Lang Chain Model IO

Content

- Document loaders
- CSV Loader
- WebBaseLoader
- Wikipedia
- PyPDFLoader
- BSHTMLLoader
- Output Parsers
- CSV Parser
- Datetime Parser
- Custom Parser - Using Pydantic
- App: Email Response App
- Wiki Company Profile App

Document loaders

Document loaders in LangChain

- Document loaders get documents from various sources.
- LangChain offers more than 100 different document loaders.
- Integrations with other big providers like AirByte and Unstructured are included.
- LangChain can load many document types (HTML, PDF, code) from different places (private S3 buckets, public websites).

Document loaders in LangChain

- Document loaders load data from various sources into Documents, which consist of text and metadata.
- There are loaders for .txt files, web pages, and YouTube video transcripts.
- They include a "load" method for loading documents from a source.
- Optionally, they can "lazy load" data, loading it into memory as needed.

Basic Document Loader

```
!wget https://raw.githubusercontent.com/venkatareddykonasani/Datasets/master/Mobile_Phone_Review/Mobile.md

loader = TextLoader("./Mobile.md")
loaded_text= loader.load()
print(type(loaded_text))
print(loaded_text)
```

```
--2024-03-30 17:55:29--  https://raw.githubusercontent.com/venkatareddykonasani/Datasets/master/Mobile_Phone_Review/
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3140 (3.1K) [text/plain]
Saving to: ‘Mobile.md’

Mobile.md           100%[===================>]   3.07K  --.-KB/s    in 0s

2024-03-30 17:55:29 (44.0 MB/s) - ‘Mobile.md’ saved [3140/3140]

<class 'list'>
[Document(page_content="### Samsung Galaxy S22 Ultra Review \n\nSamsung's Galaxy S22 Ultra is the epitome of innovat

, metadata={'source': './Mobile.md'})]
```

Contains Document and metadata

Chain on the loaded document

```
llm=OpenAI(temperature=0)

template="""
read the following review and extract the following information:
Name of the product
Brand of the product
Price of the product
Rating of the product

review is given here : {input_review}
"""

prompt=PromptTemplate(template=template,
                      input_variables=["input_review"])

chain=LLMChain(llm=llm,
               prompt=prompt)
```

Chain on the loaded document

```
result=chain.invoke({"input_review":loaded_text})
print(result['text'])
```

```
Name of the product: Samsung Galaxy S22 Ultra
Brand of the product: Samsung
Price of the product: Not mentioned in the review
Rating of the product: Not mentioned in the review
```

Basic Document Loader – Example2

```
!wget https://raw.githubusercontent.com/venkatareddykonasani/Datasets/ma

loader = TextLoader("./Mail.txt")
loaded_text= loader.load()
print(type(loaded_text))
print(loaded_text)
```

```
--2024-03-30 18:13:40--  https://raw.githubusercontent.com/venkatareddykonasani
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109
HTTP request sent, awaiting response... 200 OK
Length: 2457 (2.4K) [text/plain]
Saving to: ‘Mail.txt’

Mail.txt            100%[===================>]   2.40K  --.-KB/s    in 0s

2024-03-30 18:13:40 (43.3 MB/s) - ‘Mail.txt’ saved [2457/2457]

<class 'list'>
[Document(page_content="Subject: Urgent Attention Required: Project Delay and C
```

Basic Document Loader – Example2

```
llm=OpenAI(temperature=0)

template="""
read the following mail and extract the following information:
Name of the person
Main Point of the mail
Ticket-id
The mail is given here : {input_mail}
"""
prompt=PromptTemplate(template=template,
                      input_variables=["input_mail"])

chain=LLMChain(llm=llm,
               prompt=prompt)

result=chain.invoke({"input_mail":loaded_text})
print(result['text'])
```

Basic Document Loader – Example2

```
result=chain.invoke({"input_mail":loaded_text})
print(result['text'])
```

```
Name of the person: John Doe
Main Point of the mail: Urgent attention required for project delay and critical issues
Ticket-id: UAT1966286RT5
```

