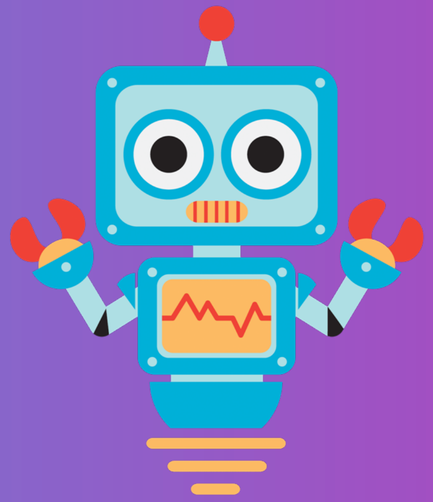
Dr. Abulkarim Albanna

Langchain

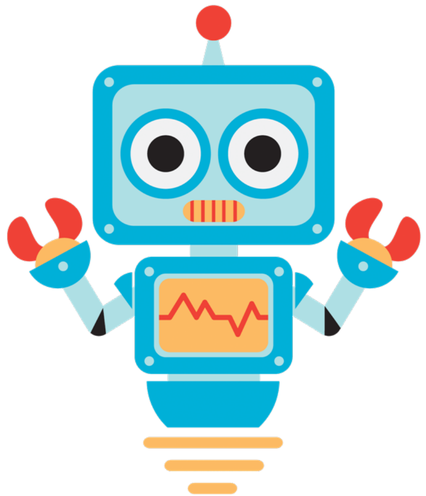
Day 5

January 23th 2024



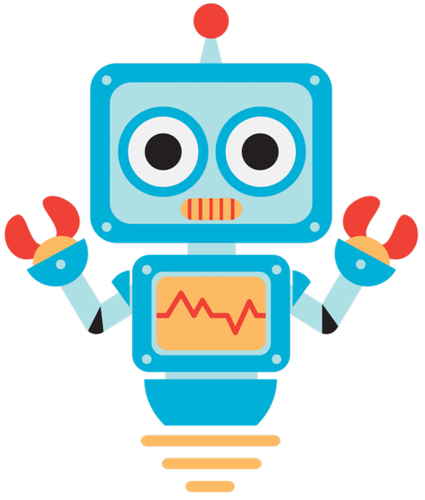
Content

- RouterChain
- Chat with your documents

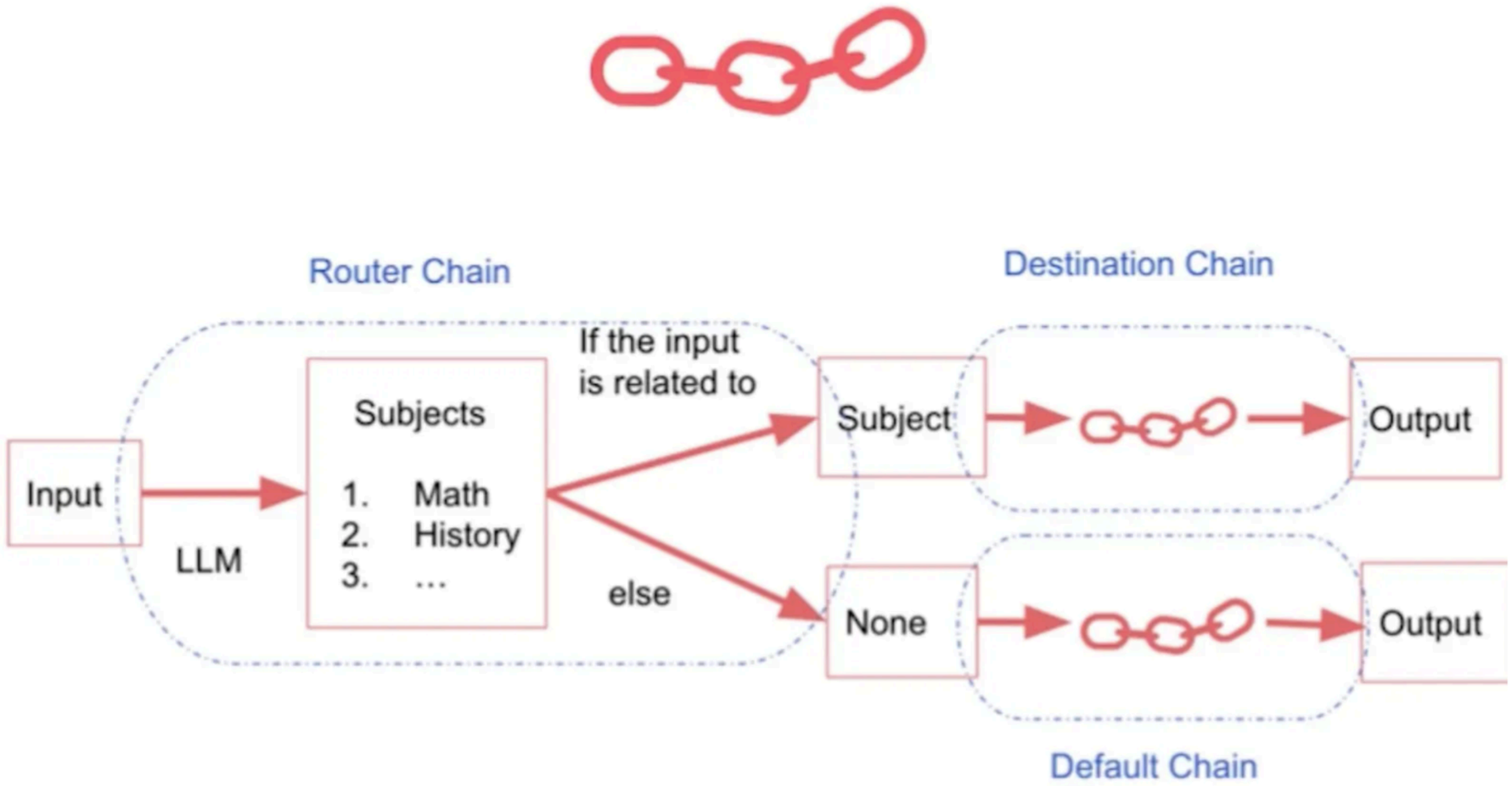


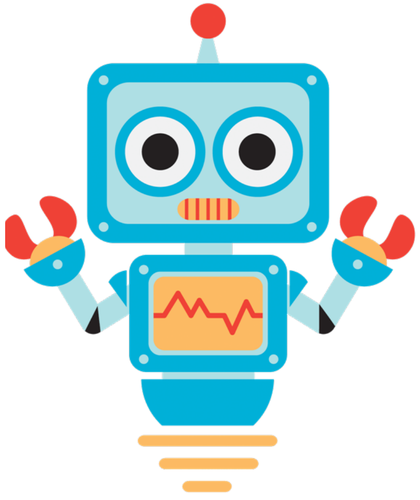
RouterChain

- RouterChain is used for **complicated** tasks. For example, a pretty common but basic operation is to route an input to a chain depending on what exactly that input is. If we have **multiple subchains**, each of which is specialized for a particular type of input, we could have a router chain that decides which subchain to pass the input to. For example, we can route between multiple subchains depending on each specialized for a particular type of input.



RouterChain





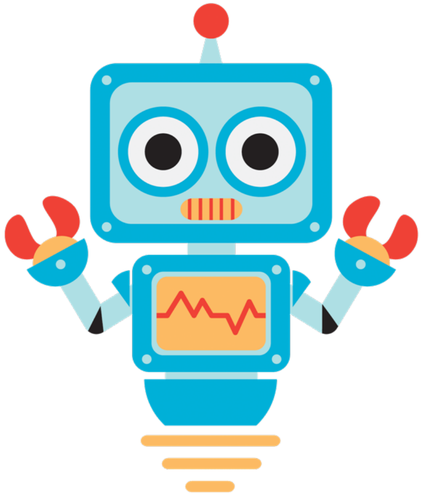
RouterChain

To achieve this goal Step (1)

- Define prompt templates for these subjects.

- Provide more information about these prompt templates as well, for example, we can give each template a **name** and a **description**.

- Pass this information to the router chain and the router chain can decide when to use which subchain.



RouterChain

```
physics_template = """You are a very smart physics professor. \
You are great at answering questions about physics in a concise\
and easy to understand manner. \
When you don't know the answer to a question you admit\
that you don't know.

Here is a question:
{input}"""


math_template = """You are a very good mathematician. \
You are great at answering math questions. \
You are so good because you are able to break down \
hard problems into their component parts,
answer the component parts, and then put them together\
to answer the broader question.

