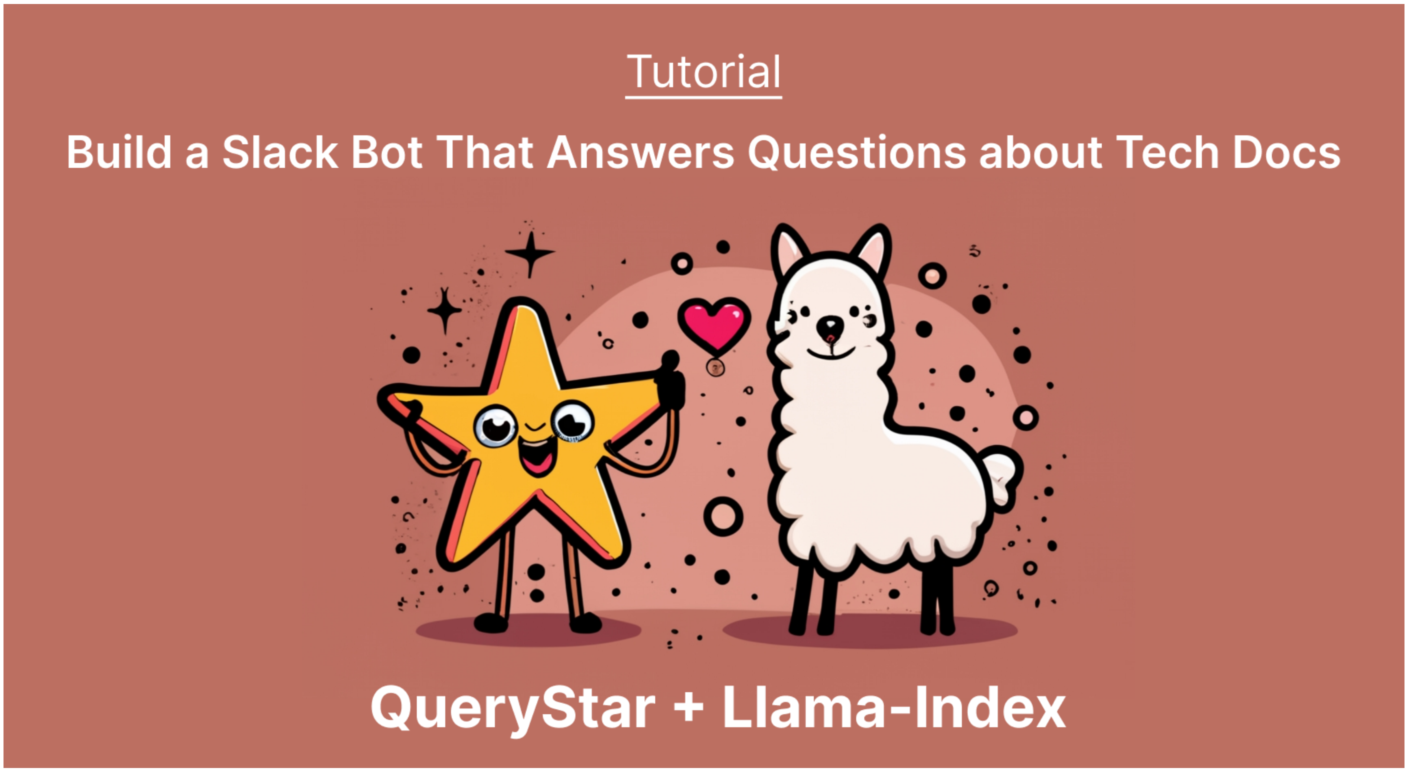
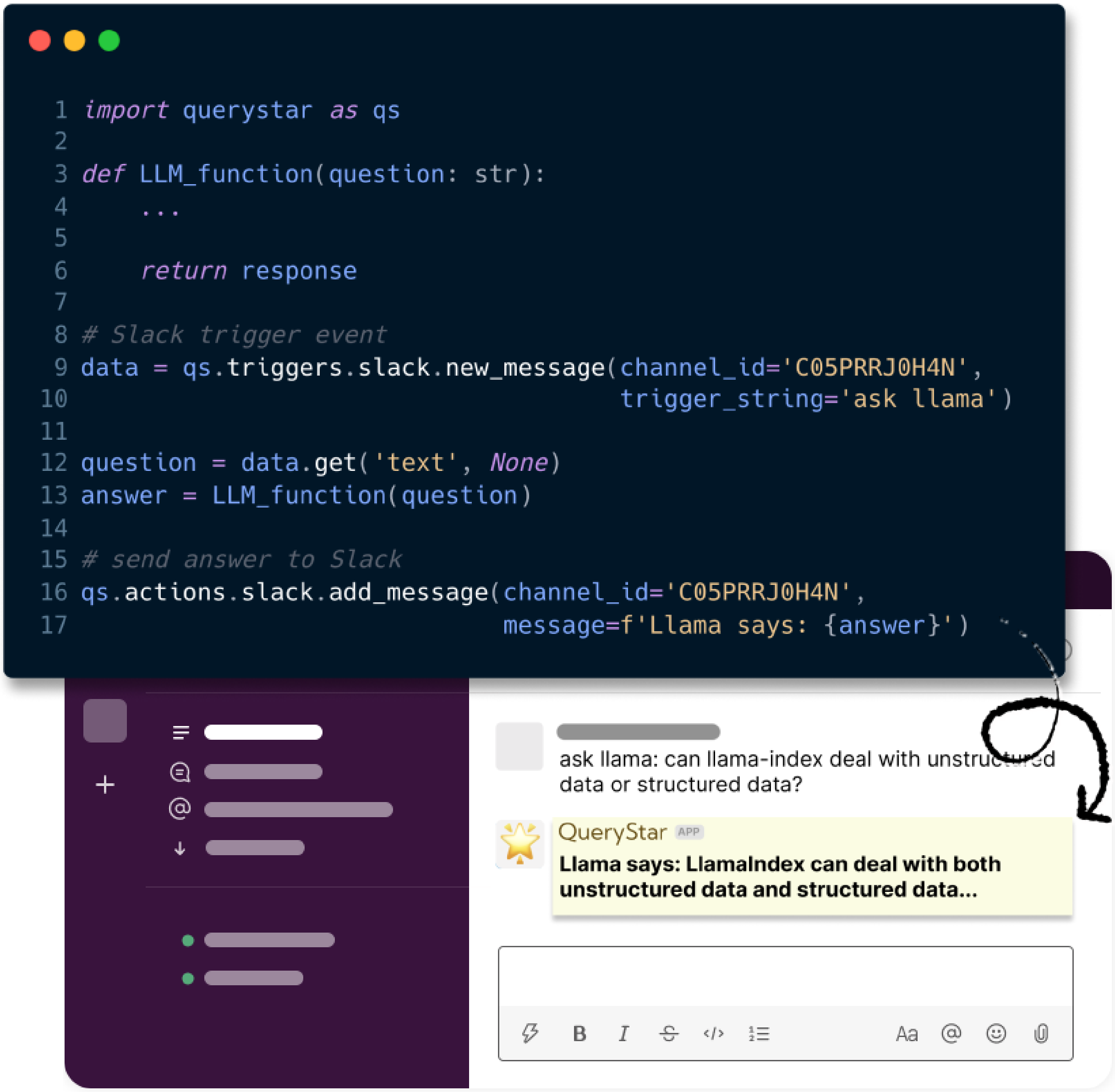
QueryStar 🌟 + LlamaIndex 🦙: Slack Bot that Understands Your Data (Draft)