CSV Loader

```
from langchain_community.document_loaders.csv_loader import CSVLoader
!wget https://raw.githubusercontent.com/venkatareddykonasani/Datasets/master/Leads.csv

loader = CSVLoader(file_path="./Leads.csv")
csv_file_data = loader.load()
print(type(csv_file_data))
print(csv_file_data)
```

```
--2024-03-30 18:46:22--  https://raw.githubusercontent.com/venkatareddykonasani/Datasets/master
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... con
HTTP request sent, awaiting response... 200 OK
Length: 1493 (1.5K) [text/plain]
Saving to: ‘Leads.csv’

Leads.csv           100%[===================>]   1.46K  --.-KB/s    in 0s

2024-03-30 18:46:22 (26.1 MB/s) - ‘Leads.csv’ saved [1493/1493]

<class 'list'>
[Document(page_content='Week_num: 1\nLeads: 756.48\nPromotion_Budget: 51735.6', metadata={'sour
```

CSV Loader

```
llm=OpenAI(temperature=0)

template="""
read the following data and extract the following information:

What is the average promotional budget ?
What is the average Leads?

data is given here : {input_data}
"""

prompt=PromptTemplate(template=template,
                      input_variables=["input_data"])

chain=LLMChain(llm=llm,
               prompt=prompt)
```

- This works only for small datasets
- We need to use agents to work with CSV data

CSV Loader

```
result=chain.invoke({"input_data":csv_file_data})
print(result['text'])
```

```
Average promotional budget: $60,000.08
Average leads: 890.08
```

Sending full data inside a prompt. This may not work for larger datasets

CSV Loader – Common Issues

```
from langchain_community.document_loaders.csv_loader import CSVLoader
!wget https://raw.githubusercontent.com/venkatareddykonasani/Datasets/

loader = CSVLoader(file_path="./AB_NYC_2019.csv")
csv_file_data = loader.load()
print(type(csv_file_data))
print(csv_file_data)
```

```
--2024-04-01 02:35:12--  https://raw.githubusercontent.com/venkatareddykonasa
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.10
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.1
HTTP request sent, awaiting response... 200 OK
Length: 7077973 (6.8M) [text/plain]
Saving to: ‘AB_NYC_2019.csv’

AB_NYC_2019.csv     100%[===================>]   6.75M  --.-KB/s    in 0.08s

2024-04-01 02:35:12 (84.6 MB/s) - ‘AB_NYC_2019.csv’ saved [7077973/7077973]
```

CSV Loader – Common Issues

```
print(csv_file_data)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

Max tokens issue

CSV Loader – Common Issues

```
llm=OpenAI(temperature=0)

template="""
read the following data and extract the following information:

What is the average price?
What are the top 5 most expensive listings?
What are the top 5 most reviewed listings?

data is given here : {input_data}
"""

prompt=PromptTemplate(template=template,
                      input_variables=["input_data"])

chain=LLMChain(llm=llm,
               prompt=prompt,
               verbose=True)
```

Sending full data inside a prompt. This may not work for larger datasets

CSV Loader – Common Issues

```
chain.invoke({"input_data":csv_file_data})
```

```
> Entering new LLMChain chain...
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)

---------------------------------------------------------------------------
RateLimitError                            Traceback (most recent call last)
<ipython-input-23-35747fb4690a> in <cell line: 1>()
----> 1 chain.invoke({"input_data":csv_file_data})
```

CSV Loader – Common Issues

```
RateLimitError: Error code: 429 - {'error': {'message': 'Request too large for gpt-3.5-turbo-instruct in organization org-
LpzZ7qKTHls32x8DEx6NLota on tokens per min (TPM): Limit 90000, Requested 5662579. The input or output tokens must be
reduced in order to run successfully. Visit https://platform.openai.com/account/rate-limits to learn more.', 'type':
'tokens', 'param': None, 'code': 'rate_limit_exceeded'}}
```

We will use other options like Agents to solve this issue.

WebBaseLoader

