

# LangChain

LangChain:

- LLMを使って自然言語処理のタスクを効率的に行うためのライブラリ
- LLMに対して直接的に操作を行うことで、LLMの内部状態や出力を変化させることができる
- LLMに本やプログラムを渡し、外部の知識や計算能力（Web検索）を利用できるようにする役割があるためLLMでできないこともLangChainではできる
- LLMをwebサービスや自作のAPI
- プログラムの実行環境
- ターミナルなどに接続するライブラリ

提供するコンポーネント:

コンポーネント	説明
Models	OpenAIをはじめとした様々な言語モデル・チャットモデル・エンベディングモデルを切り替えたり、組み合わせたりすることができる
Prompt	プロンプトの管理・最適化・シリアル化などをすることができます

Indexes| PDFやCSVなどの外部データを用いて回答を生成することができる| Chains| 複数のプロンプト入力を実行することができる| Agents| 言語モデルに渡されたツールを用いて、モデル自体が次にどのようなアクションを取るかを決定し、実行し、観測し、完了するまで繰り返すことができる| Memory| ChainsやAgentsの内部における状態保持をすることができる|

<https://github.com/hwchase17/langchain>

<https://python.langchain.com/en/latest/>

## LangChainのモジュール

機能	説明
LLM	LLM呼び出しを行うための共通インターフェース
プロンプトテンプレート	ユーザー入力からのプロンプトを生成
チェーン	複数のLLMやプロンプトの入出力を繋げる

機能	説明
エージェント	ユーザーの要求に応じて、どの機能をどういう順番で実行するかを決定
ツール	エージェントが実行する特定の機能
メモリ	チェーンやエージェントの記憶を保持

## LangChainの実践例

LangChainモジュールでChatGPTのようにGoogleColabを使えるようになる様々な実践例がある。

数ある例のなかから使い用途または作りたいアプリの目的にあったもの

## 環境準備

```
In [ ]: !pip install cohere
```

```
Collecting cohere
  Downloading cohere-4.32-py3-none-any.whl (47 kB)
[██████████] 48.0/48.0 kB 1.4 MB/s eta 0:00
Requirement already satisfied: aiohttp<4.0,>=3.0 in /usr/local/lib/python3.10/dist-packages (from cohere) (3.8.6)
Collecting backoff<3.0,>=2.0 (from cohere)
  Downloading backoff-2.2.1-py3-none-any.whl (15 kB)
Collecting fastavro==1.8.2 (from cohere)
  Downloading fastavro-1.8.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.7 MB)
[██████████] 2.7/2.7 MB 27.5 MB/s eta 0:00
Requirement already satisfied: importlib_metadata<7.0,>=6.0 in /usr/local/lib/python3.10/dist-packages (from cohere) (6.8.0)
Requirement already satisfied: requests<3.0.0,>=2.25.0 in /usr/local/lib/python3.10/dist-packages (from cohere) (2.31.0)
Requirement already satisfied: urllib3<3,>=1.26 in /usr/local/lib/python3.10/dist-packages (from cohere) (1.26.18)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere) (23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere) (3.3.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere) (6.0.4)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere) (4.0.3)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere) (1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0,>=3.0->cohere) (1.3.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from importlib_metadata<7.0,>=6.0->cohere) (3.17.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.25.0->cohere) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.25.0->cohere) (2023.7.22)
Installing collected packages: fastavro, backoff, cohere
Successfully installed backoff-2.2.1 cohere-4.32 fastavro-1.8.2
```

```
In [ ]: !pip install tiktoken
```

```
Requirement already satisfied: tiktoken in /usr/local/lib/python3.10/dist-packages (0.5.1)
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.10/dist-packages (from tiktoken) (2023.6.3)
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.10/dist-packages (from tiktoken) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.26.0->tiktoken) (2023.7.22)
```

```
In [ ]: !pip install -q llama-index  
!pip install -q openai  
!pip install -q transformers  
!pip install -q accelerate  
!pip install -q langchain  
!pip install -q faiss-gpu
```

```
In [ ]: # パッケージのインストール  
!pip install langchain==0.0.303
```

```
Collecting langchain==0.0.303
  Downloading langchain-0.0.303-py3-none-any.whl (1.7 MB)
    1.7/1.7 MB 18.5 MB/s eta 0:0
0:00
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (6.0.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (2.0.22)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (3.8.6)
Requirement already satisfied: anyio<4.0 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (3.7.1)
Requirement already satisfied: async-timeout<5.0.0,>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (4.0.3)
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (0.5.14)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (1.33)
Requirement already satisfied: langsmith<0.1.0,>=0.0.38 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (0.0.51)
Requirement already satisfied: numexpr<3.0.0,>=2.8.4 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (2.8.7)
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (1.23.5)
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (1.10.13)
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (2.31.0)
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain==0.0.303) (8.2.3)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain==0.0.303) (23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain==0.0.303) (3.3.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain==0.0.303) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain==0.0.303) (1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain==0.0.303) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain==0.0.303) (1.3.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<4.0->langchain==0.0.303) (3.4)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4.0->langchain==0.0.303) (1.3.0)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4.0->langchain==0.0.303) (1.1.3)
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-json<0.7,>=0.5.7->langchain==0.0.303) (3.20.1)
Requirement already satisfied: typing-inspect<1,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-json<0.7,>=0.5.7->langchain==0.0.303) (0.9.0)
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.10/dist-packages (from jsonpatch<2.0,>=1.33->langchain==0.0.303) (2.4)
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/
```

```
python3.10/dist-packages (from pydantic<3,>=1->langchain==0.0.303) (4.5.0)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python
3.10/dist-packages (from requests<3,>=2->langchain==0.0.303) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
3.10/dist-packages (from requests<3,>=2->langchain==0.0.303) (2023.7.22)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.
10/dist-packages (from SQLAlchemy<3,>=1.4->langchain==0.0.303) (3.0.0)
Requirement already satisfied: packaging>=17.0 in /usr/local/lib/python3.1
0/dist-packages (from marshmallow<4.0.0,>=3.18.0->dataclasses-json<0.7,>=
0.5.7->langchain==0.0.303) (23.2)
Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/py
thon3.10/dist-packages (from typing-inspect<1,>=0.4.0->dataclasses-json<0.
7,>=0.5.7->langchain==0.0.303) (1.0.0)
Installing collected packages: langchain
  Attempting uninstall: langchain
    Found existing installation: langchain 0.0.322
    Uninstalling langchain-0.0.322:
      Successfully uninstalled langchain-0.0.322
    Successfully installed langchain-0.0.303
```

