LangChain From 0 To 1

Unveiling the Power of LLM Programming

GitHub

https://github.com/Stell0/fosdem2024

- Presentation
- Code
- Useful links

Our Journey

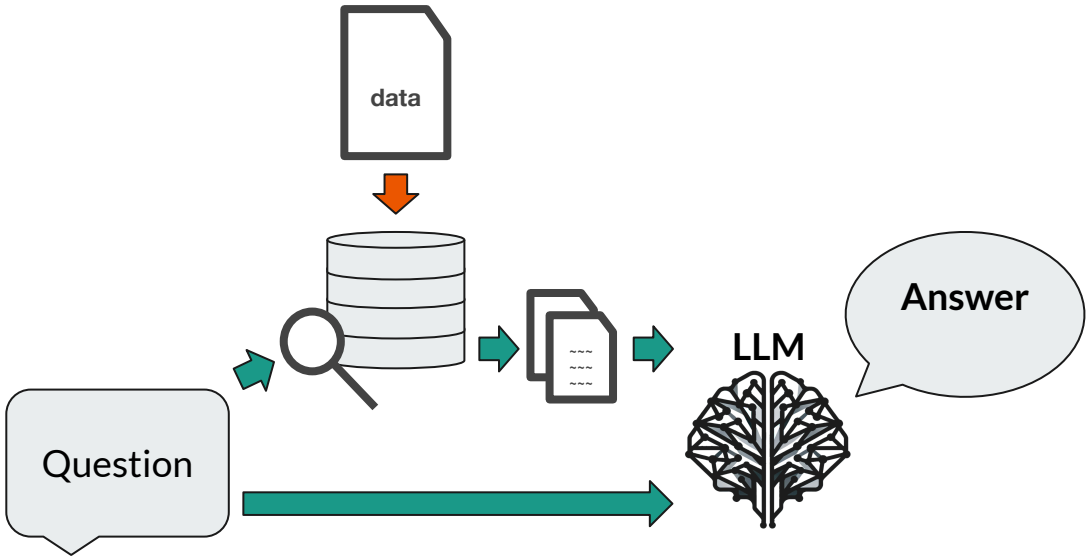
1. Introduction to LangChain
2. Document loaders
3. Text Splitters
4. Embeddings
5. Vectorstores
6. Retrievers
7. Prompts and Templates
8. Large Language Models
9. Chains
10. **RAG - Retrieval Augmented Generation**
11. Demo

Retrieval Augmented Generation (RAG) 🔥🔥🔥

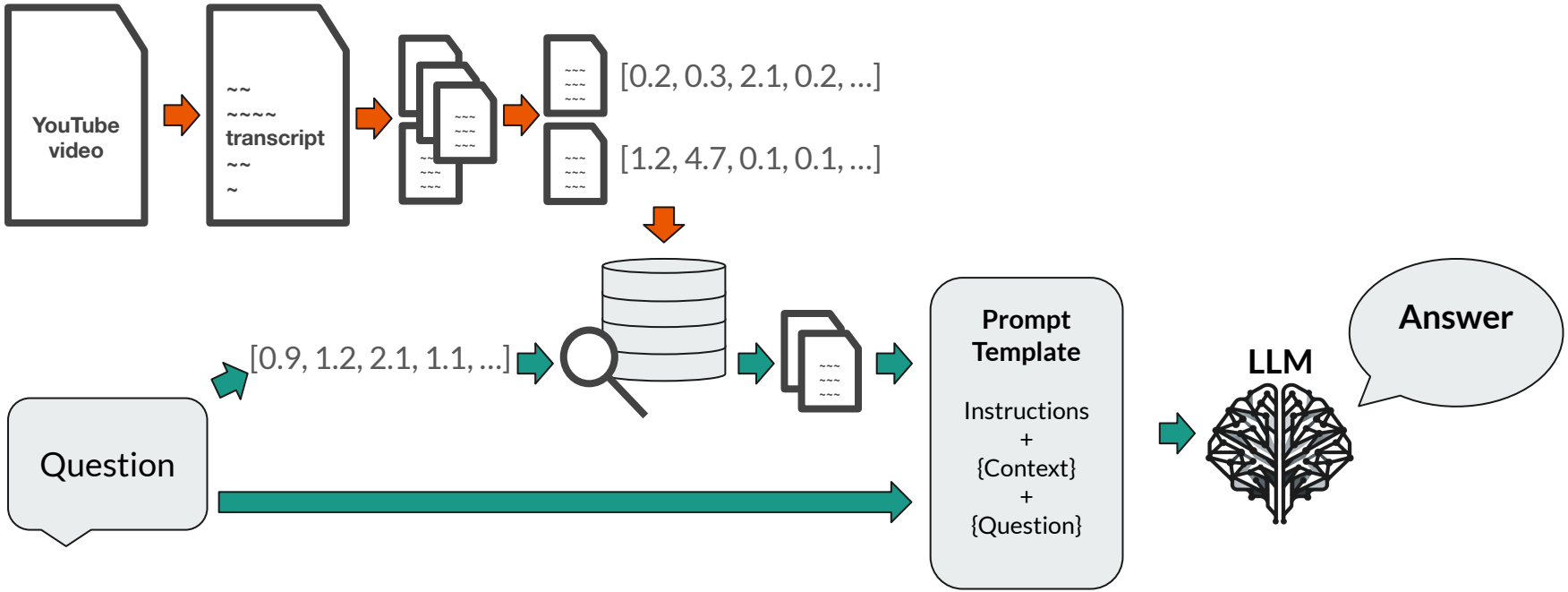
Augment LLM knowledge using additional data