```
from langchain_community.document_loaders import WebBaseLoader
```

```
loader = WebBaseLoader("https://www.amazon.in/Indigenous-Unprocessed-

web_file_data = loader.load()
print(type(web_file_data))
print(web_file_data)
print(len(web_file_data[0].page_content))
web_file_data[0].page_content=web_file_data[0].page_content[:10000]
print(len(web_file_data[0].page_content))
```

```
<class 'list'>
[Document(page_content="\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\r
20694
10000
```

Directly get the data from the webpage

Limit it to 10000 characters. Otherwise, we may face the max tokens limit issue

WebBaseLoader

```
llm=OpenAI(temperature=0)

template="""
read the following data summarize it into 4 bullet points
data is given here : {input_data}
"""

prompt=PromptTemplate(template=template,
                      input_variables=["input_data"])

chain=LLMChain(llm=llm,
               prompt=prompt)
```

WebBaseLoader

```
result=chain.invoke({"input_data":web_file_data})
result['text']
```

```
'\n- The data is a collection of customer reviews for a product on Amazon.in.\n- The product is INDIGENOUS HONEY Ra
R tested and NPOP organic certified.\n- The reviews are mostly positive, with an average rating of 4.3 out of 5 sta
taste, quality, sourcing, and packaging of the honey, and recommend it as a delicious and wholesome choice.'
```

Youtube Loader

```
!pip install --upgrade --quiet  youtube-transcript-api
```

```
from langchain_community.document_loaders import YoutubeLoader
loader = YoutubeLoader.from_youtube_url(
    "https://www.youtube.com/watch?v=fbQvVS_8ZNI"
)

youtube_file_data = loader.load()
print(type(youtube_file_data))
print(youtube_file_data)
```

```
<class 'list'>
[Document(page_content="[Music] all healing comes from inside ourselves
```

Video transcript

Youtube Loader

```
llm=OpenAI(temperature=0)

template="""
read the following data summarize it into 4 bullet points
data is given here : {input_data}
"""

prompt=PromptTemplate(template=template,
                      input_variables=["input_data"])

chain=LLMChain(llm=llm,
               prompt=prompt)
```

```
result=chain.invoke({"input_data":youtube_file_data})
result['text']
```

```
'\n- Healing comes from within ourselves and is based on love\n- We have the power to choose and have fr
ee will, but have used it to dominate the Earth\n- The five L's - life, love, laughter, labor, and liste
ning - are essential for understanding ourselves and the world around us\n- Meeting Gandhi at a young ag
e sparked a lifelong connection to the power of love and recognizing others as real beings.'
```

Wikipedia

```
!pip install wikipedia

from langchain.document_loaders import WikipediaLoader

loader=WikipediaLoader(query="Sundar Pichai", load_max_docs=1)
wiki_file_data=loader.load()
print(type(wiki_file_data))
print(wiki_file_data)
```

Limit it to a couple of documents to avoid max tokens error

PyPDFLoader

```
!pip install pypdf
from langchain.document_loaders import PyPDFLoader
loader=PyPDFLoader(file_path="Regulatory_Rules_in_Credit_Risk_Models.pdf")
```

BSHTMLLoader

