

# LangChain

# LangChain

[illegible]

Python ? JavaScript ??????????????????REST API? LangServe?????????????

LangSmith?LangChain ???

- [Introduction | ??? LangChain](#)
- [LangChain????AI????????LLM???? - ALPHA Camp](#)
- GitHub: <https://github.com/langchain-ai/langchain>
- Hub: [LangSmith \(langchain.com\)](https://langchain.com)
- ???[sugarforever/wtf-langchain](#)
- [CookBook](#)
- [LangChain Templates](#)

# LangSmith

## LangChain ????????????????????????????????? RAG ?????????

- <https://github.com/langchain-ai/langsmith-cookbook>
- [LangChain ????? LangSmith ????? - MyApollo](#)
- [??LangSmith?????????\(LLM\)???????????????? - ?? - ????? - ????](#)

# RAG

- [Learn RAG with Langchain \(ipynb\)](#)
- [LangChain: A Complete Guide & Tutorial \(nanonets.com\)](#)
- [Meta-Llama CookBook for RAG \(ipynb\)](#)
- [LangChain and Streamlit RAG | Medium](#)
  - [GitHub: https://github.com/streamlit/example-app-langchain-rag](https://github.com/streamlit/example-app-langchain-rag)

## Retrievers in LCEL

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
```

```

from langchain_core.runnables import RunnablePassthrough

template = """Answer the question based only on the following context:

{context}

Question: {question}
"""

prompt = ChatPromptTemplate.from_template(template)
model = ChatOpenAI()

def format_docs(docs):
    return "\n\n".join([d.page_content for d in docs])

chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | model
    | StrOutputParser()
)

chain.invoke("What did the president say about technology?")

```

## ChatPromptTemplate

```

few_shot_examples = [
    {"input": "Could you please clarify the terms outlined in section 3.2 of the contract?",
     "output": "Certainly, I will provide clarification on the terms in section 3.2."},
    {"input": "We are interested in extending the payment deadline to 30 days instead of the current 15 days. Additionally, we would like to add a clause regarding late payment penalties.",
     "output": "Our request is to extend the payment deadline to 30 days and include a clause on late payment penalties."},
    {"input": "The current indemnification clause seems too broad. We would like to narrow it down to cover only direct damages and exclude consequential damages. Additionally, we propose including a dispute resolution clause specifying arbitration as the preferred method of resolving disputes."},
    {"input": "We suggest revising the indemnification clause to limit it to covering direct damages and excluding

```

consequential damages.

Furthermore, we recommend adding a dispute resolution clause that specifies arbitration as the preferred method of resolving disputes.""}},

{"input": "I believe the proposed changes are acceptable.",

"output": "Thank you for your feedback. I will proceed with implementing the proposed changes."}

]

few\_shot\_template = ChatPromptTemplate.from\_messages(

[

    ("human", "{input}"),

    ("ai", "{output}")

]

)

few\_shot\_prompt = FewShotChatMessagePromptTemplate(

    example\_prompt=few\_shot\_template,

    examples=few\_shot\_examples,

)

print(few\_shot\_prompt.format())

# Loader

## Web

```
from langchain_community.document_loaders import WebBaseLoader
loader = WebBaseLoader(
    web_paths=("https://lilianweng.github.io/posts/2023-06-23-agent/"),
    bs_kwargs=dict(
        parse_only=bs4.SoupStrainer(
            class_=("post-content", "post-title", "post-header")
        )
    ),
)
docs = loader.load()
```

## Text

```
from langchain_community.document_loaders import DirectoryLoader
loader = DirectoryLoader("../", glob="**/*.md")
```

```
docs = loader.load()
len(docs)
print(docs[0].page_content[:100])
```

```
from langchain.document_loaders import TextLoader

dataset_folder_path='/path/to/dataset/'
documents=[]
for file in os.listdir(dataset_folder_path):
    loader=TextLoader(dataset_folder_path+file)
    documents.extend(loader.load())

print(documents[:3])
```

## Markdown

```
"""
%pip install "unstructured[md]"
"""

from langchain_community.document_loaders import UnstructuredMarkdownLoader
markdown_path = "../././README.md"
loader = UnstructuredMarkdownLoader(markdown_path)

data = loader.load()
assert len(data) == 1
readme_content = data[0].page_content
print(readme_content[:3])
```

## PDF + Text

```
from langchain_community.document_loaders import TextLoader
from langchain_community.document_loaders import PyPDFLoader

documents = []
for filename in SAMPLEDATA:
    path = os.path.join(os.getcwd(), filename)

    if filename.endswith(".pdf"):
        loader = PyPDFLoader(path)
        new_docs = loader.load_and_split()
```

```

        print(f"Processed pdf file: {filename}")
    elif filename.endswith(".txt"):
        loader = TextLoader(path)
        new_docs = loader.load_and_split()
        print(f"Processed txt file: {filename}")
    else:
        print(f"Unsupported file type: {filename}")

    if len(new_docs) > 0:
        documents.extend(new_docs)

SAMPLEDATA = []

print(f"\nProcessing done.")

```

????

?????

```

# Helper function for printing docs
def pretty_print_docs(docs):
    print(
        f"\n{'-' * 100}\n".join(
            [f"Document {i+1}:\n\n" + d.page_content for i, d in enumerate(docs)]
        )
    )

```

Revision #16

Created 24 June 2024 10:55:58 by Admin

Updated 23 August 2024 10:27:01 by Admin