Here is a question:
{input}"""

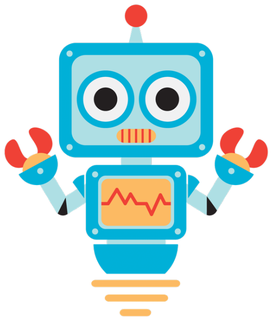
history_template = """You are a very good historian. \
You have an excellent knowledge of and understanding of people,\
events and contexts from a range of historical periods. \
You have the ability to think, reflect, debate, discuss and \
evaluate the past. You have a respect for historical evidence\
and the ability to make use of it to support your explanations \
and judgements.

Here is a question:
{input}"""


computerscience_template = """ You are a successful computer scientist.\
You have a passion for creativity, collaboration,\
forward-thinking, confidence, strong problem-solving capabilities,\
understanding of theories and algorithms, and excellent communication \
skills. You are great at answering coding questions. \
You are so good because you know how to solve a problem by \
describing the solution in imperative steps \
that a machine can easily interpret and you know how to \
choose a solution that has a good balance between \
time complexity and space complexity.

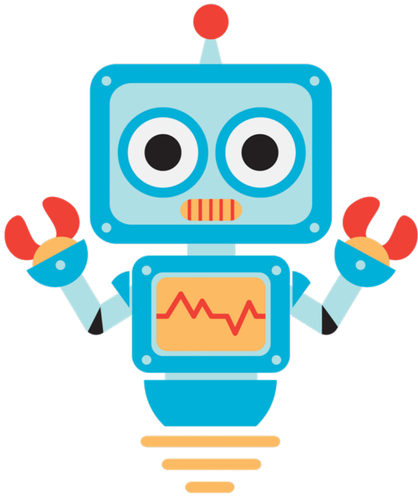
Here is a question:
{input}"""
```

```
prompt_infos = [
    {
        "name": "physics",
        "description": "Good for answering questions about physics",
        "prompt_template": physics_template
    },
    {
        "name": "math",
        "description": "Good for answering math questions",
        "prompt_template": math_template
    },
    {
        "name": "History",
        "description": "Good for answering history questions",
        "prompt_template": history_template
    },
    {
        "name": "computer science",
        "description": "Good for answering computer science questions",
        "prompt_template": computerscience_template
    }
]
```