Objective

Learn how to build a Slack bot that can answer questions about your own documents.



```python
import querystar as qs

def LLM_function(question: str):
    ...

    return response

# Slack trigger event
data = qs.triggers.slack.new_message(channel_id='C05PRRJ0H4N',
                                     trigger_string='ask llama')

question = data.get('text', None)
answer = LLM_function(question)

# send answer to Slack
qs.actions.slack.add_message(channel_id='C05PRRJ0H4N',
                             message=f'Llama says: {answer}')
```

ask llama: can llama-index deal with unstructured data or structured data?

QueryStar APP

Llama says: LlamaIndex can deal with both unstructured data and structured data...

Usefulness: ⭐⭐⭐⭐⭐ | Difficult Level: ⭐⭐

> ⓘ **NOTE**
>
> Source code: https://github.com/modelstar-labs/querystar-demo/tree/main/ask_llamaindex_slack_bot

About the Slack Bot

This bot helps Slack users to learn a new open source project, called `LlamaIndex`. It "learns" from the project documentation and can "answer" questions about it.

Use Case Example

When a user posts a message (starting with "ask llama") to `#ask-llama` channel in a Slack workspace:

> "ask llama: can llama-index ...?"

The bot sends a legit answer back to the channel:

> Llama says: ...

Module Design

- **AI (LLM) Function**: After getting a question, this function should generate an answer that aligns with the context provided in the given documents.
- **Trigger - Action**:
  - The bot should respond to messages that:
    - are sent to a designated channel, AND
    - contain trigger word "ask llama".
  - The bot should extract user question from the trigger message, and run the LLM function to generate an answer.
  - The bot should post the answer back to the channel. Then wait for future trigger events.

Tech Stack

We want to maximize speed of learning and shipping, meanwhile, leave enough room for customization. Here, we choose the `LOQ` stack (`LlamaIndex` + `OpenAI` + `Querystar`). This allows us to make a fully functioning bot in < 1 hour. Let's introduce the stack by module.

`LlamaIndex` for the LLM Function

Key capability of the function is to process documents and retrieve relevant context. We will implement this function using `Retrieval Augmented Generation` (`RAG`). The basic concepts of `RAG` are explained in Section 1: Some Basics.

`LlamaIndex` and `OpenAI`'s LLMs give us great results with the least amount of code. Details are in Section 2: Data Indexing.

> ⓘ **NOTE**
>
> If you want to skip Section 1 and 2: this Jupyter notebook shows how the AI function is implemented.

`QueryStar` for the Bot

This module determines the bot's behavior, and provides an intuitive interface for human to use AI.

`QueryStar` is used here to implement the design, with only 2 simple function calls. See details in Section 3: Bot development).

> ⓘ **NOTE**
>
> For those who are curious: this `app.py` file (24 lines of Python code) is all we need to ship the bot.

`RAG` Basics

When dealing with questions, we often need some reference materials to help to find answers. In this process, we retrieve paragraphs/context that are *relevant* to the questions.

A big challenge we will be facing if we want a computer program to do this: **How to quantify and mathematically measure `relevancy` between any two pieces of information in the format of human language.**

We must find a mathematical representation of text (words, sentences and paragraphs), and construct a measurement (as a proxy to `relevancy`) among the representations.

A technique, called `Embedding`, is widely adopted to transform texts to vectors. With embedding vectors, we can use distance between them to quantify `relevancy`.

A basic `Retrieval Augmented Generation` (`RAG`) algorithm can be implemented through the following steps:

1. For any document, we divide the content into chunks. E.g., every 3 sentences, or every 100 words. Let's say we got 500 chucks after this step.
2. We create `embeddings` for each chunk, which gives us 500 "context vectors".
3. With any given question, we can get its embedding as well, which gives us 1 "question vector".
4. Use the question vector to compare with the 500 context vectors, and select top N (e.g., 5) most similar ones. Then we believe all 5 of them are **relevant** to the question.
5. We put the top 5 **relevant** text chunks in a prompt along with the question, then ask LLMs "answer it only based on the given context, not other prior knowledge".

