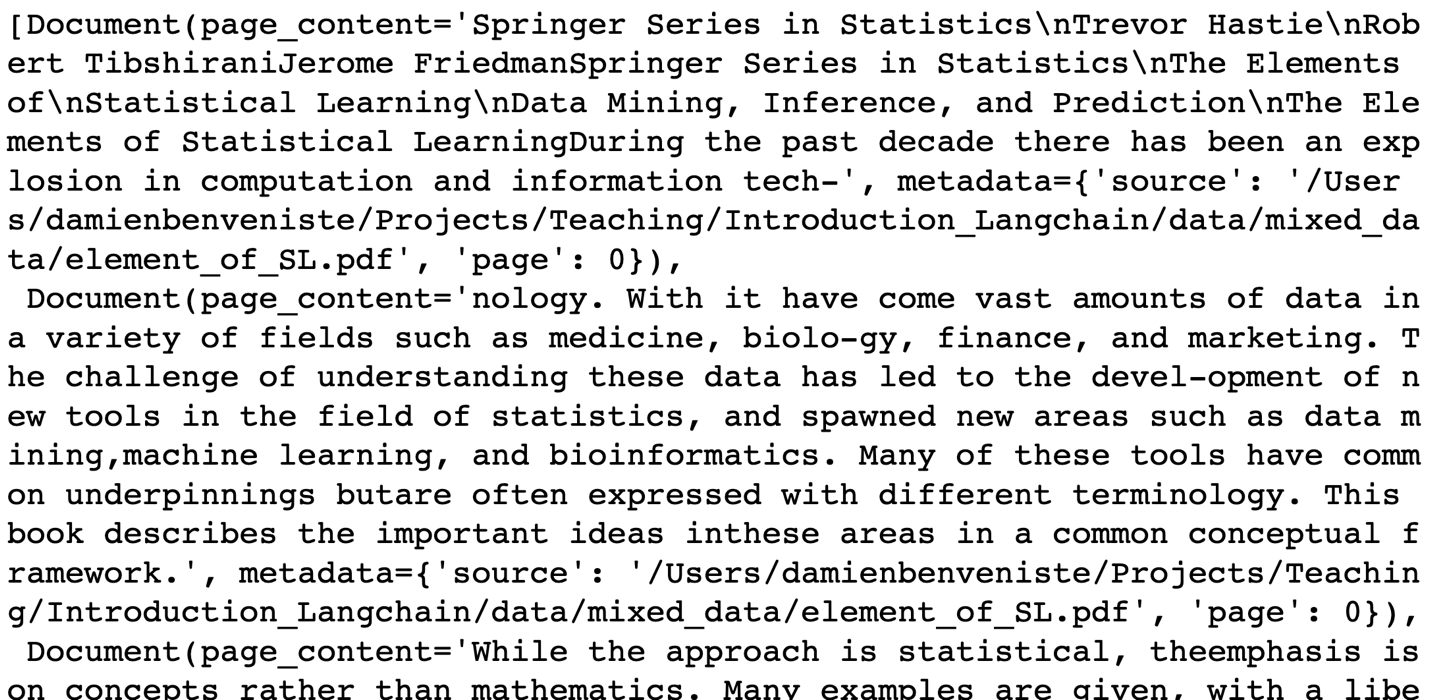
**Indexing data**

Let’s load and split the pdf file of [Element of Statistical Learning](https://hastie.su.domains/ElemStatLearn/):

1. from langchain.document\_loaders import PyPDFLoader
2. from langchain.text\_splitter import RecursiveCharacterTextSplitter
4. file\_path = '...'
6. loader = PyPDFLoader(file\_path=file\_path)
8. text\_splitter = RecursiveCharacterTextSplitter(
9. chunk\_size=500,
10. chunk\_overlap=0
11. )
13. data = loader.load\_and\_split(text\_splitter=text\_splitter)
14. data



Let’s install the [FAISS](https://github.com/facebookresearch/faiss) package:

1. %pip install faiss-cpu

And let’s embed text with the OpenAI embeddings

1. from langchain.embeddings.openai import OpenAIEmbeddings
3. embeddings = OpenAIEmbeddings(show\_progress\_bar=True)
4. vector1 = embeddings.embed\_query('How are you?')
5. len(vector1)
7. > 1536

Let’s embed the book data

1. from langchain.vectorstores import FAISS
3. index = FAISS.from\_documents(data, embeddings)

