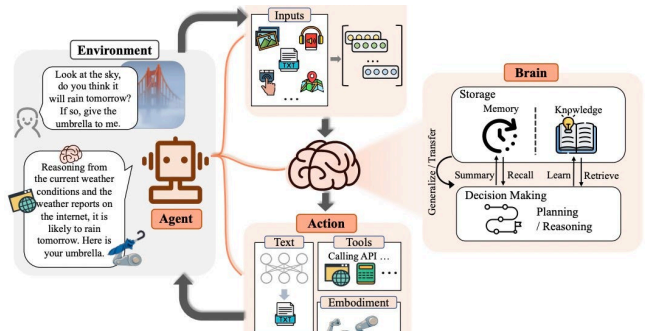
Leonie Monigatti in Towards Data Science

Retrieval-Augmented Generation (RAG): From Theory to LangChain...

From the theory of the original academic paper to its Python implementation with...

7 min read · Nov 14, 2023

1.2K 9



Henry Heng LUO

Saumya Mittal

Intro of AI agent, & AI agent projects summary

Basic introduction of AI agent, and 13 interesting AI agent frameworks and AI agen...

16 min read · Jan 23, 2024

183

Ultimate Guide to Functions-Calling with LangChain

Introduction

3 min read · Nov 24, 2023

3

See more recommendations