- Combines retrieval + generation
- Data not in training dataset
  - Private data
  - Data after cutoff date, even real time
- Improves accuracy and relevancy
- Supports evidence-Based Responses, can reference source

Example of RAG use case: QA over unstructured data

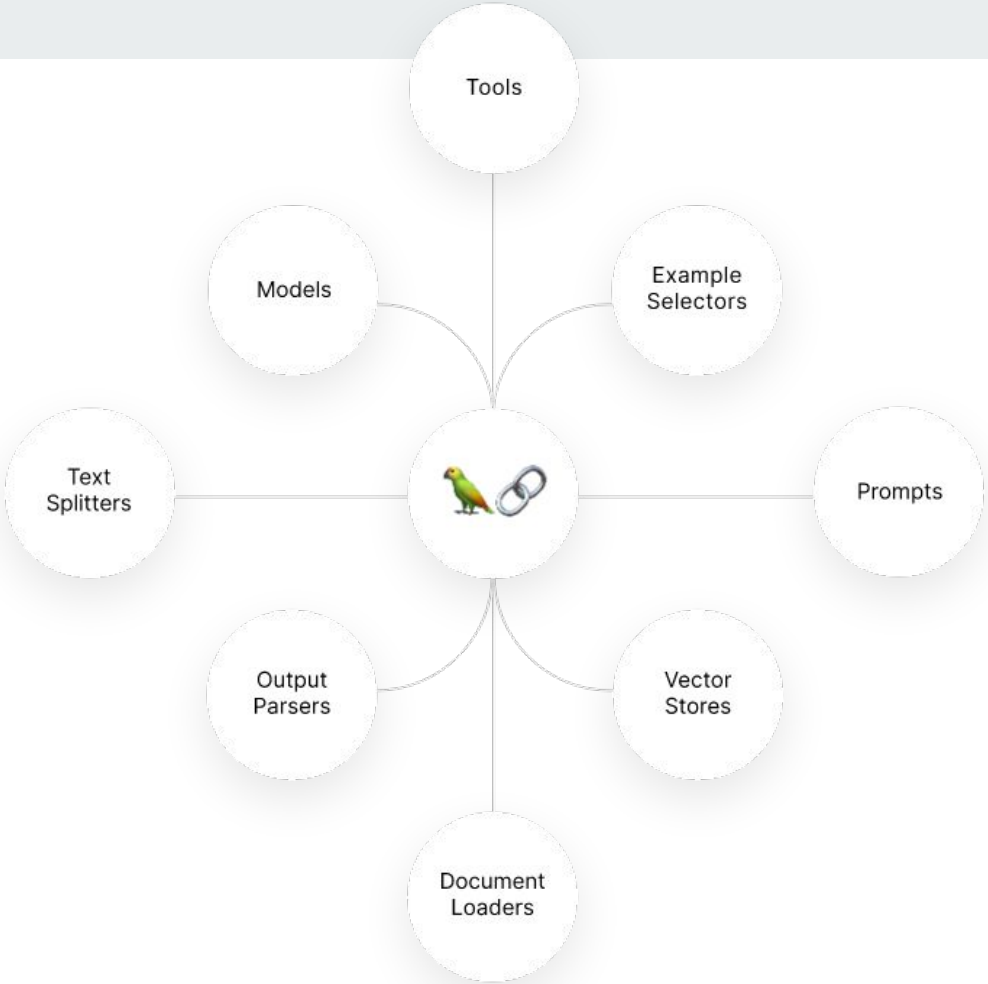


Example of RAG use case: QA over unstructured data



LangChain

