

This member-only story is on us. [Upgrade](#) to access all of Medium.

✦ Member-only story

Enhancing Recommendation Systems with Large Language Models

An implementation with LangChain and Azure OpenAI



Valentina Alto · [Follow](#)

Published in Microsoft Azure · 13 min read · Aug 23, 2023



Large Language Models have shown incredible capabilities in the field of Generative AI, and the potential to unlock new applications in many industries.

An interesting analysis that have also emerged in last months is how LLMs can enhance or even substitute current technological approaches that we use to solve specific tasks. A great example is the field of recommendation systems.

What are recommendation systems

Recommendation systems play a pivotal role in enhancing user experiences by providing personalized suggestions tailored to individual preferences. These systems have become an integral part of various online platforms, from e-commerce and streaming services to social media and news websites. The power of recommendation systems lies in their ability to anticipate user needs, leading to increased engagement, customer satisfaction, and ultimately, business growth.

There are different types of recommendation systems:

1. Collaborative Filtering: it is a fundamental approach that relies on user behavior and preferences to make recommendations. This method identifies similarities between users or items based on their historical interactions. There are two subtypes of collaborative filtering: user-based and item-based. User-based collaborative filtering suggests items that similar users have enjoyed, while item-based collaborative filtering suggests items similar to those the user has shown interest in. Despite its effectiveness, collaborative

filtering can face challenges in handling sparse data and the “cold start” problem for new users or items.

2. Content-Based Filtering: it leverages the intrinsic characteristics of items and users to make suggestions. By analyzing attributes such as text descriptions, genres, and tags, content-based systems create profiles for items and users. These profiles are used to match user preferences with items that share similar attributes. Content-based filtering is particularly useful when dealing with cold start problems for new users, as it doesn't rely solely on historical interactions.

3. Hybrid Recommendation System: it combines multiple techniques to harness the strengths of different approaches. These systems can combine collaborative filtering with content-based filtering to provide more accurate and diverse recommendations. By blending methods, hybrid systems aim to overcome the limitations of individual techniques and offer a more robust solution.

From a technical point of view, there are many techniques that have been employed throughout the years, mainly algorithmic.

Matrix Factorization

Matrix factorization is a technique in recommendation systems that decomposes the user-item interaction matrix into smaller matrices. These matrices represent latent factors that explain user preferences and item attributes. By multiplying these matrices, we predict missing interactions and recommend items to users. This method is efficient for handling sparse data and providing accurate recommendations.

K-Nearest Neighbors

K-Nearest Neighbors (k-NN) is a recommendation technique that suggests items to a user based on the preferences of similar users. It identifies users with similar behavior and recommends items that these users have liked. The “k” in k-NN represents the number of nearest neighbors to consider.

K-NN works by:

1. **Finding Similar Users** → It calculates the similarity between users based on their interactions with items. Users who have similar preferences are considered neighbors.
2. **Selecting “k” Neighbors** → The technique selects the top “k” similar users as neighbors. These users’ liked items are used to make recommendations.
3. **Aggregating Preferences** → It aggregates the preferences of the neighbors, often through weighted averages or other similarity-based methods.
4. **Making Recommendations** → Items liked by the neighbors, but not yet seen by the target user, are recommended.

K-NN is simple to understand and implement, making it a popular choice for collaborative filtering. However, it may face challenges with sparse data and the “cold start” problem for new users or items.

Deep Learning-Based Recommenders

Recent advancements in deep learning have paved the way for the development of neural network-based recommendation systems. These models can learn intricate patterns in user behavior and item attributes, enabling them to provide highly personalized suggestions. Deep learning-

based recommenders often excel in capturing complex relationships and unstructured data, and they are the fore comer of the Large Language Models used as recommenders.

Even though all these methods — especially deep neural networks — have gone though several advancements in last years, they all share a common pitfall: they are task-specific and not very much able to generalize. And here it comes the new AI paradigm of Large Foundation Models and, in our specific case, Large Language Models.

Recent studies

According to recent studies, large language models (LLMs) have shown great potential for improving recommender systems in various aspects, such as:

- LLMs can act as zero-shot rankers for recommendation systems, using their general-purpose task-solving abilities and self-attention mechanisms to rank items based on user preferences and interaction histories.
- LLMs can leverage their high-quality representations of textual features and their extensive coverage of external knowledge to enhance the recommendation quality and diversity.
- LLMs can generate natural language explanations for recommendations with a conversational interface, using their natural language generation capabilities and their ability to incorporate user feedback and contextual information.

