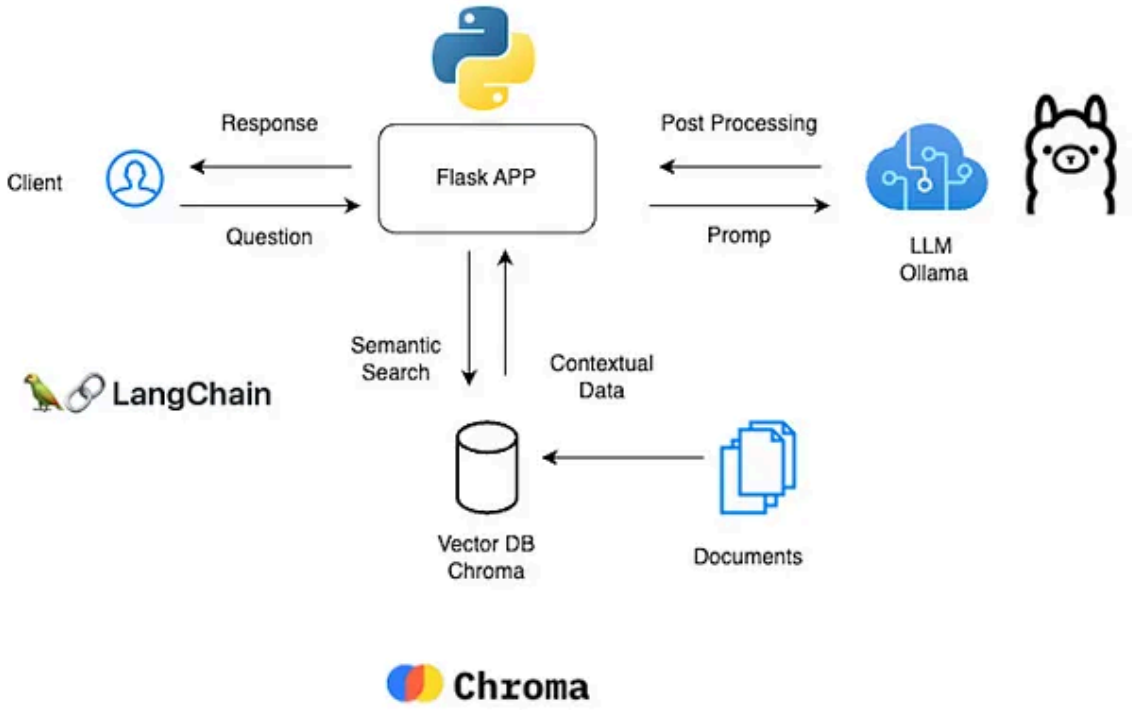
Build Your Free AI Assistant-RAG Python &Ollama

Osman Yılmaz · Follow

4 min read · Dec 18, 2024

Firstly, I explained the details and meaning of RAG (Retrieval-Augmented Generation) in another article



If you'd like to use your own local AI assistant or document-querying system, I'll explain how in this article, and the best part is, you won't need to pay for any AI requests.

Modern applications demand robust solutions for accessing and retrieving relevant information from unstructured data like PDFs. Retrieval-Augmented Generation (RAG) is a cutting-edge approach combining AI's language generation capabilities with vector-based search to provide accurate, context-aware answers.

In this article, I will walk through the development of a Flask application that implements a RAG pipeline using LangChain, Chroma. The application