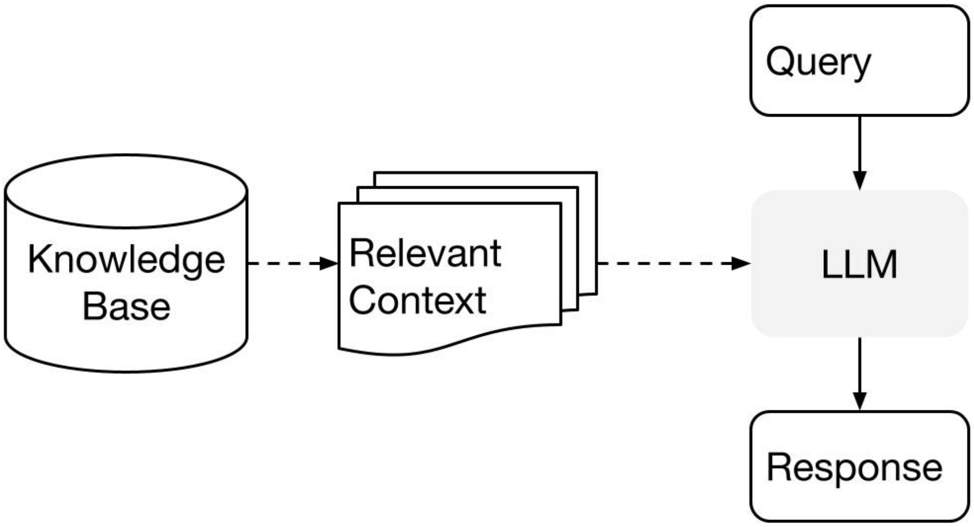


Image Source: LlamaIndex Docs

In contrast to solely sending the question to LLMs for query, `RAG` sends both question and relevant context, which can help to reduce hallucinations and improve accuracy.

However, implementing all 5 steps on ourselves is not a simple task. This is where `LlamaIndex` comes in handy. It allows us to build a nice `RAG` pipeline with a few function calls.

> ⓘ **NOTE**
>
> What makes `LlamaIndex` more appealing to experienced engineers is that it provides many low-level APIs as well for customization.

Interested in giving it a try? Let's dive into the coding process.

`RAG` Function

Prep: Cleaning

Let's start with downloading/cloning LlamaIndex's doc folder to a local folder `ask_llamaindex_slack_bot`.

There're many types of files in the folder. Some files (like `.py` or `.bat`) do not contain too much context. So, we only want to keep `.md` and `.rst` files:

```
# Data cleaning
dir_path = './lidocs'  #local folder of LlamaIndex
documentations
for root, dirs, files in os.walk(dir_path):
    for file in files:
        if file.endswith(('.md', '.rst')):
            continue
        else:
            file_path = os.path.join(root, file)
            os.remove(file_path)
```

Once the doc folder is clean, it's very convenient to use `SimpleDirectoryReader` function to load the entire folder to `lidocs` object at once.

```
from llama_index import SimpleDirectoryReader

reader = SimpleDirectoryReader(input_dir="./lidocs",
recursive=True)
lidocs = reader.load_data()
```

Prep: Indexing

Now we're ready to build `index`: dividing each document into chunks and embedding them. `LlamaIndex` has a great API for this: `VectorStoreIndex.from_documents()`. Then we store `index` in files.

```
import openai
from llama_index import VectorStoreIndex

openai.api_key = os.getenv('OPENAI_API_KEY')
index = VectorStoreIndex.from_documents(lidocs)

# save index to files
index.storage_context.persist()
```

The `storage` folder automatically appears:

```
├── ask_llamaindex_slack_bot
│   ├── storage
│   │   ├── docstore.json
│   │   ├── graph_store.json
│   │   ├── index_store.json
│   │   └── vector_store.json
```

All the `embedding` vectors are saved in `vector_store.json`. The file size is 33 MB, which contains a mathematical presentation of the entire LlamaIndex's documentation.

> **INFO**
>
> In this step, we use GPT, a commercial model service by OpenAI.
>
> - Before building the index, make sure you have an OpenAI API key. This step may cost you up to $1 for GPT tokens. To avoid the cost, you can skip this step and download the `storage` folder from QueryStar demo repo.
> - `VectorStoreIndex.from_documents()` call may take 2-5 mins to finish, highly depending on API latency in your region.

Function Development

With the index file in place, we can finally build the `RAG` function. Again, it's simple with `LlamaIndex`. We just need to load them to the `index` object, and use the built-in query engine to get `response`:

```python
def ask_llamaindex(question: str):

    # rebuild storage context
    storage_context =
StorageContext.from_defaults(persist_dir="./storage")
    # load index
    index = load_index_from_storage(storage_context)
    query_engine = index.as_query_engine()
    response = query_engine.query(question)

    return response
```

Now, let's build the bot!