We can search in that index:

1. index.similarity\_search\_with\_relevance\_scores(
2. "What is machine learning?"
3. )

A screenshot of a computer

Description automatically generated

We can use that index in a chain

1. from langchain.chains import RetrievalQA
2. from langchain.chat\_models import ChatOpenAI
3. from langchain.callbacks import StdOutCallbackHandler
5. retriever = index.as\_retriever()
6. retriever.search\_kwargs['fetch\_k'] = 20
7. retriever.search\_kwargs['maximal\_marginal\_relevance'] = True
8. retriever.search\_kwargs['k'] = 10
10. llm = ChatOpenAI()
12. chain = RetrievalQA.from\_chain\_type(
13. llm=llm,
14. retriever=retriever,
15. verbose=True
16. )
18. handler = StdOutCallbackHandler()
20. chain.run(
21. 'What is machine learning?',
22. callbacks=[handler]
23. )

*Machine learning is a field of study that involves the development of algorithms and models that can learn from data and make predictions or decisions without being explicitly programmed. It focuses on creating computer systems that can automatically learn and improve from experience, rather than being explicitly programmed for specific tasks. Machine learning algorithms analyze large amounts of data to identify patterns, make predictions, or learn from examples and feedback. It is widely used in various fields such as science, finance, and industry for tasks like predicting stock prices, medical diagnoses, and customer behavior analysis.*

*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*----------------

Loading data into a Vector Database

We are going to load the data in [Pinecone](https://app.pinecone.io/). Let’s install the Python package

1. %pip install pinecone-client

And let’s load the data into the database

1. import pinecone
2. from langchain.vectorstores import Pinecone
4. pinecone.init(
5. api\_key=PINECONE\_API\_KEY, # find at app.pinecone.io
6. environment=PINECONE\_ENV # next to api key in console
7. )
9. index\_name = "langchain-demo"
10. db = Pinecone.from\_documents(
11. data,
12. embeddings,
13. index\_name=index\_name
14. )

And we can now augment an LLM with the database

1. chain = RetrievalQA.from\_chain\_type(
2. llm=llm,
3. retriever=db.as\_retriever(),
4. verbose=True
5. )
7. chain.run(
8. 'What is machine learning?',
9. callbacks=[handler]
10. )

*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*----------------

Providing sources

I will show to provide the sources as we answer questions. Let’s install the [NewsAPI](https://newsapi.org/" \t "_blank) Python package:

1. %pip install newsapi-python

Let’s get the news about “Artificial Intelligence“ from the past week:

1. from datetime import date, timedelta
2. from newsapi import NewsApiClient
4. newsapi = NewsApiClient(api\_key=NEWS\_API\_KEY)
6. today = date.today()
7. last\_week = today - timedelta(days=7)
9. latest\_news = newsapi.get\_everything(
10. q='artificial intelligence',
11. from\_param=last\_week.strftime("%Y-%m-%d"),
12. to=today.strftime("%Y-%m-%d"),
13. sort\_by='relevancy',
14. language='en'
15. )

and let’s create documents:

1. from langchain.docstore.document import Document
2. docs = [
3. Document(
4. page\_content=article['title'] + '\n\n' + article['description'],
5. metadata={
6. 'source': article['url'],
7. }
8. ) for article in latest\_news['articles']
9. ]

Let’s create a chain that provides the sources with the answers

1. from langchain.chains import create\_qa\_with\_sources\_chain
2. from langchain.chains.combine\_documents.stuff import StuffDocumentsChain
3. from langchain.prompts import PromptTemplate
5. qa\_chain = create\_qa\_with\_sources\_chain(llm)
7. doc\_prompt = PromptTemplate(
8. template="Content: {page\_content}\nSource: {source}",
9. input\_variables=["page\_content", "source"],
10. )
12. final\_qa\_chain = StuffDocumentsChain(
13. llm\_chain=qa\_chain,
14. document\_variable\_name="context",
15. document\_prompt=doc\_prompt,
16. )
18. index = FAISS.from\_documents(docs, embedding=embeddings)

21. chain = RetrievalQA(
22. retriever=index.as\_retriever(),
23. combine\_documents\_chain=final\_qa\_chain
24. )

Let’s ask a question:

1. question = """
2. What is the most important news about artificial intelligence from last week?
3. """
5. answer = chain.run(question)
6. answer
7. {
8. "answer": "The most important news about artificial intelligence from last week is the use of AI to train on the works of authors Stephen King and Margaret Atwood. These authors responded to the revelation that their work is being used to train AI. Additionally, AI took the stage at the Edinburgh Fringe festival, raising the question of whether AI can deliver a satisfying punchline. Furthermore, a tech expert from the University of Oxford highlighted the potential workplace threats of AI, including the possibility of AI becoming a monitoring boss. Finally, AI is being seen as a tool that can help companies connect with customers in a more personalized and efficient way.",
9. "sources": [
10. "https://www.theatlantic.com/newsletters/archive/2023/09/books-briefing-ai-stephen-king-margaret-atwood/675213/?utm\_source=feed",
11. "https://www.cnet.com/tech/ai-took-the-stage-at-the-worlds-largest-arts-festival-heres-what-happened/",
12. "https://www.foxnews.com/tech/tech-expert-existential-fears-ai-are-overblown-sees-very-disturbing-workplace-threats",
13. "https://www.techradar.com/pro/ai-could-help-companies-connect-with-customers-like-never-before"
14. ]
15. }

*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*----------

Indexing a website

We are going to use [Apify](https://apify.com/" \t "_blank) to crawl a website. Let’s download the Python package

1. %pip install apify-client chromadb

and let’s create a loader that will crawl the AiEdge Newsletter website:

1. from langchain.utilities import ApifyWrapper
2. from langchain.document\_loaders.base import Document
4. apify = ApifyWrapper()
6. loader = apify.call\_actor(
7. actor\_id="apify/website-content-crawler",
8. run\_input={
9. "startUrls": [{"url": "https://newsletter.theaiedge.io/"}],
10. "aggressivePrune": True,
11. },
12. dataset\_mapping\_function=lambda item: Document(
13. page\_content=item["text"] or "", metadata={"source": item["url"]}
14. ),
15. )

Let’s index the website data:

1. from langchain.indexes import VectorstoreIndexCreator
3. text\_splitter = RecursiveCharacterTextSplitter(
4. chunk\_size=500,
5. chunk\_overlap=0
6. )
8. index = VectorstoreIndexCreator(
9. text\_splitter=text\_splitter
10. ).from\_loaders([loader])

Let’s make a search on that index:

1. query = "What is the main subject of the aiedge newsletter?"
3. index.query\_with\_sources(query)
4. {
5. 'question': 'What is the main subject of the aiedge newsletter?',
6. 'answer': ' The main subject of the AiEdge newsletter is Machine Learning applications, Machine Learning System Design, MLOps, and the latest techniques and news about the field.\n',
7. 'sources': ''
8. }

Let’s now pass that index to a chain

1. retriever = index.vectorstore.as\_retriever()
3. qa = RetrievalQA.from\_chain\_type(
4. llm=llm,
5. retriever=retriever,
6. )
8. query = "What is the most recent article of the aiedge newsletter?"
10. qa.run(
11. query,
12. callbacks=[handler]
13. )

*"I'm sorry, but I don't have access to the specific articles or the most recent content of the AiEdge Newsletter. As an AI language model, I don't have real-time access to current articles or newsletters. It would be best to subscribe to the newsletter and check the latest edition for the most recent article."*

*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*----------

Indexing a GitHub repo

Let’s install the Python package:

1. %pip install GitPython

Let’s load a repo

1. from langchain.document\_loaders import GitLoader
3. loader = GitLoader(
4. clone\_url="https://github.com/langchain-ai/langchain",
5. repo\_path="./data/repo/",
6. file\_filter=lambda file\_path: file\_path.endswith(".py"),
7. branch="master",
8. )
10. documents = loader.load()

Let’s resplit the documents for the Python language:

1. from langchain.text\_splitter import Language
3. python\_splitter = RecursiveCharacterTextSplitter.from\_language(
4. language=Language.PYTHON,
5. chunk\_size=1000,
6. chunk\_overlap=200
7. )
9. documents = python\_splitter.split\_documents(documents)

Let’s index the data and create a chain

1. index = FAISS.from\_documents(documents, embeddings)
2. retriever = index.as\_retriever()
4. qa = RetrievalQA.from\_chain\_type(
5. llm=llm,
6. retriever=retriever,
7. )
9. query = "What is a stuff chain?"
11. qa.run(query, callbacks=[handler])