```
!pip install beautifulsoup4
from langchain.document_loaders import BSHTMLLoader
loader=BSHTMLLoader(file_path="South_Asia_Global_Debt_Summary.html")
```

Many more Integrations

Activeloop Deep Lake
AI21 Labs
Aim
AINetwork
Airbyte
Airtable
Aleph Alpha
Alibaba Cloud
AnalyticDB
Annoy
Anyscale
Apache Doris
Apify
ArangoDB
Arcee
Argilla
Arthur
Arxiv
Astra DB
Atlas
AwaDB
AZLyrics
BagelDB
Baichuan
Baidu
Banana
Baseten
Beam
Beautiful Soup
BiliBili
Bittensor
Blackboard
Brave Search
Chroma
Clarifai
ClearML
ClickHouse
Cloudflare
CnosDB
Cohere
College Confidential
Comet
Confident AI
Confluence
Context
C Transformers
CTranslate2
DashVector
Databricks
Datadog Tracing
Datadog Logs
DataForSEO
DeepInfra
DeepSparse
Diffbot
DingoDB
Discord
DocArray
Doctran
Docugami
DSPy
DuckDB
Eden AI
Elasticsearch
ElevenLabs
CerebriumAI
Epsilla
EverNote
Exa Search
Facebook - Meta
Fiddler
Figma
Fireworks
Flyte
ForefrontAI
Git
GitBook
Golden
Serper - Google Search API
GooseAI
GPT4All
Gradient
Graphsignal
Grobid
Groq
Gutenberg
Hacker News
Hazy Research
Helicone
Hologres
HTML to text
IBM
iFixit
IMSDb
Infinispan VS
Infinity
Infino
Intel
Jaguar
Javelin AI Gateway
Jina
Johnsnowlabs
KDB.AI
Kinetica
Konko
Label Studio
LanceDB
LangChain Decorators ✨
Lantern
Llama.cpp
LLMonitor
Log10
Marqo
MediaWikiDump
Meilisearch
Metal
Milvus
Minimax
MistralAI
MLflow Deployments for LLMs
MLflow AI Gateway
MLflow
Modal
ModelScope
Modern Treasury
Momento
MongoDB Atlas
Motherduck
Motörhead
MyScale
Neo4j
NLPCloud
Nuclia
NVIDIA
Obsidian
Oracle Cloud Infrastructure (OCI)
Ollama
Ontotext GraphDB
OpenLLM
OpenSearch
OpenWeatherMap
Outline
Petals
Postgres Embedding
PGVector
Pinecone
PipelineAI
Portkey
Predibase
Prediction Guard
PremAI
PromptLayer
PubMed
PygmalionAI
Qdrant
RAGatouille
Ray Serve
Rebuff
Reddit
Redis
Remembrall
Replicate
Chaindesk
Nomic
Roam
Robocorp
Rockset
Runhouse
RWKV-4
Salute Devices
SearchApi
SearxNG Search API
SemaDB
SerpAPI
Shale Protocol
SingleStoreDB
scikit-learn
Slack
spaCy
SparkLLM
Spreedly
SQLite
Stack Exchange
StarRocks
StochasticAI
Streamlit
Stripe
Supabase (Postgres)
Nebula
Tair
Telegram
Tencent
TensorFlow Datasets
TiDB
Cassandra
Notion DB
TigerGraph
Tigris
Together AI
2Markdown
Trello
Trubrics
TruLens
Twitter
Typesense
Unstructured
Upstash Redis
USearch
VDMS
Vearch
Vectara
Vespa
VoyageAI
WandB Tracing
Weights & Biases
Weather
Weaviate
WhatsApp
WhyLabs
Wikipedia
Wolfram Alpha
Writer
Xata
Xorbits Inference (Xinference)
Yandex
Yeager.ai
YouTube
Zep
Zilliz

Output Parsers

Output parsers

- Output parsers transform the output of language models (LLMs) into a more structured and suitable format
- LangChain OutputParsers feature a large collection of parsers for different types of structured outputs.
- Output parsers are essential for converting the textual output of language models into more structured information.

Two main methods

There are two main methods(functions) in an output parser

Get format instructions

- Gives you steps on how to arrange language model outputs.

Parse

- Changes a language model's text response into a structured form.

Parse with prompt (optional)

- This also turns a language model's response into a structured form but uses the original question for help.
- The original question can help fix or retry shaping the answer if needed.
- This method is optional and helps when more context is needed to structure the response.

CSV Parser

```
from langchain.output_parsers import CommaSeparatedListOutputParser
```

```
output_parser = CommaSeparatedListOutputParser()
format_instructions = output_parser.get_format_instructions()
print(format_instructions)
```

```
Your response should be a list of comma separated values, eg: `foo, bar, baz`
```

Simple prompt generated in system

CSV Parser