All the above make those models adaptable to different scenarios, from cold start to content-based recommendations.

As studied in recent papers (you can see in the references some research papers about this new field of research), there are various ways to make an LLM a recommender, starting from a **pre-training from scratch** (such as the P5, introduced by Shijie Gang et al. in their paper “Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5)”), to **fine-tuning** and, finally, to simple **prompt engineering**. And it will be this last technique we are going to use within our hands-on experiment.

Building an anime recommendation system with LangChain and Azure OpenAI

Let’s now start building our own recommendation system with LangChain, leveraging the embeddings and conversational models of Azure OpenAI.

For this purpose, we are going to use an anime dataset with textual description of each title, so that we can leverage a good chunk of context for our LLMs.

The idea is that of building a chain that is able to suggest some anime to an user who interact for the first time with the system — the so called cold start. Below we will examine the various steps to achieve this goal.

Data Preprocessing

After downloading the dataset from https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020?resource=download&select=anime_with_synopsis.csv and examined its first rows:

```
import pandas as pd

anime = pd.read_csv('data/anime_with_synopsis.csv')
anime.head()
```

	MAL_ID	Name	Score	Genres	synopsis
0	1	Cowboy Bebop	8.78	Action, Adventure, Comedy, Drama, Sci-Fi, Space	In the year 2071, humanity has colonized sever...
1	5	Cowboy Bebop: Tengoku no Tobira	8.39	Action, Drama, Mystery, Sci-Fi, Space	other day, another bounty—such is the life of ...
2	6	Trigun	8.24	Action, Sci-Fi, Adventure, Comedy, Drama, Shounen	Vash the Stampede is the man with a \$\$60,000,0...
3	7	Witch Hunter Robin	7.27	Action, Mystery, Police, Supernatural, Drama, ...	ches are individuals with special powers like ...
4	8	Bouken Ou Beet	6.98	Adventure, Fantasy, Shounen, Supernatural	It is the dark century and the people are suff...

I added a *combined_info* column where I merged all the textual information, so that we can embed only one column of the dataset.

```
anime['combined_info'] = anime.apply(lambda row: f"Title: {row['Name']}. Overview: {row['synopsis']}", axis=1)
anime.head(2)
```

	MAL_ID	Name	Score	Genres	synopsis	combined_info
0	1	Cowboy Bebop	8.78	Action, Adventure, Comedy, Drama, Sci-Fi, Space	In the year 2071, humanity has colonized sever...	Title: Cowboy Bebop. Overview: In the year 207...
1	5	Cowboy Bebop: Tengoku no Tobira	8.39	Action, Drama, Mystery, Sci-Fi, Space	other day, another bounty—such is the life of ...	Title: Cowboy Bebop: Tengoku no Tobira. Overvi...

Then, I've added a column where I counted the number of tokens of each *combined_info* row, to make sure they do not exceed the maximum number of tokens accepted by Azure OpenAI embedding models (8191 tokens). For this purpose, I've used the *cl100k_base* encoding algorithm, the same used by the embedding models we are going to use.



Search



Write



```
max_tokens = 8000 # the maximum for text embedding ada-002 is 8191

encoding = tiktoken.get_encoding(embedding_encoding)

# omit descriptions that are too long to embed
anime["n_tokens"] = anime.combined_info.apply(lambda x: len(encoding.encode(x)))
anime = anime[anime.n_tokens <= max_tokens]
```

Note: LangChain offers several modules for text chunking which are able to segment texts longer than 8000 tokens in an efficient way, so that we do not have to remove records. For simplicity, in this scenario we will proceed only with descriptions shorter than the tokens limit, but if you are interested to learn more about LangChain's text splitters you can read more at <https://docs.langchain.com/docs/components/indexing/text-splitters>.

Embedding and VectorDB

Now that we filtered our dataset, we need to embed our textual description using Azure OpenAI Embeddings models. Generally speaking, embedding refer to the process of converting unstructured pieces of texts into numerical representations in a multi-dimensional space, in such a way that their distance among each others is a measure of their semantic similarity.