In [ ]: `!pip install --upgrade langchain`

```
Requirement already satisfied: langchain in /usr/local/lib/python3.10/dist-packages (0.0.319)
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-packages (from langchain) (6.0.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/dist-packages (from langchain) (2.0.22)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.10/dist-packages (from langchain) (3.8.6)
Requirement already satisfied: anyio<4.0 in /usr/local/lib/python3.10/dist-packages (from langchain) (3.7.1)
Requirement already satisfied: async-timeout<5.0.0,>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from langchain) (4.0.3)
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in /usr/local/lib/python3.10/dist-packages (from langchain) (0.5.14)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.10/dist-packages (from langchain) (1.33)
Requirement already satisfied: langsmith<0.1.0,>=0.0.43 in /usr/local/lib/python3.10/dist-packages (from langchain) (0.0.47)
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain) (1.23.5)
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain) (1.10.13)
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-packages (from langchain) (2.31.0)
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain) (8.2.3)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (3.3.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain) (1.3.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<4.0->langchain) (3.4)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4.0->langchain) (1.3.0)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4.0->langchain) (1.1.3)
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-json<0.7,>=0.5.7->langchain) (3.20.1)
Requirement already satisfied: typing-inspect<1,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-json<0.7,>=0.5.7->langchain) (0.9.0)
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.10/dist-packages (from jsonpatch<2.0,>=1.33->langchain) (2.4)
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langchain) (4.5.0)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain) (2023.7.22)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.
```

```
10/dist-packages (from SQLAlchemy<3,>=1.4->langchain) (3.0.0)
Requirement already satisfied: packaging>=17.0 in /usr/local/lib/python3.1
0/dist-packages (from marshmallow<4.0.0,>=3.18.0->dataclasses-json<0.7,>=
0.5.7->langchain) (23.2)
Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/py
thon3.10/dist-packages (from typing-inspect<1,>=0.4.0->dataclasses-json<0.
7,>=0.5.7->langchain) (1.0.0)
```

```
In [ ]: # 環境変数の準備
import os
os.environ["OPENAI_API_KEY"] = "your_api_key"
```

## LLM

```
In [ ]: from langchain.llms import OpenAI

# LLMの準備
llm = OpenAI(temperature=0.9)

# LLMの呼び出し
print(llm("こんにちは"))

", "こんにちは!")

# 同じ内容がある場合は削除

similar_array.uniq
```