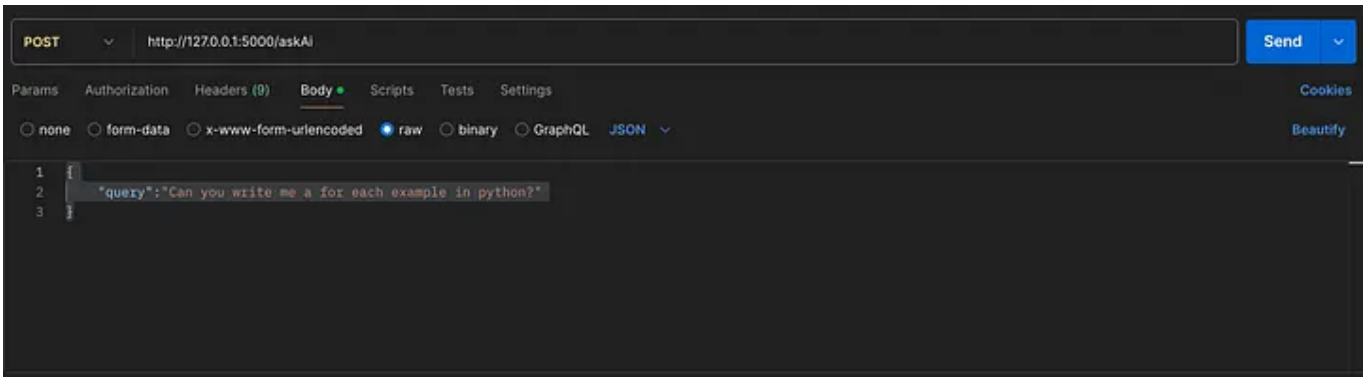
allows users to upload PDF documents, store embeddings, and query them for information retrieval — all powered by Ollama.

If you're ready to create a simple RAG application on your computer or server, this article will guide you. In other words, you'll learn how to build your own local assistant or document-querying system.

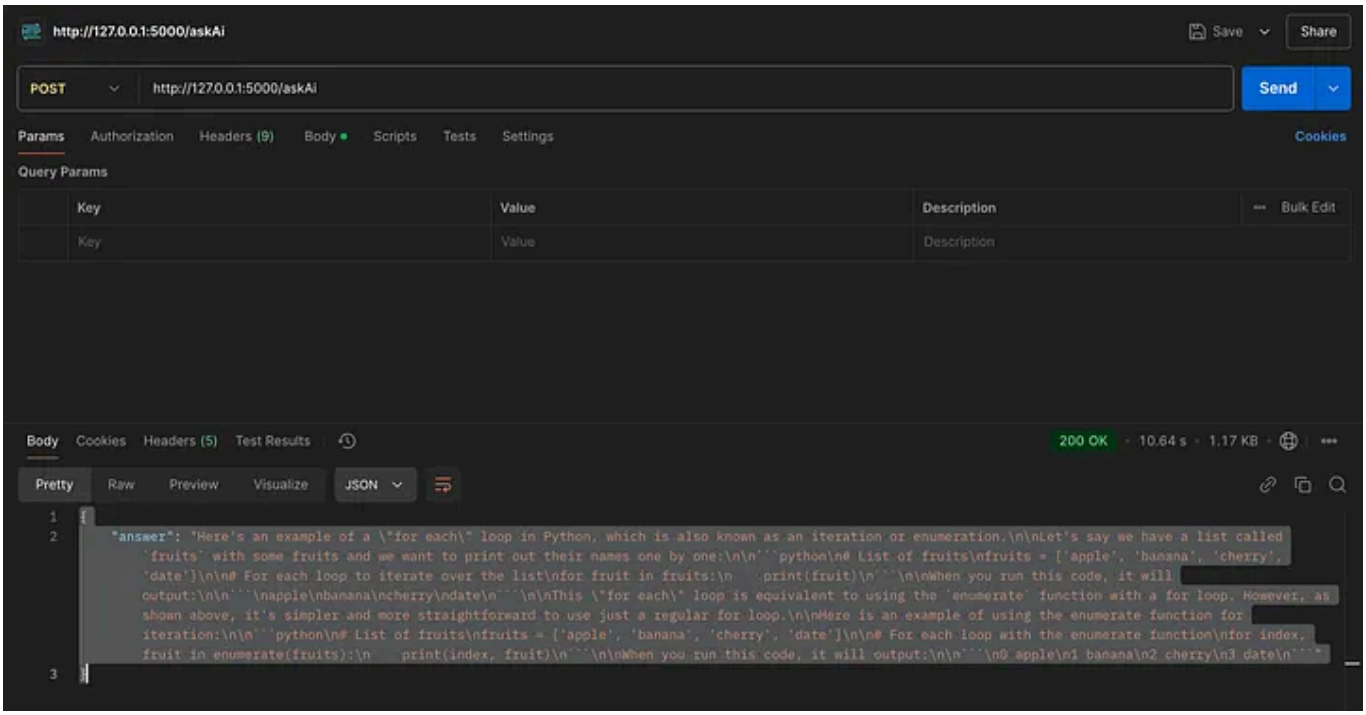
What We'll Build

Our application includes the following features:

- Querying a general-purpose AI for any question. You can ask local LLM model direct question by using flask post method

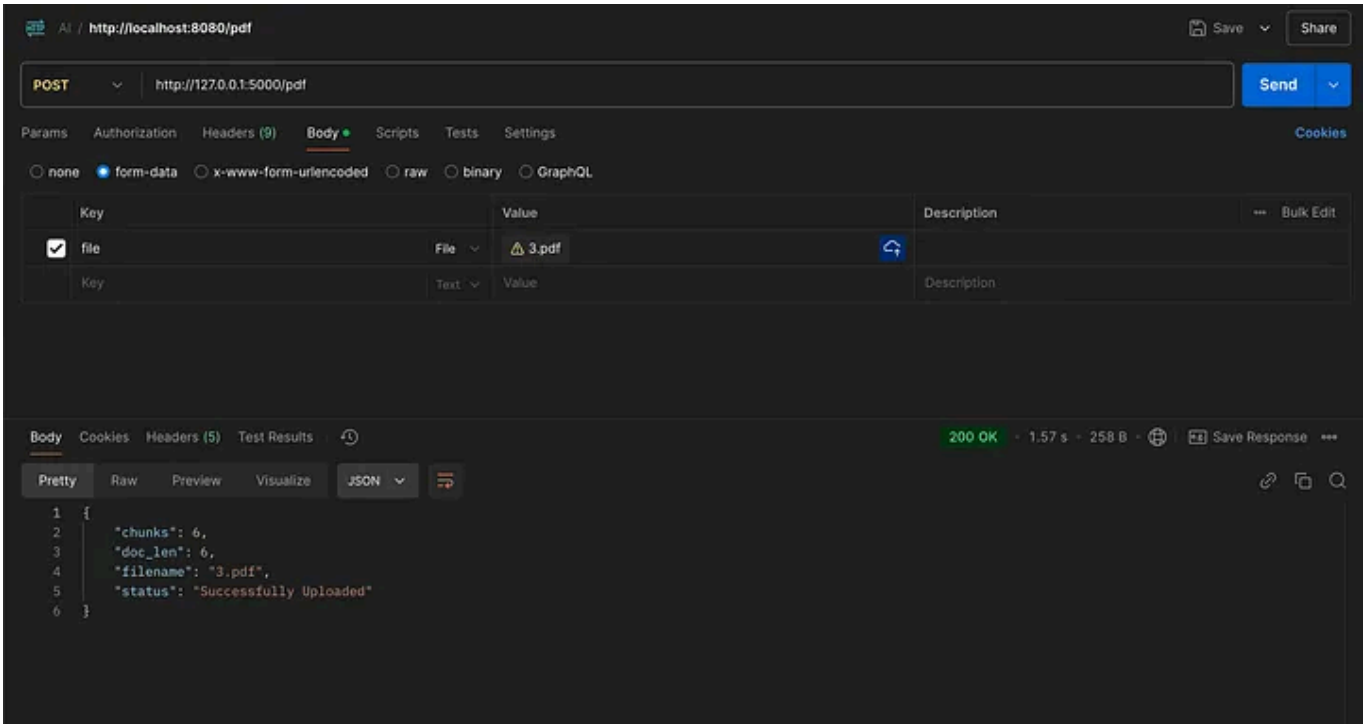


Post method -> simple question



Ollama answered

- Uploading PDF documents to process and store. You can upload your PDFs to local storage and local vector db.



- You can ask anything about your documents. Retrieving answers specific to the content of uploaded PDFs.

Tools and Libraries

We'll use the following stack to build this application:

- **Flask:** A lightweight Python framework for creating REST APIs.
- **LangChain:** For constructing the retrieval and generation pipeline.
- **Chroma:** A vector store for storing and querying document embeddings.
- **PDFPlumber:** For extracting text from PDFs.
- **Ollama LLM:** Open source a large language model to generate AI responses.

Step-by-Step Implementation

1. Setting Up Flask

Flask serves as the backbone of our API. It exposes the endpoints required to handle user queries, PDF uploads, and AI responses.