Among Azure OpenAI models, the *text-embedding-ada-002* does exactly this, and it is the model we are going to employ. To do so, we need to retrieve our keys and endpoint from our Azure OpenAI service (you can read more about that in my former article [here](#)).


```
import openai
from openai.embeddings_utils import get_embedding

# set the environment variables needed for openai package to know to reach out to
import os

os.environ["OPENAI_API_TYPE"] = "azure"
os.environ["OPENAI_API_BASE"] = "https://<your-endpoint>.openai.azure.com/"
os.environ["OPENAI_API_KEY"] = "your AzureOpenAI key"
os.environ["OPENAI_API_VERSION"] = "2023-05-15"

anime["embedding"] = anime.combined_info.apply(lambda x: get_embedding(x, engine=
anime.head()
```

	MAL_ID	Name	Score	Genres	synopsis	combined_info	n_tokens	embedding
0	1	Cowboy Bebop	8.78	Action, Adventure, Comedy, Drama, Sci-Fi, Space	In the year 2071, humanity has colonized sever...	Title: Cowboy Bebop. Overview: In the year 207...	245	[0.00921056978404522, -0.012633174657821655, 0...
1	5	Cowboy Bebop: Tengoku no Tobira	8.39	Action, Drama, Mystery, Sci-Fi, Space	other day, another bounty—such is the life of ...	Title: Cowboy Bebop: Tengoku no Tobira. Overvi...	199	[-0.008109764195978642, -0.028518257662653923, ...
2	6	Trigun	8.24	Action, Sci-Fi, Adventure, Comedy, Drama, Shounen	Vash the Stampede is the man with a \$\$60,000,0...	Title: Trigun. Overview: Vash the Stampede is ...	252	[0.0019446373917162418, -0.001545737381093204, ...
3	7	Witch Hunter Robin	7.27	Action, Mystery, Police, Supernatural, Drama, ...	ches are individuals with special powers like ...	Title: Witch Hunter Robin. Overview: ches are ...	125	[-0.014938411302864552, 0.007340028416365385, ...
4	8	Bouken Ou Beet	6.98	Adventure, Fantasy, Shounen, Supernatural	It is the dark century and the people are suff...	Title: Bouken Ou Beet. Overview: It is the dar...	188	[0.010889030061662197, 0.0069219209253787994, ...

As you can see, we now have an additional column of vectors. We can then change some columns names (in order to be mapped later on from our chain) and then save the dataframe as pickle.

```
anime.rename(columns = {'embedding': 'vector'}, inplace = True)
anime.rename(columns = {'combined_info': 'text'}, inplace = True)
anime.to_pickle('data/anime.pkl')
```

Finally, we want to store our embeddings into a VectorDB, so that we can perform similarity search with the embedded query of the user. LangChain

offers many integrations with 3rd party vector stores and, in this case, we are going to use LanceDB:

```
import lancedb
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import LanceDB
from langchain.chains import RetrievalQA

uri = "dataset/sample-anime-lancedb"
db = lancedb.connect(uri)
table = db.create_table("anime", anime)

embeddings = OpenAIEmbeddings(engine="text-embedding-ada-002")

docsearch = LanceDB(connection = table, embedding = embeddings)
```

To test our vector store, let's start with a simple similarity computation which returns the first most similar result using cosine similarity as distance metric:

```
query = "I'm looking for an animated action movie. What could you suggest to me?"
docs = docsearch.similarity_search(query, k=1)
docs
```

Output:

```
[Document(page_content='Title: Marin X. Overview: Korean animated movie mixing Mecha with anti-communist themes. Genres: Action, Adventure, Mecha, Sci-Fi, Shounen', metadata={'MAL_ID': 16784, 'Name': 'Marin X', 'Score': 'Unknown', 'Genres': 'Action, Adventure, Mecha, Sci-Fi, Shounen',
```

```
'synopsis': 'Korean animated movie mixing Mecha with anti-communist themes.', 'n_tokens': 35, 'vector': array([-0.01003191,  0.00912034, 0.0071506 , ...,  0.01442947, -0.00128262, -0.00605121], dtype=float32), '_distance': 0.3988204002380371}]]
```

As you can see, the retrieved document exhibit a `page_content`, that will be the context passed to our LLM, alongside some metadata which are the other variables within the dataset.

Now that we have all the ingredients, we can start building our chain.