Trigger-Action Based Slack Bot

Prep: QueryStar Setup

QueryStar is a low code Python package to simplify `Trigger-Action` based bot development. It shares the same design philosophy behind Streamlit, PyWebIO , Gradio, Greppo: Making it super easy for data teams and Python developers to ship interfaces between human and AI/data. These four projects are used for Web UI, while Querystar is intended for bot development (more in the introduce page).

Before running any code in this module, please make sure you already got a QueryStar token, installed the library, and can run the `hello world` slackbot. The setup process should only take you less than 10 mins.

INFO

> QueryStar automatically integrate 3rd party API services which also include Slack authorization. So, we do **NOT** need a Slack token here.
>
> QueryStar token is free for 1 Slack workspace connection and unlimited bots in that workspace.

Prep: Creating `app.py`

With QueryStar, a bot can be developed out of a single py file. Let's create a file called `app.py` in `ask_llamaindex_slack_bot` folder, imports some packages, and copy the `ask_llamaindex()` function here.

```
# app.py
import querystar as qs
from llama_index import StorageContext,
load_index_from_storage
import os, openai

openai.api_key = os.getenv('OPENAI_API_KEY')

def ask_llamaindex(question: str):

    # rebuild storage context
    storage_context =
StorageContext.from_defaults(persist_dir="./storage")
    # load index
    index = load_index_from_storage(storage_context)
    query_engine = index.as_query_engine()
    response = query_engine.query(question)
    return response
```

`new_message()` Trigger

Let's recap how we designed the trigger in the beginning of the tutorial:

> - The bot should respond to messages that:
>   - are sent to a designated channel, AND
>   - contain trigger word "ask llama".

This Slack message `trigger` can be easily done with `triggers.slack.new_message()` function:

```
data =
qs.triggers.slack.new_message(channel_id='C05PRRJ0H4N',
# channel: llama-qa

trigger_string='ask llama')
```

This script is quite self-explanatory. The bot is set to listen to new Slack messages. When a message matches the filter condition (`channel_id` *AND* `trigger_string`), a `json` object of the message will return to variable `data`.

> ⓘ **NOTE**

▸ An example message object (click to expand)

`add_message()` Action

This is what we designed for the actions:

> - The bot should extract user question from the trigger message, and run the LLM function to generate an answer.
> - The bot should post the answer back to the channel. Then, wait for future trigger events.

Let's do it:

```
question = data.get('text', None)
answer = ask_llamaindex(question)
# send answer to Slack
qs.actions.slack.add_message(channel_id='C05PRRJ0H4N',
message=f'Llama says: {answer}')
```

We first parse the trigger message to get `question`, which will be passed to `ask_llamaindex()` to generate an answer. Then, we use `actions.slack.add_message()` to post the answer back to the same channel.