First we access to local LLM than we will return the answers by using Json.

```
cached_llm = Ollama(model="llama3.2:latest")
```

```
@app.route("/askAi", methods=["POST"])
def ask_ai():
    print("Post /ai called")
    json_content = request.json
```

```
        query = json_content.get("query")
        print(f"query: {query}")
        response = cached_llm.invoke(query)
        print(response)
        response_answer = {"answer": response}
        return response_answer
```

2. Uploading and Processing PDFs

Using `PDFPlumber`, we extract text from PDFs. LangChain's `RecursiveCharacterTextSplitter` splits the text into manageable chunks, which are embedded and stored in Chroma for efficient querying.

```
from langchain_community.document_loaders import PDFPlumberLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter

loader = PDFPlumberLoader("example.pdf")
docs = loader.load_and_split()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1024, chunk_overlap=80
chunks = text_splitter.split_documents(docs)
```

3. Storing Data in Chroma

Chroma serves as our vector database, storing embeddings for the document chunks. These embeddings allow similarity-based search during retrieval.

```
from langchain_community.vectorstores import Chroma

vector_store = Chroma.from_documents(chunks, embedding=FastEmbedEmbeddings())
vector_store.persist(directory="db")
```

4. Querying the PDFs

The `/askPdf` endpoint retrieves context from the vector store and uses a prompt template to generate answers via the AI model.

```
@app.route("/askPdf", methods=["POST"])
def ask_pdf():
    query = request.json["query"]
    retriever = vector_store.as_retriever(search_type="similarity_score_threshol
    chain = create_retrieval_chain(retriever, document_chain)
    result = chain.invoke({"input": query})
    return {"answer": result["answer"]}
```