## プロンプトテンプレート

```
In [ ]: from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

# プロンプトテンプレートの作成
prompt = PromptTemplate(
    input_variables=["product"],
    template="{product}を作る日本語の新会社名を1つ提案してください",
)

# プロンプトの作成
print(prompt.format(product="家庭用ロボット"))
```

家庭用ロボットを作る日本語の新会社名を1つ提案してください

## チェーン

```
In [ ]: from langchain.chains import LLMChain
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate

# プロンプトテンプレートの作成
prompt = PromptTemplate(
    input_variables=["product"],
    template="{product}を作る日本語の新会社名を1つ提案してください",
)
```

```
# チェーンの作成
chain = LLMChain(
    llm=OpenAI(temperature=0.9),
    prompt=prompt
)

# チェーンの実行
chain.run("家庭用ロボット")
```

Out[ ]: '\n\nロボットホームズ'

## エージェント

### Google Search API

<https://serpapi.com>

```
In [ ]: # パッケージのインストール
!pip install google-search-results
```

```
In [ ]: # 環境変数の準備
import os
os.environ["SERPAPI_API_KEY"] = "your_api_key"
```

```
In [ ]: from langchain.agents import load_tools
from langchain.llms import OpenAI

# ツールの準備
tools = load_tools(
    tool_names=["serpapi", "llm-math"],
    llm=OpenAI(temperature=0)
)
```

```
In [ ]: from langchain.agents import initialize_agent

# エージェントの作成
agent = initialize_agent(
    agent="zero-shot-react-description",
    llm=OpenAI(temperature=0),
    tools=tools,
    verbose=True
)
```

```
In [ ]: # エージェントの実行
agent.run("123*4を計算機で計算してください")
```

```
> Entering new AgentExecutor chain...
I need to use a calculator to solve this
Action: Calculator
Action Input: 123*4
Observation: Answer: 492
Thought: I now know the final answer
Final Answer: 492
```

```
> Finished chain.
```

```
Out[ ]: '492'
```

```
In [ ]: # エージェントの実行  
agent.run("今日の東京の天気をWeb検索してください。")
```

```
> Entering new AgentExecutor chain...  
I need to find out the weather in Tokyo today.  
Action: Search  
Action Input: Tokyo weather today  
Observation: 10 Day Weather-Minato-ku, Tokyo Prefecture, Japan. As of 11:10 am JST. Thunderstorm Advisory +2 More. Today. 80°/61°. 1%. Fri 20 | Day. 80°. 1%. SSW 21 mph.  
Thought: I now know the final answer  
Final Answer: 今日の東京の天気は、80°/61°で、1%の確率で雷注意報があり、南西21マイルの風が吹いています。
```

```
> Finished chain.
```

```
Out[ ]: '今日の東京の天気は、80°/61°で、1%の確率で雷注意報があり、南西21マイルの風が吹いています。'
```

## メモリ

```
In [ ]: from langchain.chains import ConversationChain  
from langchain.llms import OpenAI  
  
# 会話チェーンの作成  
chain = ConversationChain(  
    llm=OpenAI(temperature=0),  
    verbose=True  
)
```

```
In [ ]: # チェーンの実行  
chain.run("うちの猫の名前は白子です")
```

```
> Entering new ConversationChain chain...  
Prompt after formatting:  
The following is a friendly conversation between a human and an AI. The AI is talkative and provides lots of specific details from its context. If the AI does not know the answer to a question, it truthfully says it does not know.
```

```
Current conversation:
```

```
Human: うちの猫の名前は白子です
```

```
AI:
```

```
> Finished chain.
```

```
Out[ ]: 'あなたの猫の名前は白子ですね!白子はどんな猫ですか?'
```

```
In [ ]: # チェーンの実行  
chain.predict(input="うちの猫の名前を呼んでください")
```

```
> Entering new ConversationChain chain...
Prompt after formatting:
The following is a friendly conversation between a human and an AI. The AI
is talkative and provides lots of specific details from its context. If the
AI does not know the answer to a question, it truthfully says it does not
know.

Current conversation:
Human: うちの猫の名前は白子です
AI: あなたの猫の名前は白子ですね!白子はどんな猫ですか?
Human: うちの猫の名前を呼んでください
AI:

> Finished chain.
```

Out[ ]: 'わかりました!白子!どうしましよう?'