RouterChain

To achieve this goal Step (2)



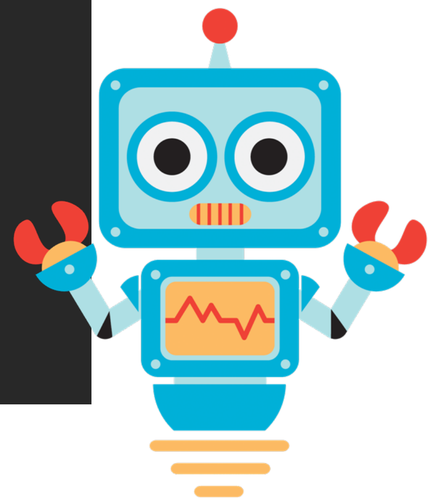
- **Create a multi-prompt chain**, which is a specific type of chain that is used when routing between multiple different prompt templates. We also need **LLMRouterChain**, which uses a language model to route between different subchains.

RouterChain

To achieve this goal Step (2)

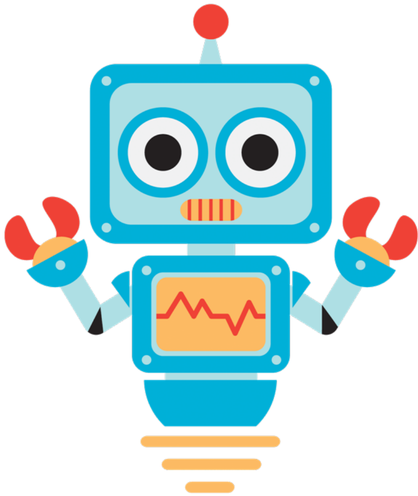
```
4    from langchain.chat_models import ChatOpenAI
5    from langchain.prompts import ChatPromptTemplate
6    from langchain.chains import LLMChain
```

```
83   destination_chains = {}
84   for p_info in prompt_infos:
85       name = p_info["name"]
86       prompt_template = p_info["prompt_template"]
87       prompt = ChatPromptTemplate.from_template(template=prompt_template)
88       chain = LLMChain(llm=llm, prompt=prompt)
89       destination_chains[name] = chain
90
```



RouterChain

To achieve this goal Step (3)

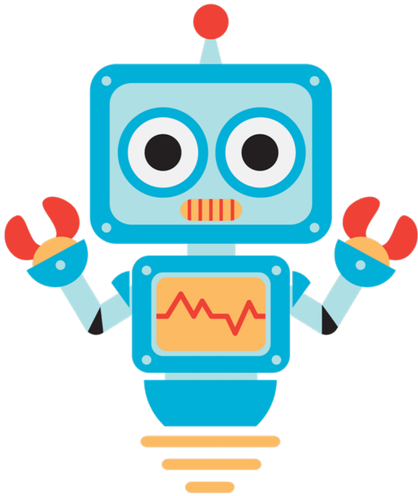


- **Create destination chains**. Destination chains are chains that get called by RouterChain. Each destination chain itself is a language model chain, an LLM chain.

- **Define a default chain**, which gets called when the router can’t decide which of the subchains to use.

RouterChain

To achieve this goal Step (3)

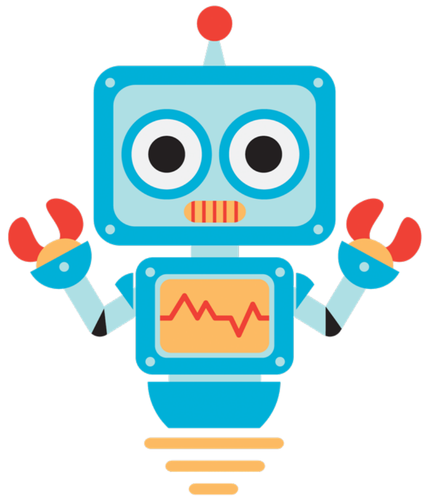


```
88  destinations = [f"{p['name']}: {p['description']}" for p in prompt_infos]
89  destinations_str = "\n".join(destinations)
90
91  default_prompt = ChatPromptTemplate.from_template("{input}")
92  default_chain = LLMChain(llm=llm, prompt=default_prompt)
```

RouterChain

To achieve this goal Step (4)

- Define a template that is used by the LLM to route between different chains. This has instructions on the task to be done, as well as the specific formatting that the output should be in.