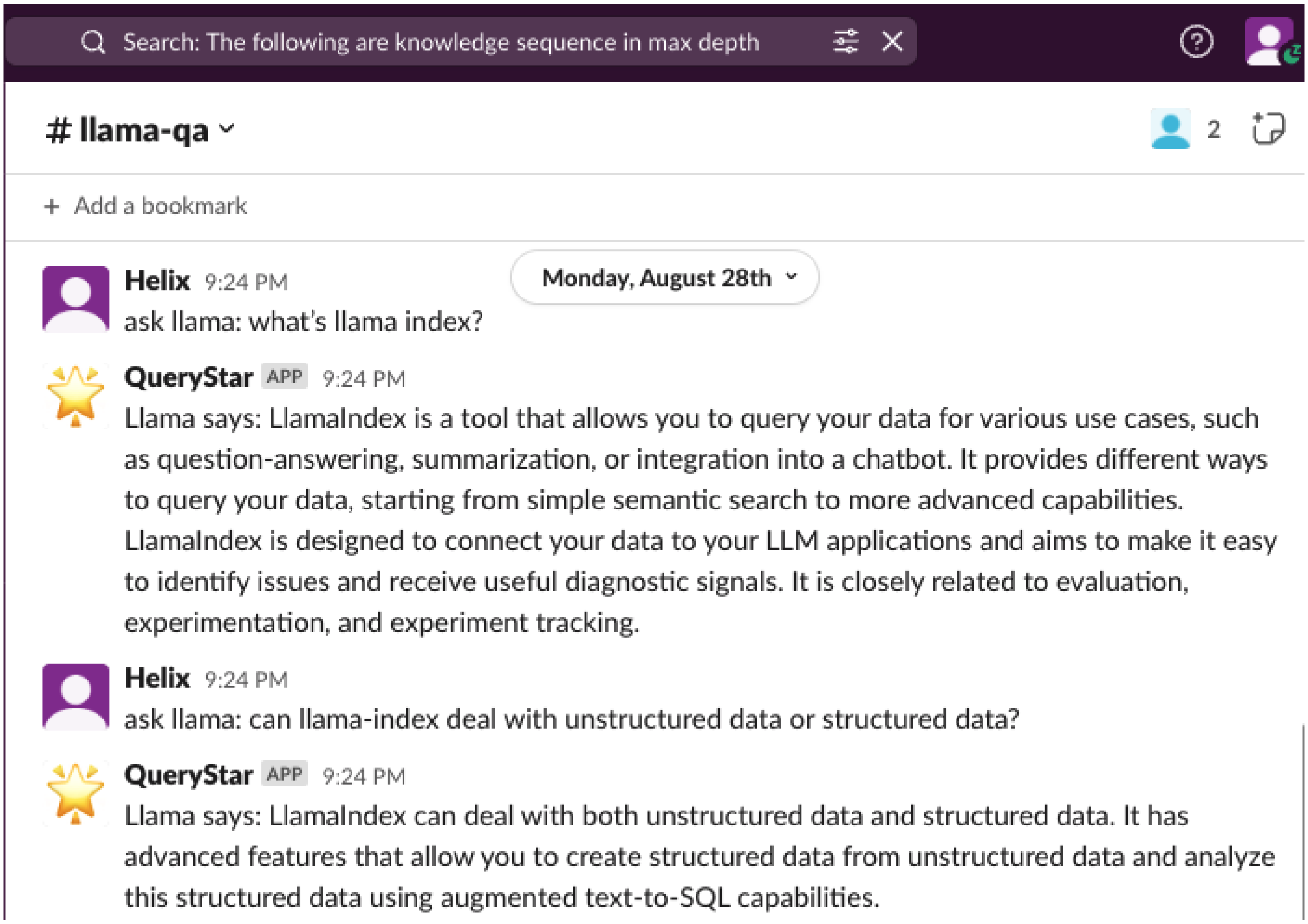
That's all. Let test it!

End-to-end Test

Open your terminal, run this command in `ask_llamaindex_slack_bot` folder:

```
$ querystar run app.py
```

Go to Slack and post a trigger message. It works 🤩 🤩 🤩



Helix 9:24 PM
ask llama: what's llama index?

QueryStar APP 9:24 PM
Llama says: LlamaIndex is a tool that allows you to query your data for various use cases, such as question-answering, summarization, or integration into a chatbot. It provides different ways to query your data, starting from simple semantic search to more advanced capabilities. LlamaIndex is designed to connect your data to your LLM applications and aims to make it easy to identify issues and receive useful diagnostic signals. It is closely related to evaluation, experimentation, and experiment tracking.

Helix 9:24 PM
ask llama: can llama-index deal with unstructured data or structured data?

QueryStar APP 9:24 PM
Llama says: LlamaIndex can deal with both unstructured data and structured data. It has advanced features that allow you to create structured data from unstructured data and analyze this structured data using augmented text-to-SQL capabilities.

Now, you can invite your entire team join this channel and learn `LlamaIndex` together!

> *Acknowledgement: Thanks to Yi Ding (Head of Typescript and DevRel @LlamaIndex) for proofreading! This tutorial is inspired by Build a*

> *chatbot with custom data sources, powered by LlamaIndex by `Streamlit` and `LlamaIndex` teams.*

*Last updated on **Oct 14, 2018***

(Simulated during dev for better perf)