- Python (also JS/TS) framework
- Building blocks
- Swappable components
- Examples
- From PoC to Production
- Speed of improvement

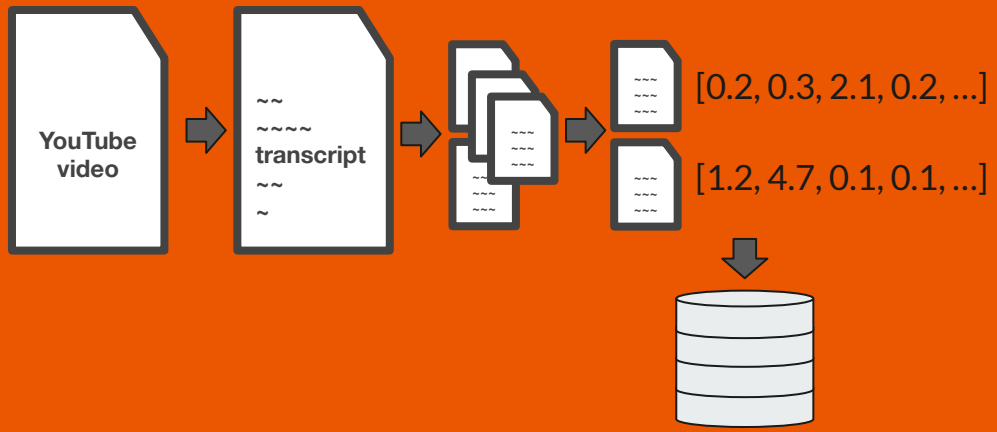


LangChain

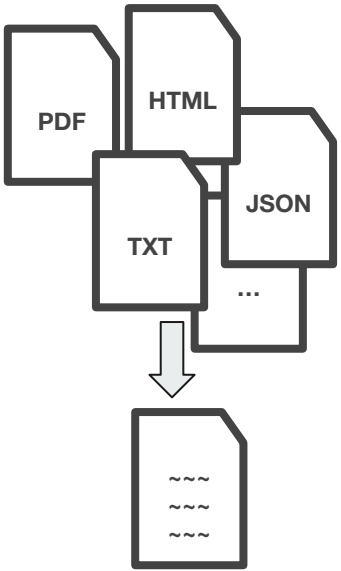
```
$ cat requirements.txt
langchain
openai
chromadb
…
```

```
export OPENAI_API_KEY=xx-xXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxXxX
```