RouterChain

To achieve this goal Step (4)

```
MULTI_PROMPT_ROUTER_TEMPLATE = """Given a raw text input to a \
language model select the model prompt best suited for the input. \
You will be given the names of the available prompts and a \
description of what the prompt is best suited for. \
You may also revise the original input if you think that revising\
it will ultimately lead to a better response from the language model.

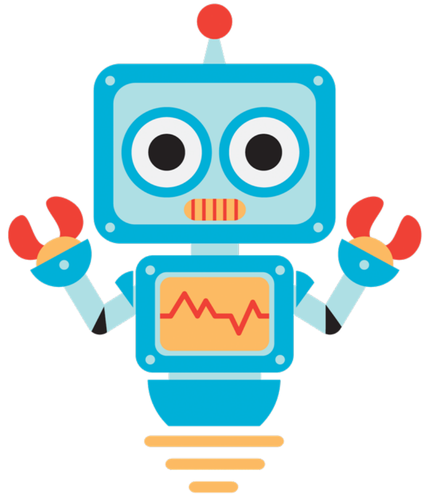
<< FORMATTING >>
Return a markdown code snippet with a JSON object formatted to look like:
```json
{{{{
    "destination": string \ name of the prompt to use or "DEFAULT"
    "next_inputs": string \ a potentially modified version of the original input
}}}}
```

REMEMBER: "destination" MUST be one of the candidate prompt \
names specified below OR it can be "DEFAULT" if the input is not\
well suited for any of the candidate prompts.
REMEMBER: "next_inputs" can just be the original input \
if you don't think any modifications are needed.

<< CANDIDATE PROMPTS >>
{destinations}

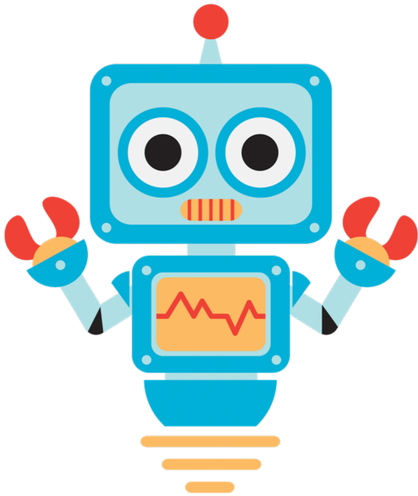
<< INPUT >>
{{input}}

<< OUTPUT (remember to include the ```json)>>"""
```



RouterChain

To achieve this goal Step (5)

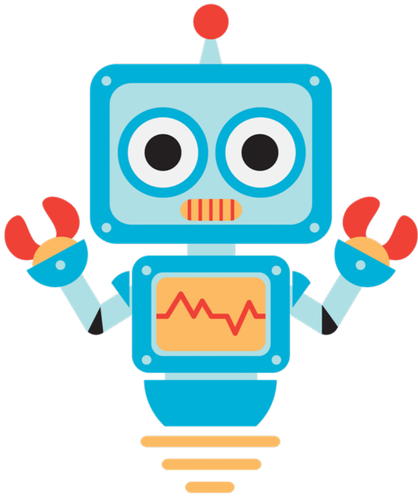


- **Create a full router template** by formatting it with the destinations that we defined above. This template is flexible to different types of destinations. We can add different types of destinations like English/ Latin apart from Physics, Math, History or Computer Science defined above.

- **Create the prompt template from this template and create the router chain by passing in the LLM and overall router prompt**. We have used a **router output parser** as well which helps the router chain decide which subchains to route between.

RouterChain

To achieve this goal Step (5)

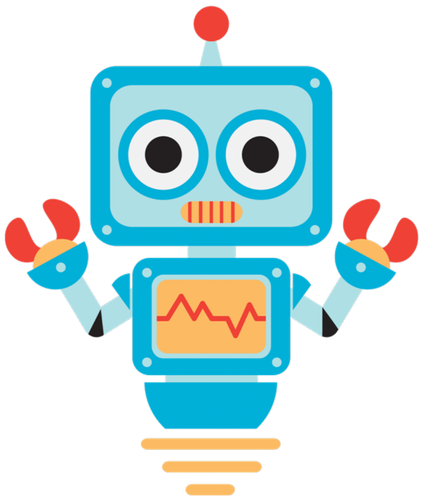


```
router_template = MULTI_PROMPT_ROUTER_TEMPLATE.format(
    destinations=destinations_str
)
router_prompt = PromptTemplate(
    template=router_template,
    input_variables=["input"],
    output_parser=RouterOutputParser(),
)
```

RouterChain

To achieve this goal Step (7)

- **Create the overall chain**, which has a router chain, destination chain and default chain.