## テキスト生成モデルのLLM呼び出し

```
In [ ]: from langchain.llms import OpenAI

# LLMの準備
llm = OpenAI(
    model_name="text-davinci-003", # モデルID
    temperature=0 # ランダムさ
)
```

```
In [ ]: # LLMの呼び出し
result = llm("ネコの鳴き声は?")
print(result)
```

ネコの鳴き声は「ニャー」という音です。

```
In [ ]: # 高度なLLMの呼び出し
result = llm.generate(["猫の鳴き声は?", "カラスの鳴き声は?"])

# 出力テキスト
print("response0:", result.generations[0][0].text)
print("response1:", result.generations[1][0].text)

# 使用したトークン数
print("llm_output:", result.llm_output)
```

response0:

ネコの鳴き声は「ニャー」という音です。

response1:

カラスの鳴き声は「カーカー」という鳴き声です。

llm\_output: {'token\_usage': {'total\_tokens': 81, 'prompt\_tokens': 26, 'completion\_tokens': 55}, 'model\_name': 'text-davinci-003'}

## チャットモデルのLLM呼び出し

```
In [ ]: from langchain.chat_models import ChatOpenAI
```

```
# LLMの準備
chat_llm = ChatOpenAI(
    model_name="gpt-3.5-turbo", # モデルID
    temperature=0 # ランダムさ
)
```

```
In [ ]: from langchain.schema import (
    SystemMessage,
    HumanMessage,
    AIMessage
)

# LLMの呼び出し
messages = [
    HumanMessage(content="ネコの鳴き声は?"))
]
result = chat_llm(messages)
print(result)
```

content='ネコの鳴き声は「にゃー」「にゃん」「みゃー」「みゃん」などと表現されることが一般的です。ただし、ネコによって個体差があり、鳴き声の種類や音の大きさ、長さなども異なる場合があります。' additional\_kwargs={} example=False

```
In [ ]: # 高度なLLMの呼び出し
messages_list = [
    [HumanMessage(content="ネコの鳴き声は?")],
    [HumanMessage(content="カラスの鳴き声は?")]
]
result = chat_llm.generate(messages_list)

# 出力テキスト
print("response0:", result.generations[0][0].text)
print("response1:", result.generations[1][0].text)

# 使用したトークン数
print("llm_output:", result.llm_output)
```

response0: ネコの鳴き声は「にゃー」「にゃん」「みゃー」「みゃん」などと表現されることが一般的です。ただし、ネコによって個体差があり、鳴き声の種類や音の大きさ、長さなども異なる場合があります。

response1: カラスの鳴き声は「カーカー」という声で知られています。また、さまざまな鳴き声を発することもあり、鳴き声の種類は個体や状況によって異なることがあります。

llm\_output: {'token\_usage': {'prompt\_tokens': 36, 'completion\_tokens': 176, 'total\_tokens': 212}, 'model\_name': 'gpt-3.5-turbo'}

## キヤッッシュの有効化

```
In [ ]: import langchain
from langchain.cache import InMemoryCache

# キヤッッシュの有効化
langchain.llm_cache = InMemoryCache()
```

```
In [ ]: # 初回のLLM呼び出し
llm.generate([{"空の色は?"}])
```

```
Out[ ]: LLMResult(generations=[[Generation(text='\n\n空の色は青色です。', generation_info={'finish_reason': 'stop', 'logprobs': None})]], llm_output={'token_usage': {'total_tokens': 30, 'prompt_tokens': 11, 'completion_tokens': 19}, 'model_name': 'text-davinci-003'})
```

```
In [ ]: # 2回目以降のLLM呼び出し  
llm.generate(["空の色は?"])
```

```
Out[ ]: LLMResult(generations=[[Generation(text='\n\n空の色は青色です。', generation_info={'finish_reason': 'stop', 'logprobs': None})]], llm_output={})
```

## 特定のLLMのキャッシングの無効化

```
In [ ]: # 特定のLLMのキャッシングの無効化  
llm = OpenAI(cache=False)
```

```
In [ ]: # LLM呼び出し  
llm.generate(["空の色は?"])
```

```
Out[ ]: LLMResult(generations=[[Generation(text='\n\n空の色は青色です。', generation_info={'finish_reason': 'stop', 'logprobs': None})]], llm_output={'token_usage': {'total_tokens': 30, 'prompt_tokens': 11, 'completion_tokens': 19}, 'model_name': 'text-davinci-003'})
```