```
template="""
List down the top 10 concepts to learn from the following Topic
Topic name is: {topic_name}
{format_instructions}
"""
prompt=PromptTemplate(template=template,
                      input_variables=["topic_name", "format_instructions"])

llm=OpenAI(temperature=0)
chain=LLMChain(llm=llm,
               prompt=prompt)

result=chain.invoke({"topic_name":"Machine Learning",
                "format_instructions":format_instructions })
print(result["text"])
```

CSV Parser

```
result=chain.invoke({"topic_name":"Machine Learning",
                     "format_instructions":format_instructions })
print(result["text"])
```

```
1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning
4. Deep Learning
5. Neural Networks
6. Decision Trees
7. Support Vector Machines
8. Clustering
9. Dimensionality Reduction
10. Model Evaluation and Selection
```

Datetime Parser

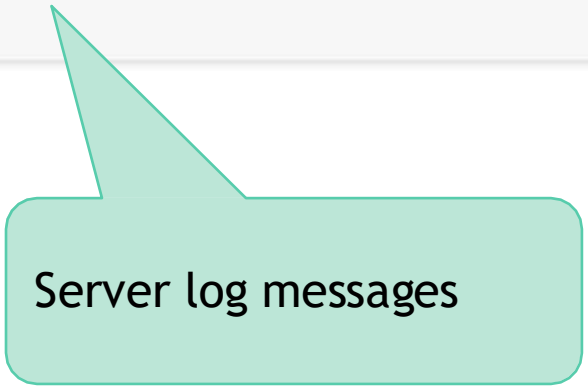
```python
from langchain.output_parsers import DatetimeOutputParser
output_parser = DatetimeOutputParser()
format_instructions=output_parser.get_format_instructions()
print(format_instructions)
```

```
Write a datetime string that matches the following pattern: '%Y-%m-%dT%H:%M:%S.%fZ'.

Examples: 1727-03-07T01:08:50.024216Z, 1753-06-25T10:03:48.396187Z, 1447-05-19T18:18:11.329552Z

Return ONLY this string, no other words!
```

Format instructions

Datetime Parser

```
Server_Logs =[
    "[2024-04-01 13:48:11] ERROR: Failed to connect to database. Retrying in 60 seconds.",
    "[2023-08-04 12:01:00 AM - Warning: The system is running low on disk space.",
    "[04-01-2024 13:55:39] CRITICAL: System temperature exceeds safe threshold. Initiating shutdown"
    "[Monday, April 01, 2024 01:55:39 PM] DEBUG: User query executed in 0.45 seconds.",
    "[13:55:39 on 2024-04-01] ERROR: Unable to send email notification. SMTP server not responding."
]
```



Datetime Parser

```
template="""
Read the server log text and extract the date and time
log text is: {log_text}
{format_instructions}
"""
prompt=PromptTemplate(template=template,
                      input_variables=["log_text", "format_instructions"])

llm=OpenAI(temperature=0)
chain=LLMChain(prompt=prompt,
               llm=llm)
```

Datetime Parser

```
for log_message in Server_Logs:
    result=chain.invoke({"log_text":log_message,
                         "format_instructions":format_instructions })
    print(result["text"],";", log_message )
```

```
2024-04-01T13:48:11.000000Z ; [2024-04-01 13:48:11] ERROR: Failed to connect to database. Retrying in 60 seconds.

2023-08-04T00:01:00.000000Z ; [2023-08-04 12:01:00 AM - Warning: The system is running low on disk space.

2024-04-01T13:55:39.000000Z ; [04-01-2024 13:55:39] CRITICAL: System temperature exceeds safe threshold. Initiating shutdown

2024-04-01T13:55:39.000000Z ; [Monday, April 01, 2024 01:55:39 PM] DEBUG: User query executed in 0.45 seconds.