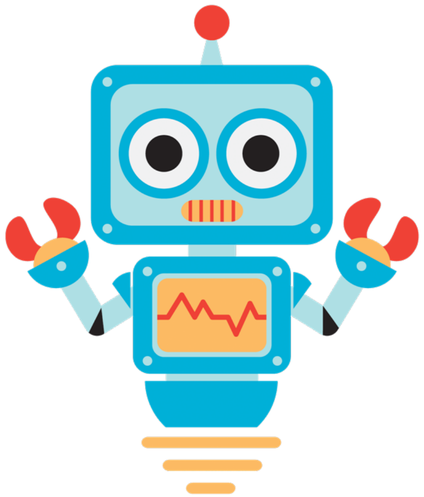


RouterChain

To achieve this goal Step (8)

- **Create the overall chain**, which has a router chain, destination chain and default chain.

```
137    router_chain = LLMRouterChain.from_llm(llm, router_prompt)
138
139    chain = MultiPromptChain(router_chain=router_chain,
140                             destination_chains=destination_chains,
141                             default_chain=default_chain, verbose=True
142                            )
```

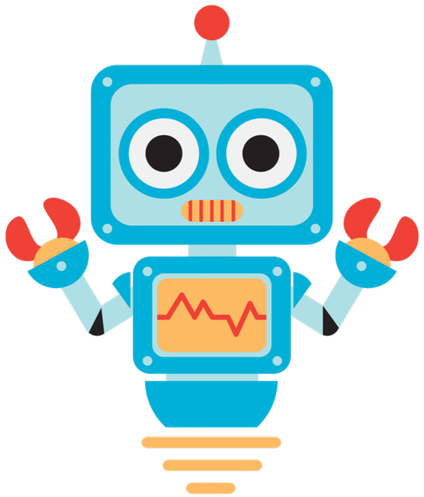


RouterChain

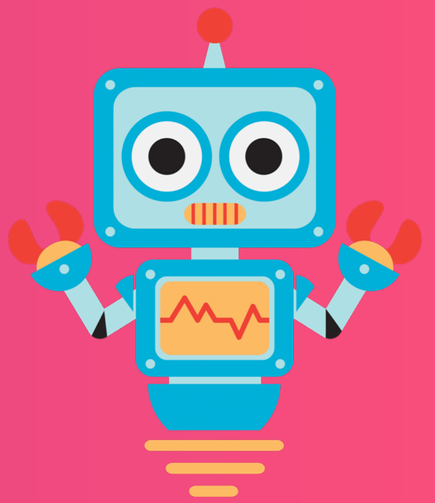
To achieve this goal Step (9)

- Run the whole chain and ask your question.

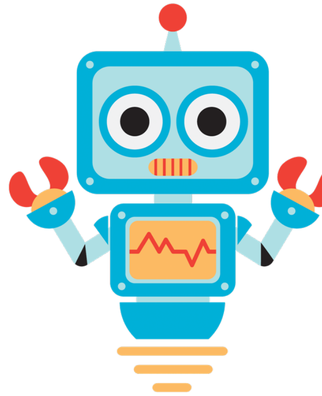
```
149
150 print(chain.run("what is  greedy algorithms?"))
151
152
```