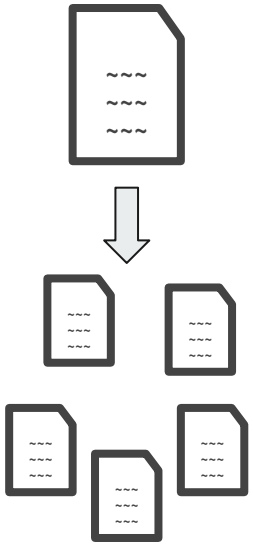
Preparing and storing data



Document loader

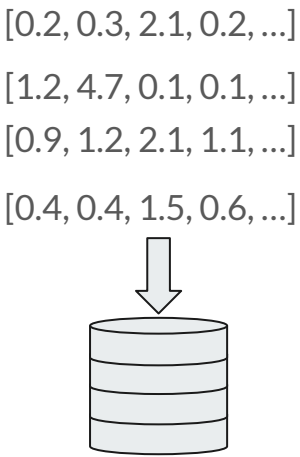


Text Splitter

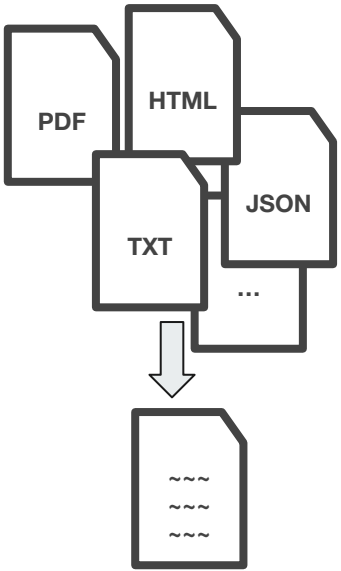


Embedding Function

Vectorstore



Document Loaders



Arxiv
CSV
Discord
Email
EPub
EverNote
Facebook Chat
Figma
Git
GitHub
HTML
JSON
Markdown
Mastodon
MediaWiki Dump
Microsoft Word
MongoDB
Open Document Format (ODT)
Pandas DataFrame
PubMed
ReadTheDocs Documentation
Reddit
RSS Feeds
Slack
Snowflake
Telegram
X
URL
WhatsApp Chat
Wikipedia
XML
YouTube audio
YouTube transcripts

Document Loaders

Loading a YouTube video transcript

- YoutubeLoader from LangChain **Community**
- loaders return a **list of Documents**

```
from langchain_community.document_loaders import YoutubeLoader
loader = YoutubeLoader.from_youtube_url("https://www.youtube.com/watch?v=8fEEbKJoNbU")
documents = loader.load()
```

Document class

page_content: Document text

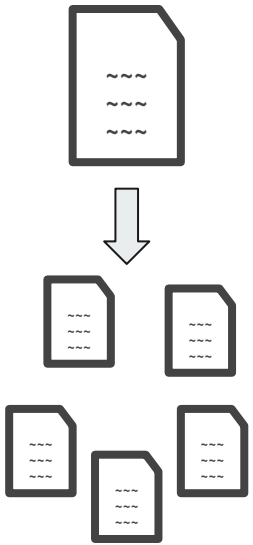
metadata: dictionary { “source”:”https://…”}

```
class Document(Serializable):
    """Class for storing a piece of text and associated metadata."""

    page_content: str
    """String text."""
    metadata: dict = Field(default_factory=dict)
    """Arbitrary metadata about the page content (e.g., source, relationships to other
        documents, etc.).
    """
```

Text Splitters

Break text into smaller chunks



```
## What is FOSDEM?

FOSDEM is a free and non-commercial event organised by the community for the community. The goal is to provide free and open source software developers and communities a place to meet to:

-   get in touch with other developers and projects;
-   be informed about the latest developments in the free software world;
-   be informed about the latest developments in the open source world;
-   attend interesting talks and presentations on various topics by project leaders and committers;
-   to promote the development and benefits of free software and open source solutions.

Participation and attendance is _totally free_, though the organisers gratefully accept donations and sponsorship.## Developer rooms

The FOSDEM team feels it is very important for free and open source software developers around the world to be able to meet in “real life”.

To this end, we have set up developer rooms (devrooms) with network/internet connectivity and projectors where teams can meet and showcase their projects. Devrooms are a place for teams to discuss, hack and publicly present latest directions, lightning talks, news and discussions. We believe developers can benefit a lot from these meetings.## A bit of history

In 2000, Raphael Bauduin, a fan of the Linux movement in Belgium, decided to organise a small meeting for developers of Open Source software. He called it 'Open Source Developers’ European Meeting' (OSDEM).

Raphael created a mailing list, a small website and spread the word to people around him. Only a few weeks later, lots of people were waiting for an exciting event in Brussels! Invitations were sent to well-known figures in the community: Rasterman, Fyodor, Jeremy Allison and so on. They all gave a very positive response, and OSDEM was on the road to success.

For the second year, OSDEM was renamed FOSDEM. And now, many years later, it has grown into the event it is today. We now try to cover a wide spectrum of free and open source software projects, and offer a platform for people to collaborate. Every year, we host more than 5000 developers at the ULB Solbosch campus.Raphael is no longer the driving force behind FOSDEM. After 7 years of hard work he left the team for new Open Source plans. The FOSDEM flag is now proudly carried by the following people:
```

https://chunkviz.up.railway.app

Text Splitters: 5 levels of text splitting

Characters / Tokens

Recursive Character

Document structure

Semantic Chunker

Agent-like Splitting

Text Splitters

RecursiveCharacterTextSplitter

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=2000,
    chunk_overlap=0,
)
```

Embeddings

- Numerical representation
- **Vectors** in **High-dimensional space**
- Each dimension reflects an aspect
- Similarity = Proximity in embedding space

Embeddings

- Complexity is hidden
- We rely on an **external provider**
- **note:** data is sent to the external provider