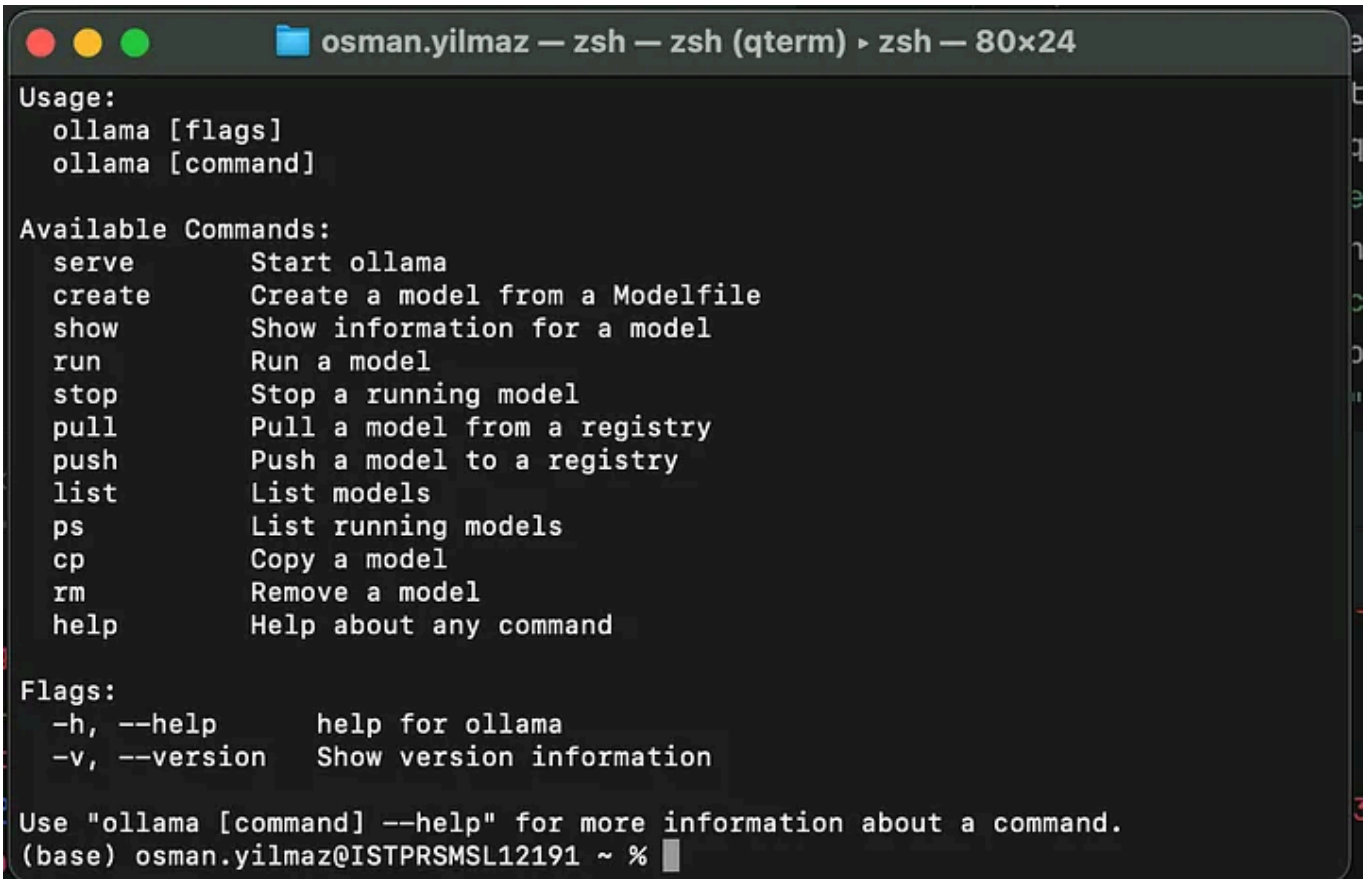
Endpoints Overview

The Flask application provides three main endpoints:

1. **/askAi**: Directly queries the AI for general-purpose answers.
2. **/askPdf**: Queries the uploaded PDFs for context-specific answers.
3. **/addPdf**: Uploads and processes a PDF document for storage.

Each endpoint is designed for simplicity and integrates tightly with the LangChain framework for RAG operations.

Running the Application



First you have to serve Ollama than install the requirements libs.

```
1- ollama serve

2- pip install -r requirements.txt

3- python app.pyhttp://127.0.0.1:5000
```

Challenges and Solutions

1. Efficient Text Chunking

Splitting documents without losing context is critical. Using LangChain's `RecursiveCharacterTextSplitter`, we ensure chunks retain coherence while remaining small enough for embedding.

2. Scalable Vector Storage

Chroma provides a fast, lightweight vector store ideal for handling large datasets while enabling quick queries.

3. Context-Aware Responses

A custom prompt template ensures that the AI model integrates retrieved document content seamlessly into its answers.

Conclusion

By combining AI, vector search, and document processing, this application demonstrates the power of RAG in solving real-world problems. Whether you're a researcher , a developer or a company employee, this guide equips you with the tools to build a custom document-querying system.

Want to take this further? Clone the GitHub repository and start building today!

Llm Ollama Python

Written by Osman Yılmaz

78 Followers · 29 Following

Director of Mobile | EA | AI

Follow

Responses (1)

Akhan

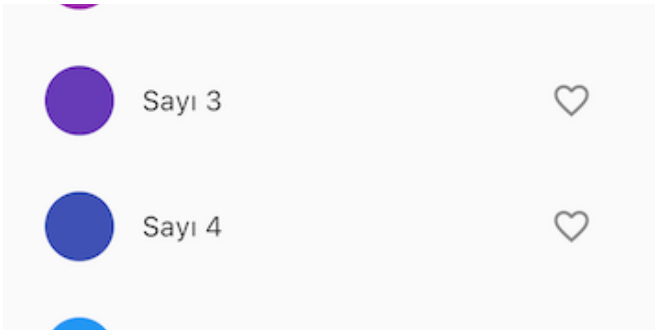
What are your thoughts?

Daniel
Jan 2

I tried to run this one but keep getting " np.float_ ` was removed in the NumPy 2.0 release" when uploading a PDF file.

Reply

More from Osman Yılmaz



Osman Yılmaz

What is RAG (Retrieval augmented generation)

In Team Kraken by Osman Yılmaz

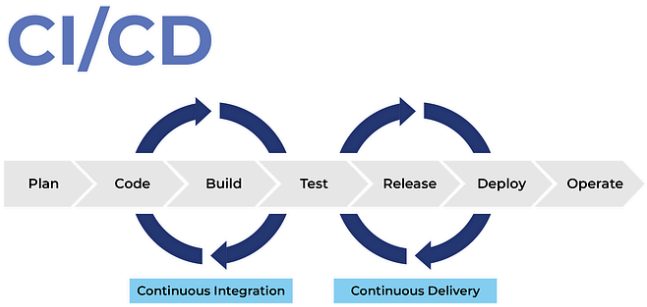
Flutter uygulamarında Birim Test(Unit Test) Süreçleri

Nowadays, everyone is talking about RAG. I want to break it down simply—what RAG is...