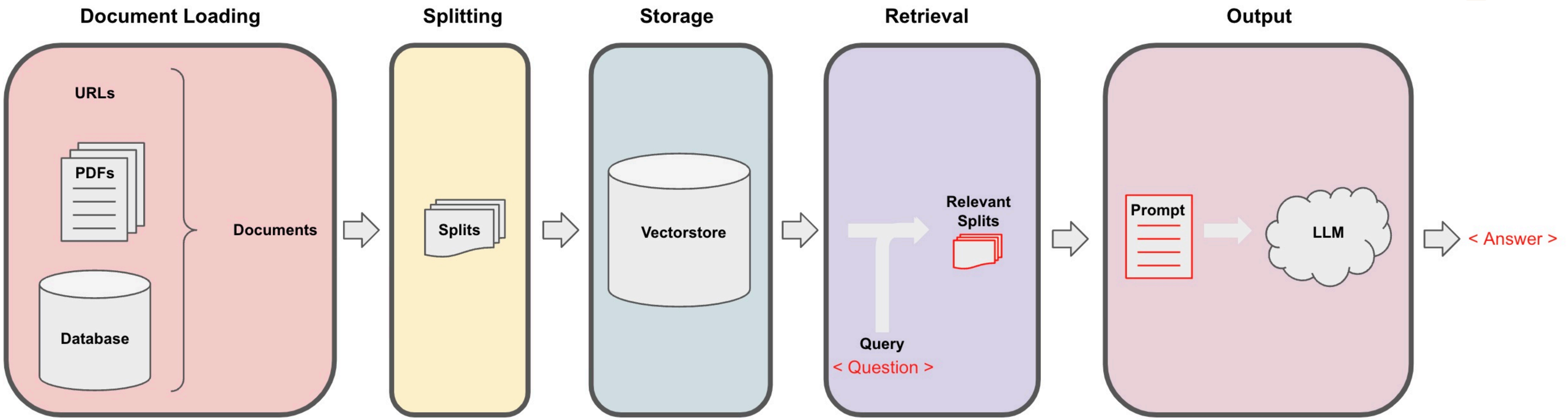


Chat with your documents



Chat with your documents

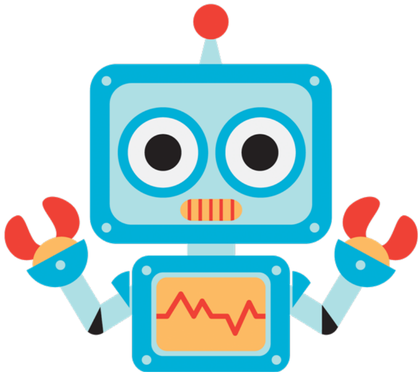




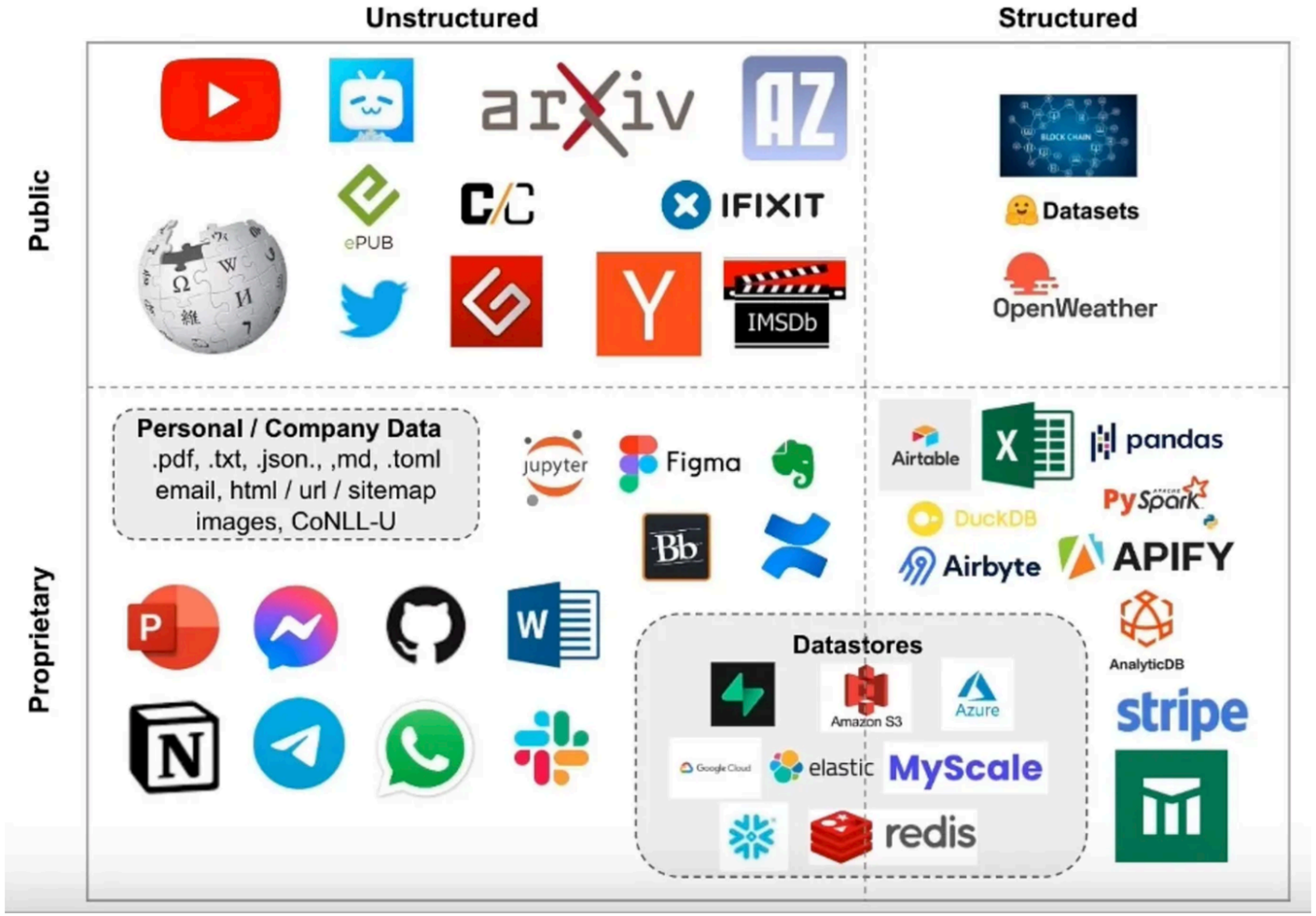
In retrieval augmented generation (RAG) framework, an LLM retrieves contextual documents from an external dataset as part of its execution.