Building the chain

For this purpose, we are going to use LangChain [RetrievalQA](#) chain, alongside the LanceDB as retriever.

```
# Import Azure OpenAI
from langchain.llms import AzureOpenAI

qa = RetrievalQA.from_chain_type(llm=AzureOpenAI(deployment_name="text-davinci-002"), retriever=retriever)

query = "I'm looking for an action anime. What could you suggest to me?"
result = qa({"query": query})
result['result']
```

```
' You could watch Kaitouranma The Animation, Kikaider 01 The Animation, SMAnime, or De:vadasy.'
```

Since we used the `return_source_documents=True` parameter, we can also retrieve the source documents from our `result` variable:

```
result['source_documents'][0]
```

```
Document(page_content="Title: Kaitouranma The Animation. Overview: Shinjuro may be a little young to be the master of a swordsmanship school, but fate didn't give him a choice. Now his position and martial arts skill land him in the middle of serious trouble when the rogue samurai Mikage faces off against the Tokugawa government, and Shinjuro and the small group of swordsmen who train at the school must now protect the town from harm. However, the group won't be challenged until the Tokugawa house calls in a favor, and the young swordsman is forced to duel Mikage himself. (Source: AniDB) Genres: Action, Adventure, Martial Arts, Samurai", metadata={'MAL_ID': 2483, 'Name': 'Kaitouranma The Animation', 'Score': '5.99', 'Genres': 'Action, Adventure, Martial Arts, Samurai', 'synopsis': "Shinjuro may be a little young to be the master of a swordsmanship school, but fate didn't give him a choice. Now his position and martial arts skill land him in the middle of serious trouble when the rogue samurai Mikage faces off against the Tokugawa government, and Shinjuro and the small group of swordsmen who train at the school must now protect the town from harm. However, the group won't be challenged until the Tokugawa house calls in a favor, and the young swordsman is forced to duel Mikage himself. (Source: AniDB)", 'n_tokens': 146, 'vector': array([-0.01823905, 0.00753973, 0.03045369, ..., -0.00741998, 0.01266601, -0.01191987], dtype=float32), '_distance': 0.35099881887435913})
```

Another nice thing about the retriever is that we customize its *kwargs*. For example, let's say we are only interested in anime that, among their genres, are tagged as "Action". To do so, we can proceed as follows:

```
df_filtered = anime[anime['Genres'].apply(lambda x: 'Action' in x)]
qa = RetrievalQA.from_chain_type(llm=AzureOpenAI(deployment_name="text-davinci-001", model_name="text-davinci-003", openai_api_key=openai_api_key, openai_api_version="2022-11-01"))
```

```
retriever=docsearch.as_retriever(search_kwargs={'data': df_filtered}), return

query = "I'm looking for an anime with animals and an adventurous plot."
result = qa({"query": query})
```

We could also filter directly at metadata level using the following code (this time, imagining that we want only to see anime with rating above 7):

```
qa = RetrievalQA.from_chain_type(llm=AzureOpenAI(deployment_name="text-davinci-003",
model_name="text-davinci-003", openai_api_key=openai_api_key, openai_api_version=
retriever=docsearch.as_retriever(search_kwargs={'filter': {'score__gt':7})),
```

Insofar, we didn't customized our prompt, but to have better result it is important to tailor our LLMs to be a recommender.

Prompt engineering

With RetrievalQA, you can pass a custom prompt that you can easily define using LangChain prompt templates. Let's start with a simple prompt as follows:

```
from langchain.prompts import PromptTemplate

template = """You are a movie recommender system that help users to find anime to watch.
Use the following pieces of context to answer the question at the end.
For each question, suggest three anime, with a short description of the plot and
If you don't know the answer, just say that you don't know, don't try to make up

{context}"""
```

Question: {question}

Your response:""

```
PROMPT = PromptTemplate(
    template=template, input_variables=["context", "question"])

chain_type_kwargs = {"prompt": PROMPT}

llm=AzureOpenAI(deployment_name="text-davinci-003",
model_name="text-davinci-003", openai_api_key=openai_api_key, openai_api_version=

qa = RetrievalQA.from_chain_type(llm=llm,
    chain_type="stuff",
    retriever=docsearch.as_retriever(),
    return_source_documents=True,
    chain_type_kwargs=chain_type_kwargs)

query = "I'm looking for an action anime with animals, any suggestions?"
result = qa({'query':query})
print(result['result'])
```

1. Urikupaen Kyuujo-tai: This adventure comedy follows a team of brave young animals that rescues others in peril. With a dog, a boar, a deer, a koala, a mouse, a seagull, and a lion, this show is sure to please those looking for an action anime featuring animals.
2. Nekketsu Jinmen Inu: Life Is Movie: This parody follows a passionate human-faced dog NEET/would be detective in his adventures. Fans of action anime with animals are sure to be engaged by this mystery story.
3. Daisetsusan no Yuusha Kibaou: The main character of this drama is Fang, who was born to a hunting dog and a circus-runaway European wolf. Fang returns from the circus to face his foe, a giant brown bear which killed his family, making this story a great pick for those looking for an action anime with animals.