```
db = Chroma.from_documents(chunks, OpenAIEmbeddings())
```

Vectorstore

Storing embeddings

- Stores
- Search
- Retrieve

[0.2, 0.3, 2.1, 0.2, ...]

[1.2, 4.7, 0.1, 0.1, ...]

[0.9, 1.2, 2.1, 1.1, ...]

[0.4, 0.4, 1.5, 0.6, ...]

Vectorstore

- ChromaDB initialized from our documents
- OpenAI embedding function
- Optional: persist directory

```
db = Chroma.from_documents(chunks, OpenAIEmbeddings())
```

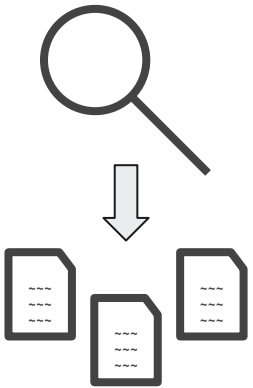
Most Used Vectorstores

https://blog.langchain.dev/langchain-state-of-ai-2023/

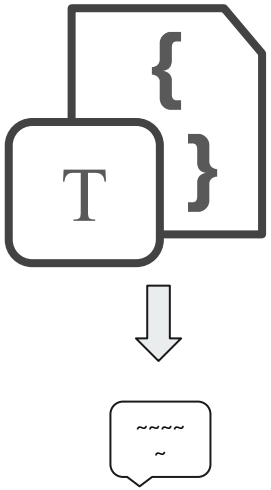
Using data


[0.9, 1.2, 2.1, 1.1, ...]
Question
Prompt Template
Instructions
+
{Context}
+
{Question}
LLM
Answer

Retriever



Prompt/Template



LLM

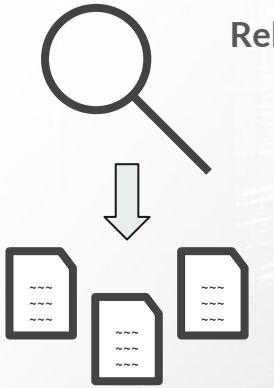
Chain

Retriever

Question → Embedding → distance

⬇️

Relevant Documents



Retriever

```
retriever = db.as_retriever()
```

Another Retriever

Multi Query Retriever

- use **LLM** to generate multiple **variations** of our questions
- increase chances of finding Documents near to the questions

```
from langchain.retrievers.multi_query import MultiQueryRetriever
retriever = MultiQueryRetriever.from_llm(
    retriever=db.as_retriever(), llm=llm
)
```

Prompt/Template

- Guide LLM output

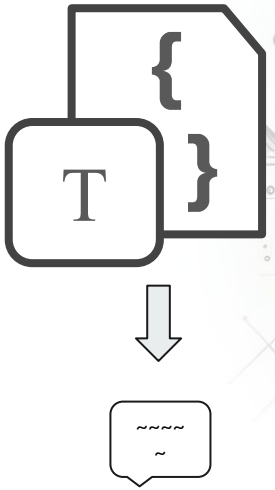
Question

+

Documents

⬇️

context



Prompt

```
from langchain.prompts import ChatPromptTemplate, PromptTemplate,
HumanMessagePromptTemplate
prompt = ChatPromptTemplate(
    input_variables=['context', 'question'],
    messages=[
        HumanMessagePromptTemplate(
            prompt=PromptTemplate(
                input_variables=['context', 'question'],
                template="You are an assistant for question-answering tasks. Use the
following pieces of retrieved context to answer the question. If you don't know the
answer, just say that you don't know. Use three sentences maximum and keep the answer
concise.\nQuestion: {question} \nContext: {context} \nAnswer:"
                )
            )
    ]
)
```

Prompt from Hub