2024-04-01T13:55:39.000000Z ; [13:55:39 on 2024-04-01] ERROR: Unable to send email notification. SMTP server not responding.
```

Custom Parser - Using Pydantic

- You can specify any Pydantic Model and query LLMs for outputs that are structured in a particular format
- You need to have some basic idea on the Pydantic data validation package that is widely used in building applications using frameworks like flask
- With Pydantic, you tell your program what kind of data it should accept (like numbers, text, or dates) using simple rules.

Pydantic Package

- Pydantic ensures that the data you receive matches your specifications. If the data is incorrect or in an unexpected format, Pydantic will raise an error, highlighting exactly what went wrong.
- If the data matches the rules, your program works smoothly. If not, Pydantic helps by pointing out the problem, making it easier to keep your program safe and error-free.

```python
from pydantic import BaseModel

class User(BaseModel):
    name: str
    age: int

# Correct data
user = User(name="Alice", age=30)
print(user)

# Incorrect data raises an error
User(name="Bob", age="thirty")
```

Pydantic

```
User(name="Bob", age="thirty")
```

Failed data validation

```
name='Alice' age=30
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
<ipython-input-7-ed292bcafae2> in <cell line: 12>()
     10
     11 # Incorrect data raises an error
---> 12 User(name="Bob", age="thirty")

/usr/local/lib/python3.10/dist-packages/pydantic/main.py in __init__(self, **data)
    169         # `__tracebackhide__` tells pytest and some other tools to omit this function from
tracebacks
    170         __tracebackhide__ = True
--> 171         self.__pydantic_validator__.validate_python(data, self_instance=self)
    172
    173     # The following line sets a flag that we use to determine when `__init__` gets overridden
by the user

ValidationError: 1 validation error for User
age
  Input should be a valid integer, unable to parse string as an integer [type=int_parsing,
input_value='thirty', input_type=str]
    For further information visit https://errors.pydantic.dev/2.6/v/int_parsing
```

Pydantic

```
from langchain.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field
```

```
class Scientist(BaseModel):
    name: str= Field(description= "Name of the Scientist")
    dob: str= Field(description= "Date of Birth of the Scientist")
    bio: str= Field(description= "Biography of the Scientist")
```

Pydantic

```
custom_output_parser= PydanticOutputParser(pydantic_object=Scientist)
print(custom_output_parser.get_format_instructions())
```

```
The output should be formatted as a JSON instance that conforms to the JSON schema below.

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description": "a list of strings", "t
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The object {"properties": {"f

Here is the output schema:
```
{"properties": {"name": {"description": "Name of the Scientist", "title": "Name", "type": "string"}, "dob":
```
```

Pydantic

```
template="""
Take the name of the scientist is {name} and try to fill the rest of the details
{format_instructions}
"""
prompt=PromptTemplate(template=template,
                      input_variables=["name", "format_instructions"])

llm=OpenAI(temperature=0)
chain=LLMChain(prompt=prompt,
               llm=llm)

result=chain.invoke({"name":"Ramanujan",
                     "format_instructions":custom_output_parser.get_format_instructions() })
print(result["text"])
```

```
Here is the output instance:
```

```
{"name": "Ramanujan", "dob": "22 December 1887", "bio": "Srinivasa Ramanujan was an Indian mathematicia
```

App: Email Response App

Email Response App

1. **Collect Customer Communication**: Obtain customer's email or message.[Written in any language]
2. **Language Identification**: Detect the language of the message.
3. **Translate to English**: Translate message to English.
4. **Summarize Key Points**: Document critical points from the email.
5. **Craft Preliminary Response**: Draft a concise, empathetic initial reply.

```
email_location="https://raw.githubusercontent.com/venkatareddykonasani/Datasets/master/
Customer_Emails/Mail"+str(rand.randint(1,5))+".txt"
print(email_location)

loader = WebBaseLoader(email_location)
loaded_text= loader.load()
print(type(loaded_text))
final_mail=loaded_text[0].page_content
print(final_mail)
```

Randomly takes a language

```
Тема: Проблемы с последним заказом

Уважаемая служба поддержки,

Здравствуйте,

Я обращаюсь к вам с проблемами, с которыми столкнулась после оформления последнего заказ

Пожалуйста, предоставьте мне информацию о том, как я могу вернуть неподходящий товар и г

Спасибо за вашу помощь и ожидаю вашего скорого ответа.