Dec 17, 2024 32

Birim test, bir yazılım projesindeki farklı birim veya bileşenlerin test edilebilmesini sağlaya...

Mar 1, 2021 374



In Team Kraken by Osman Yılmaz

Codemagic ile CI/CD Part1

Merhaba, bu hafta Team Kraken olarak CI/CD (Continuous Integration-Continuous Deliver...

Apr 12, 2021 95 1

Osman Yılmaz

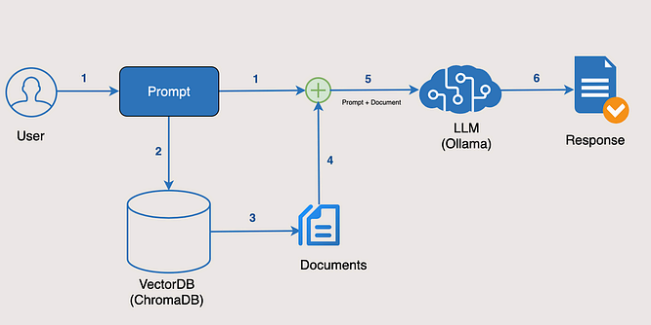
Flutter Drag&Drop

Bu yazımda size aşağıdaki örnek üzerinden Flutterda drag&drop deyimini anlatacağım.

Dec 18, 2023 3

See all from Osman Yılmaz

Recommended from Medium



In Let's Code Future by TheMindShift

10 AI Tools That Replace a Full Dev Team (Almost)

The future isn't coming—it's already here. And it's writing your code, fixing bugs, and...

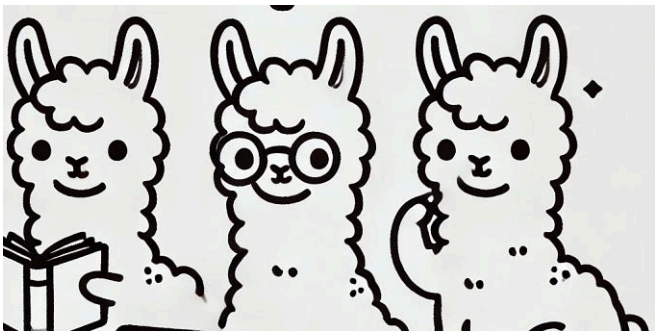
Apr 5 1.2K 52

Arun Patidar

How to Implement RAG with ChromaDB and Ollama: A Python...

Overview of Retrieval-Augmented Generation (RAG)

Dec 10, 2024 79 1

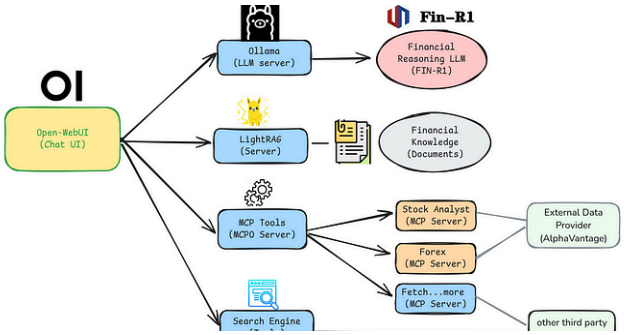


vdcapriles

Simple RAG Application with Ollama and Langgraph

Tempted to use one of the AI models available for work but you can't due to data privacy...

Mar 15 114



Minyang Chen

Awesome Financial Reasoning Workflow (Fin-R1 Model + MCP...

In my previous article, I discussed how LLM agents can be use to emulate the investmen...

Apr 6 107

In Towards Finance by Tinz Twins

Build a Local RAG App to Chat with Earnings Reports

Complete LLM App with Less Than 30 Lines of Python Code (Step-by-Step Guide)

Dec 29, 2024 121

In about ai by Edgar Bermudez

Understanding Embedding Models in the Context of Large Language...

Large Language Models (LLMs) like GPT, BERT, and similar architectures have...

Jan 28 1

See more recommendations