Chat with your documents

document loaders

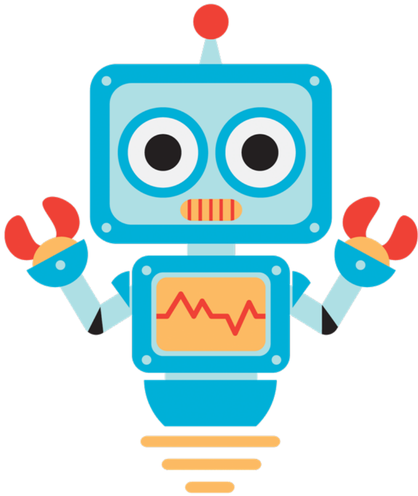


- Document loaders deal with the specifics of accessing and converting data from a variety of different formats and sources into a standardized format.
- Document loaders take in data from these data sources and load them into a standard document object, consisting of content and associated metadata.



Chat with your documents

document loaders



- PyPDF DataLoader
- Youtube DataLoader
- WebBaseLoader

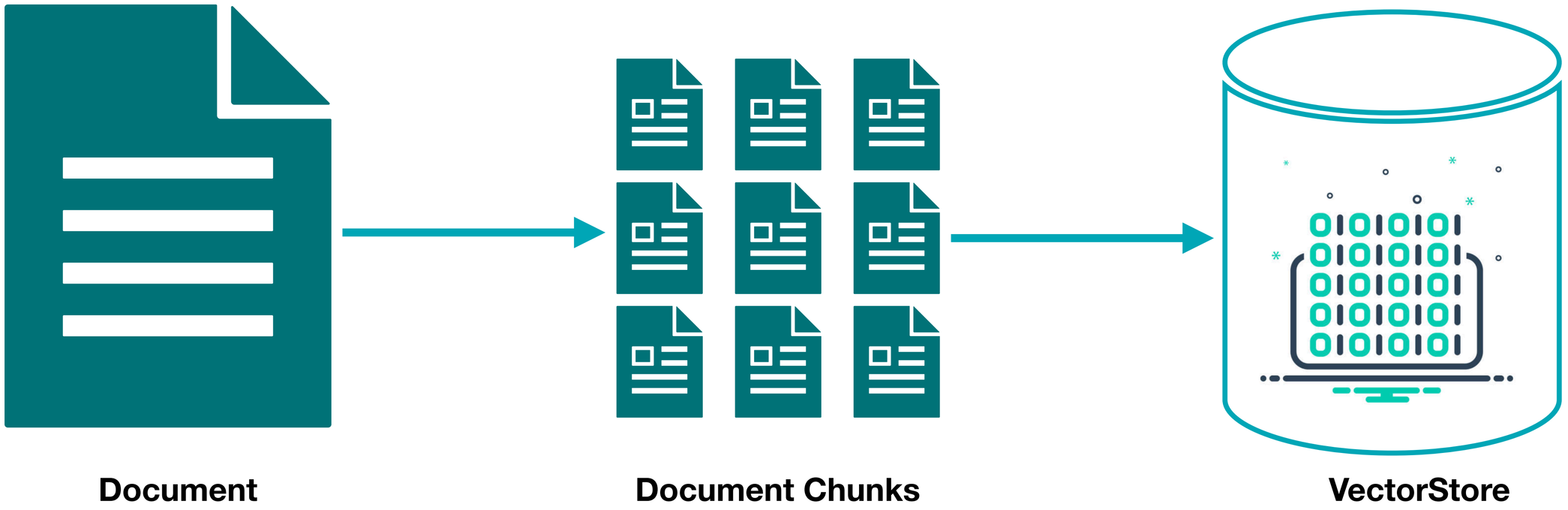
```
#! pip install yt_dlp
# ! pip install pydub
def load_youTube(url):
    save_dir = "docs/youtube/"
    loader = GenericLoader(
        YoutubeAudioLoader( urls: [url], save_dir),
        OpenAIWhisperParser()
    )
    docs = loader.load()
    return docs
```

```
import os
from langchain.document_loaders import PyPDFLoader
from langchain.document_loaders.generic import GenericLoader
from langchain.document_loaders.blob_loaders.youtube_audio import YoutubeAudioLoader
from langchain.document_loaders import WebBaseLoader

```

Chat with your documents

Documents splitters



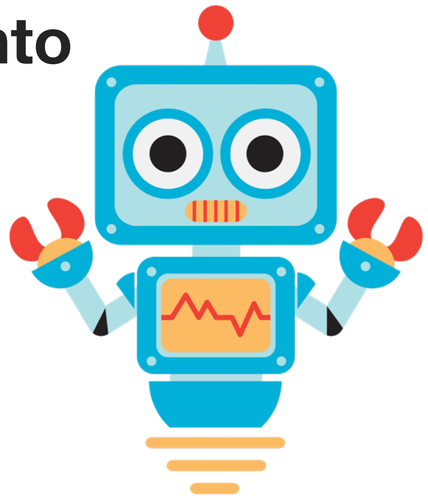
Splitting documents into smaller chunks is important and tricky as we need to maintain meaningful relationships between the chunks.