С уважением,
Ольга Петрова
```

```python
from langchain.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field

class EmailResponse(BaseModel):
  Email_Language: str= Field(description= "The Original Language of the Email")
  Customer_ID: str= Field(description= "The Customer ID mentioned in the mail")
  English_email: str= Field(description= "The email after translating to English")
  Summary: str= Field(description= "A 4 bullets point summary of the email")
  Reply: str= Field(description= "A polite 2 line reply to the email")

custom_output_parser= PydanticOutputParser(pydantic_object=EmailResponse)
print(custom_output_parser.get_format_instructions())
```

```
The output should be formatted as a JSON instance that conforms to the JSON schema be

As an example, for the schema {"properties": {"foo": {"title": "Foo", "description":
the object {"foo": ["bar", "baz"]} is a well-formatted instance of the schema. The ob

Here is the output schema:
```
{"properties": {"Email_Language": {"description": "The Original Language of the Email
```
```

```
template="""
Take the email as input. Email text is {email}
{format_instructions}
"""
prompt=PromptTemplate(template=template,
                      input_variables=["email","format_instructions"])

llm=OpenAI(temperature=0)

chain=LLMChain(prompt=prompt,
               llm=llm)

result=chain.invoke({"email":final_mail,
                     "format_instructions":custom_output_parser.get_format_instructions()})
print(result["text"])
```


```
{
  "Email_Language": "Russian",
  "Customer_ID": "67890ABC",
  "English_email": "Subject: Issues with my last order\n\nDear customer support,\n\nI am writing
  "Summary": "- Product received does not match description\n- One item missing from order\n- Rec
  "Reply": "Dear Olga Petrova,\n\nThank you for bringing this to our attention. We apologize for
```

Wiki Company Profile App

Wiki Company Profile App

- Take the company name as input.
- Extract the Company Details from Wikipedia:
  - The founder of the company.
  - When it was founded.
  - The current market capital of the company.
  - How many employees are working in it.
- Provide a brief 4-line summary of the company.

```python
from langchain.document_loaders import WikipediaLoader
loader=WikipediaLoader(query="Bharti Airtel",  load_max_docs=1)
company_documents=loader.load()
print(company_documents)
```

```python
from langchain.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field

class CompanyProfile(BaseModel):
  Company_Name: str= Field(description= "The Name of the Company")
  Founder: str= Field(description= "The Founder of the Company")
  Start_Date: str= Field(description= "The date or the founding year of the compnay")
  Revenue: int= Field(description= "The Revenue of the company")
  Employees: str= Field(description= "How many employees are working in it")
  Summary: str= Field(description= "Provide a brief 4-line summary of the company")

custom_output_parser= PydanticOutputParser(pydantic_object=CompanyProfile)
print(custom_output_parser.get_format_instructions())
```

```
template="""
Take the company wiki page information as input
Company Details from Wikipedia:{wiki_page_info}
{format_instructions}
"""
prompt=PromptTemplate(template=template,
                      input_variables=["wiki_page_info","format_instructions"])

llm=OpenAI(temperature=0)

chain=LLMChain(prompt=prompt,
               llm=llm)

result=chain.invoke({"wiki_page_info":company_documents,
                     "format_instructions":custom_output_parser.get_format_instructions()})
print(result["text"])
```

```
'{"Company_Name": "Bharti Airtel", "Founder": "Sunil Mittal", "Start_Date": "1984", "Revenue": 191.5, "E
mployees": "21,299", "Summary": "Bharti Airtel is an Indian multinational telecommunications services co
mpany founded by Sunil Mittal in 1984. It operates in 18 countries across South Asia and Africa, and is
the second largest mobile network operator in India and the world."}'
```

Thank you