```
from langchain import hub
prompt = hub.pull("rlm/rag-prompt")
```

LLM

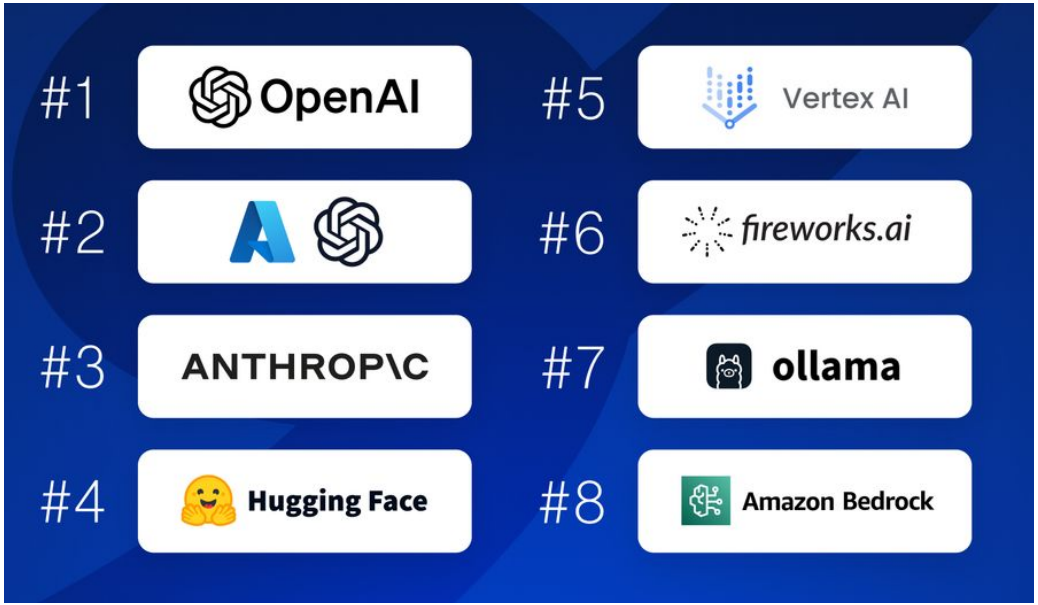
https://python.langchain.com/docs/integrations/llms/

LLM

```
from langchain.chat_models import ChatOpenAI
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
llm = ChatOpenAI(streaming=True, callbacks=[StreamingStdOutCallbackHandler()],
 temperature=0)
```

“Nobody Gets Fired For Buying ~~IBM~~ OpenAI”

Most Used LLM Providers



https://blog.langchain.dev/langchain-state-of-ai-2023/

Most Used OSS Model Providers

https://blog.langchain.dev/langchain-state-of-ai-2023/

Put everything together

```
# search for similar documents
docs = retriever.get_relevant_documents(question)
# create context merging docs together
context = "\n\n".join(doc.page_content for doc in docs)
# get valorized prompt from template
prompt_val = prompt.invoke({"context": context, "question": question})
# get response from llm
result = llm(prompt_val.to_messages())
```

Chains

Sequence of calls

- Advantages:
  - Simple
  - Modular
  - Efficient
- compose your own
- Off-the-shelf
- Legacy Class
- LCEL
  - Streaming
  - Async (and sync) support
  - Optimized parallel execution
  - integrated with LangSmith and LangServe
  - ...

Put everything together using LCEL

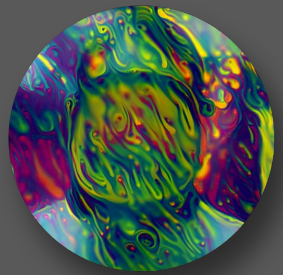
```
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser
rag_chain = (
    {
        "context": retriever | (lambda docs: "\n\n".join(doc.page_content for doc in
 docs)),
        "question": RunnablePassthrough()
    }
    | prompt
    | llm
    | StrOutputParser()
)

result = rag_chain.invoke(question)
```

Other use cases

- QA over structured data
  - Question → SQL Query → Query Results → Additional Context →Answer
- Extraction
  - Unstructured Text + JSON Schema → Compiled JSON
- Summarization
  - **MOAR text** → LESS text
- Synthetic data generation
  - JSON Schema → [Unstructured Text, Unstructured Text, Unstructured Text, Unstructured Text ...]
- Agents
  - let LLM takes actions

The End



https://github.com/Stell0

https://x.com/Stll00

https://t.me/Stll0

https://www.linkedin.com/in/stefano-fancello