Chat with your documents

Documents splitters

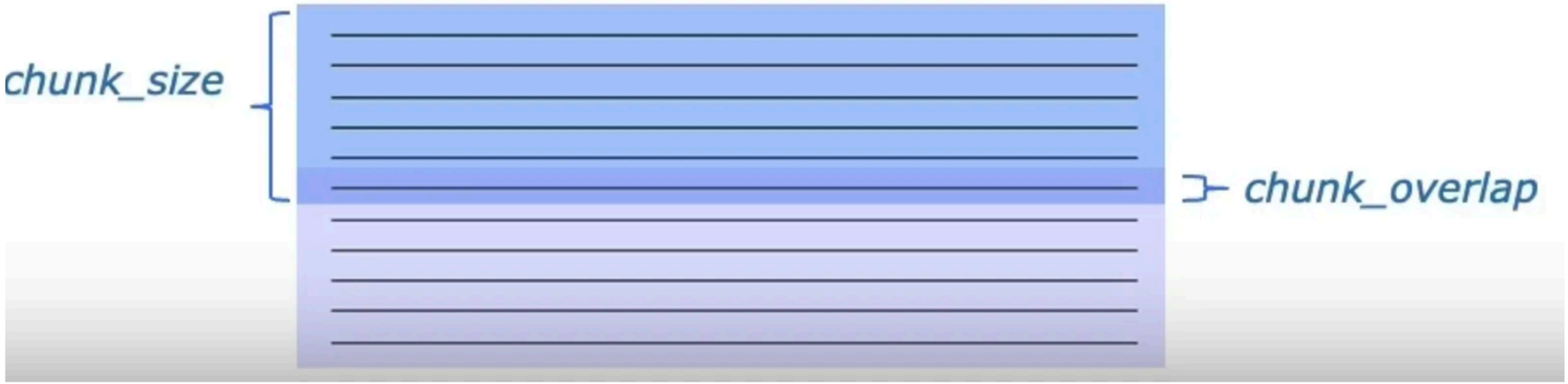
- The **CharacterTextSplitter** is used to split a long text into smaller chunks **based on a specified character**. It splits the text by trying to keep paragraphs, sentences, and words together as long as possible, as these are semantically related pieces of text.

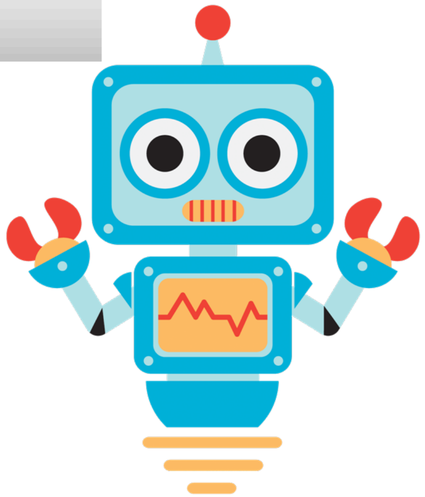
- The **RecursiveCharacterTextSplitter** splits the text by trying to keep paragraphs, sentences, and words together as long as possible, similar to the CharacterTextSplitter. **However, it also recursively splits the text into smaller chunks if the chunk size exceeds a specified threshold.**



Chat with your documents

Documents splitters

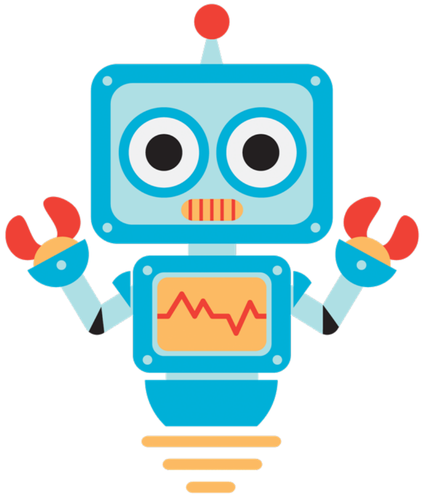




Chat with your documents

Documents splitters

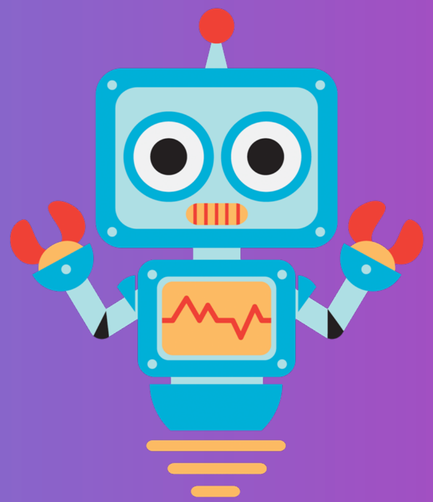
```python
from langchain.text_splitter import RecursiveCharacterTextSplitter,CharacterTextSplitter
# case 1
chunk_size=24
chunk_overlap=4

r_splitter= RecursiveCharacterTextSplitter(
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap
)

c_splitter=CharacterTextSplitter(
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap
)
```



Thank you for your attention

Day 5

January 23th 2024



GAME Khoot