As you can see, thanks to our prompt the model only responded with three suggested anime. We can go even further and provide the prompt with dynamic parameters, given by some preliminary information that the

system might ask to the user before starting the conversation. To do so, we will split the prompt into three chunks, in order to format only the one containing the dynamic parameters:

```
from langchain.prompts import PromptTemplate

template_prefix = """You are a movie recommender system that help users to find
Use the following pieces of context to answer the question at the end.
For each question, take into account the context and the personal information pr
If you don't know the answer, just say that you don't know, don't try to make up

{context}"""

user_info = """This is what we know about the user, and you can use this informa
Age: {age}
Gender: {gender}"""

template_suffix= """Question: {question}
Your response: """

user_info = user_info.format(age = 18, gender = 'female')

COMBINED_PROMPT = template_prefix + '\n' + user_info + '\n' + template_suffix
print(COMBINED_PROMPT)
```

You are a movie recommender system that help users to find anime that match their preferences.
Use the following pieces of context to answer the question at the end.
For each question, take into account the context and the personal information provided by the user.
If you don't know the answer, just say that you don't know, don't try to make up an answer.

{context}
This is what we know about the user, and you can use this information to better tune your research:
Age: 18
Gender: female

Question: {question}

Your response:

Let's see it in action:

```
PROMPT = PromptTemplate(
    template=COMBINED_PROMPT, input_variables=["context", "question"])

chain_type_kwargs = {"prompt": PROMPT}
qa = RetrievalQA.from_chain_type(llm=OpenAI(),
    chain_type="stuff",
    retriever=docsearch.as_retriever(),
    return_source_documents=True,
    chain_type_kwargs=chain_type_kwargs)

query = "Can you suggest me some action movie?"
result = qa({'query':query})
result['result']
```

Based on the information you provided, I suggest Urikupaen Kyuujo-tai. It is an adventure, comedy and kids anime about four young animals, a rabbit, squirrel, bear and penguin, who form a team to rescue others in peril. It is a limited broadcast show on some American local TV stations for the Japanese community. Another suggestion is Daisetsusan no Yuusha Kibaou, an adventure and drama anime about a wolf born to a hunting dog and European wolf. It follows Fang as he returns from the circus to face a giant brown bear that killed his family. Lastly, I suggest Ookami wa Ookamida, a fantasy anime about village animals struggling against a tyrannical wolf.

Great! Now the model took into account all the info and, in an hypothetical recommendation application, it could ask the user some details to be accounted within the prompt or the context.

If you want to run the code on your own, you can check my GitHub repo at <https://github.com/Valentina-Alto/Recommendation-systems-with-LLMs>.

Conclusions and further improvements

Using LLMs to enhance recommendation systems is an emerging field of research which is paving the way to great experiments. In this article, we saw a very simple implementation using a simple prompt engineering to tailor an LLM to be a recommender, however there are more refined techniques that are being further explored, such as pre-training and fine-tuning. Furthermore, we analyzed a cold start scenario, however LLMs could also face content-based scenarios where we already know some information about the users we are interacting with.

Overall, LLMs as recommenders are promising and I'm curious to see what discoveries will reserve to us the upcoming months 🚀

References

- https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020?resource=download&select=anime_with_synopsis.csv
- [2305.08845] [Large Language Models are Zero-Shot Rankers for Recommender Systems \(arxiv.org\)](#).
- [2307.02046] [Recommender Systems in the Era of Large Language Models \(LLMs\) \(arxiv.org\)](#).
- [2305.19860] [A Survey on Large Language Models for Recommendation \(arxiv.org\)](#).
- <https://medium.com/microsoftazure/azure-openai-and-langchain-eba69f18f050>