*'A stuff chain is a sequence of operations performed on a language model (LLM) to generate or process text. It typically consists of a language model chain (LLMChain) and a document chain (StuffDocumentsChain). The LLMChain is responsible for generating text based on a prompt, while the StuffDocumentsChain is used to process and manipulate documents or summaries. The specific details and functionality of a stuff chain can vary depending on the context and configuration.'*

*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*-----------------*-*----------

The Stuff strategy

A diagram of a diagram

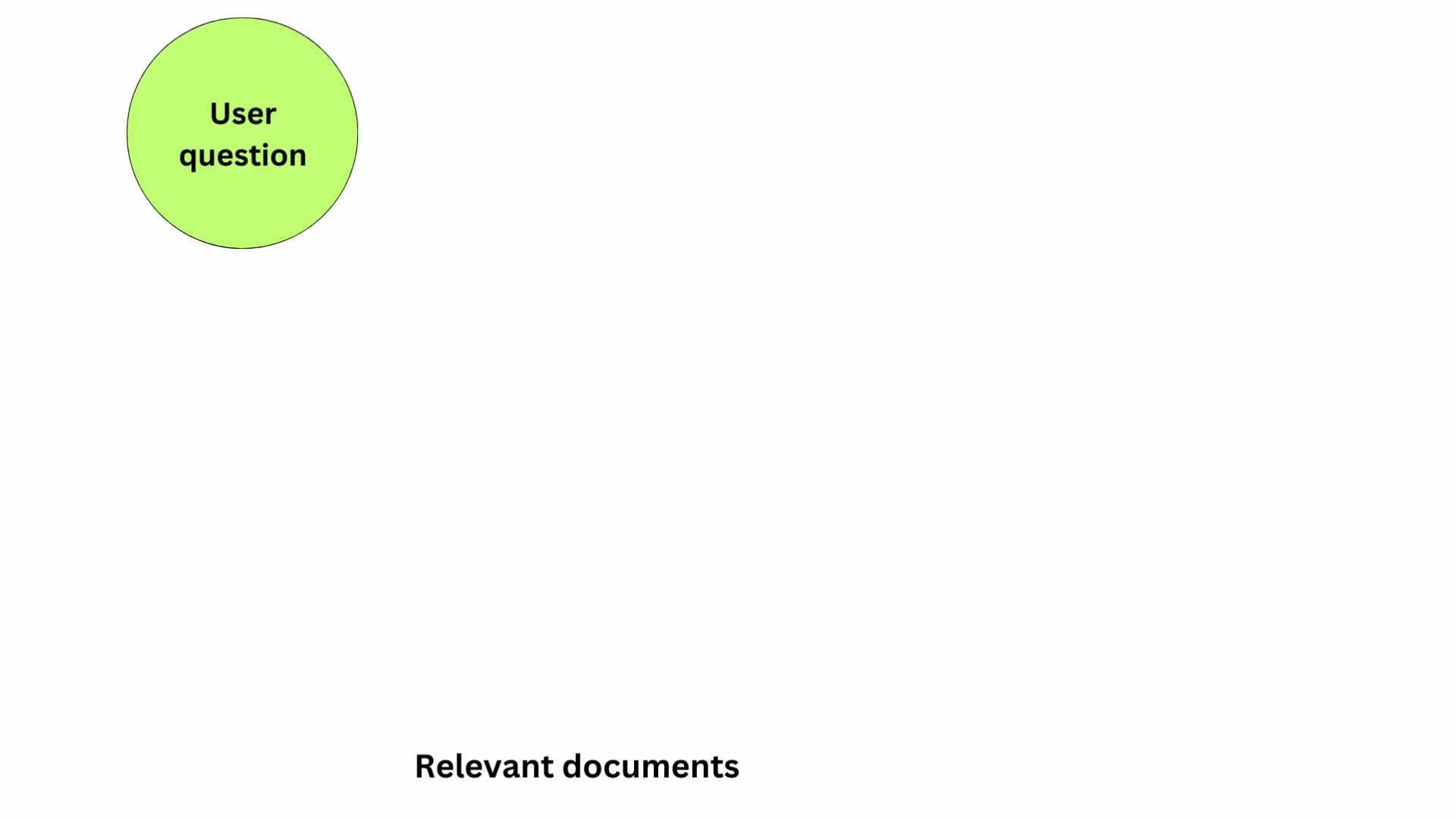
Description automatically generated

The Map-reduce strategy

A diagram of a diagram

Description automatically generated

The Refine strategy



The Map-rerank strategy

A diagram of a question

Description automatically generated