Large Language Models

OpenAI

Recommendation System

Langchain

AI



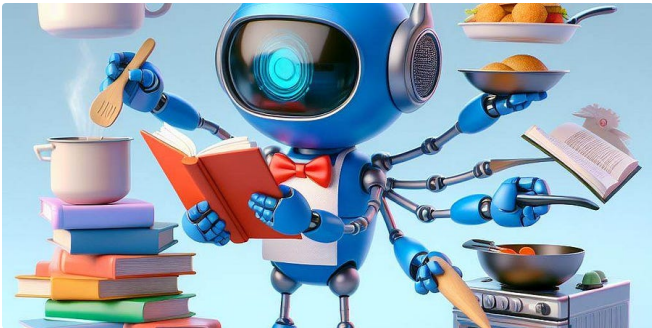
Written by Valentina Alto

4.3K Followers · Writer for Microsoft Azure

Follow

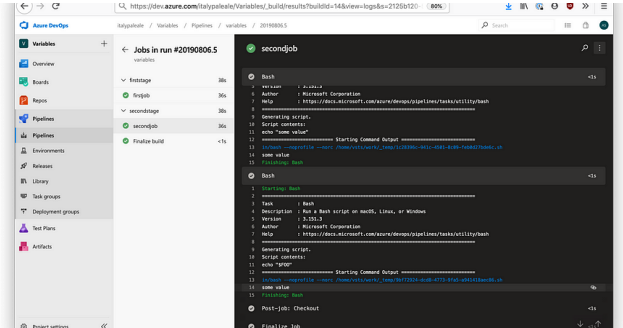
Data&AI Specialist at @Microsoft | MSc in Data Science | AI, Machine Learning and Running enthusiast

More from Valentina Alto and Microsoft Azure



 Valentina Alto in Towards Data Science

Getting Started with Multimodality
Understanding vision capabilities of Large Multimodal Models



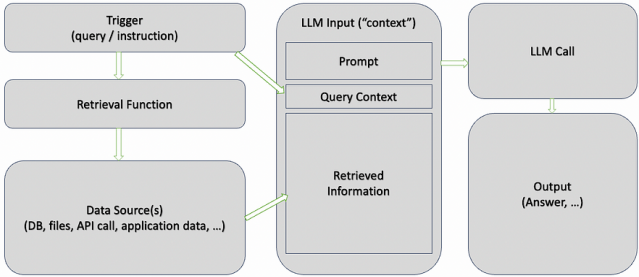
 Alessandro Segala in Microsoft Azure

How to pass variables in Azure Pipelines YAML tasks
Passing variables between steps, jobs, and stages: explained

★ · 9 min read · Dec 28, 2023

 205  3  

Simple RAG Model



 Saverio Proto in Microsoft Azure


Understanding how the Azure OpenAI “use your data” feature...

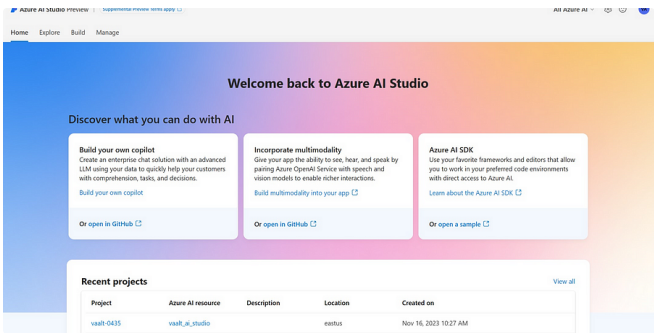
As of September 2023, Azure OpenAI has a preview feature called “Use your data.” To...

5 min read · Sep 27, 2023

 71  1  

5 min read · Aug 5, 2019

 496  9  




 Valentina Alto in Microsoft Azure

Getting started with Azure AI Studio

Overview and first RAG Application

★ · 5 min read · Nov 17, 2023

 102  1  

See all from Valentina Alto

See all from Microsoft Azure

Recommended from Medium



Rabie Rh

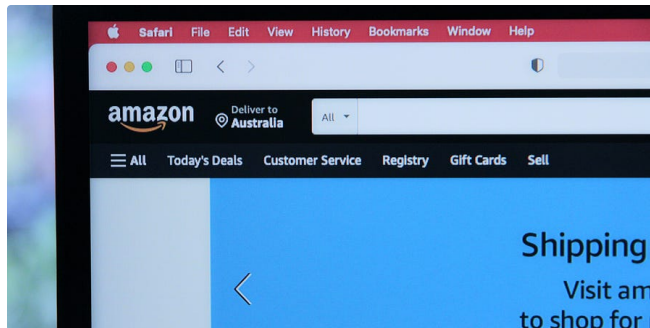
Building a Recommendation Engine with GPT-3 and...

Recommendation engines have become a staple of our online experiences, from...

4 min read · Sep 5, 2023



58



Mehul Gupta in Data Science in your pocket

Recommendation Systems using Langchain and LLMs with codes

using RAG framework and chains

4 min read · Oct 15, 2023



268



3



Lists



Natural Language Processing

1072 stories · 549 saves



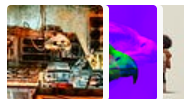
Generative AI Recommended Reading

52 stories · 596 saves



AI Regulation

6 stories · 265 saves



What is ChatGPT?

9 stories · 268 saves