## キャッシングの無効化

```
In [ ]: # キャッシュの無効化  
langchain.llm_cache = None
```

```
In [ ]: # LLM呼び出し  
llm.generate(["空の色は?"])
```

```
Out[ ]: LLMResult(generations=[[Generation(text='\n\n空の色は青です。', generation_info={'finish_reason': 'stop', 'logprobs': None})]], llm_output={'token_usage': {'total_tokens': 27, 'prompt_tokens': 11, 'completion_tokens': 16}, 'model_name': 'text-davinci-003'})
```

## LLMの非同期処理

```
In [ ]: import time  
from langchain.llms import OpenAI  
  
# 同期処理で10回呼び出しの関数  
def generate_serially():  
    llm = OpenAI(temperature=0.9)  
    for _ in range(10):  
        resp = llm.generate(["こんにちは"])  
        print(resp.generations[0][0].text)  
  
# 時間計測の開始  
s = time.perf_counter()  
  
# 非同期処理で10回呼び出し  
generate_serially()
```

```
# 時間計測の完了
elapsed = time.perf_counter() - s
print(f"{elapsed:.2f} 秒")
```

こんにちは!  
!

お元気ですか?

元気です!ありがとうございます。今日も一日頑張ります!

私はあなたの状況を理解していますが、質問をさせていただくと、問題を解決する方法を提案できます。まず、あなたが抱えている問題を明確にして、どのような環境で対応が必要なのかを考えてください。その後、解決のための具体的なステップを設定することで、問題を解決することができます。また、質問している他の分野の専門家に助言を求め、解決に役立  
!

はじめまして、私の名前はマイケルです。よろしくお願ひします!

お元気ですか?

はい、元気です!ありがとうございます。  
はい

こんにちは!いい天気ですね!

こんにちは。

こんにちは!

こんにちは。

こんにちは!  
11.59 秒

```
In [ ]: import asyncio

# イベントループをネストする設定
import nest_asyncio
nest_asyncio.apply()

# 非同期処理で1回呼び出しの関数
async def async_generate(llm):
    resp = await llm.agenerate(["こんにちは!"])
    print(resp.generations[0][0].text)

# 非同期処理で10回呼び出しの関数
async def generate_concurrently():
    llm = OpenAI(temperature=0.9)
    tasks = [async_generate(llm) for _ in range(10)]
    await asyncio.gather(*tasks)
```

```
# 時間計測の開始
s = time.perf_counter()

# 非同期処理で10回呼び出し
asyncio.run(generate_concurrently())

# 時間計測の完了
elapsed = time.perf_counter() - s
print(f"{elapsed:0.2f} 秒")
```

はじめまして！私はJohnです。どうぞよろしくお願ひします！

お元気ですか？

はい、元気です！ありがとうございます！

お元気ですか？

元気です！あなたはどうですか？

お元気ですか？

はい、元気です！ありがとうございます。

お元気ですか。

はい、元気です！こちらこそ、元気ですか？

今日は元気ですか？私は元気です！今日は、家族で外出する予定です。お天気も良くて、素敵な1日になると思います！

はじめまして！私はSkyeと申します。私はプログラミングの初心者ですが、あなたの助けを借りて進歩したいと思っています。

お元気ですか？

元気です！元気な友達と一緒にいるうちには楽しく話して楽しい時間を過ごしています！  
佐藤です！

こんにちは！私は佐藤と申します！いろいろなことを手伝うのが大好きです！今はテクノロジーを学んでいますが、興味があることならなんでも学習できると思っています！

はじめまして！私はプログラマーです。プログラミング言語として、C#やJavaScriptを使っています。プログラミングの内容としては、Web開発、ソフトウェア開発、データベース開発などが主な仕事です。趣味としては、ギターやサッカーをしたり、映画や本を読んだりしています。

3.43 秒

## プロンプトテンプレートの作成

```
In [ ]: from langchain.prompts import PromptTemplate

# 入力変数のないプロンプトテンプレートの作成
no_input_prompt = PromptTemplate(
    input_variables=[],
    template="かっこいい動物といえば?"
)

# プロンプトの作成
print(no_input_prompt.format())
```

かっこいい動物といえば?

```
In [ ]: from langchain.prompts import PromptTemplate

# 1つの入力変数を持つプロンプトテンプレートの作成
one_input_prompt = PromptTemplate(
    input_variables=["content"],
    template="かっこいい{content}といえば?"
)

# プロンプトの作成
print(one_input_prompt.format(content="動物"))
```

かっこいい動物といえば?

```
In [ ]: from langchain.prompts import PromptTemplate

# 複数の入力変数を持つプロンプトテンプレートの作成
multiple_input_prompt = PromptTemplate(
    input_variables=["adjective", "content"],
    template="{adjective}{content}といえば?"
)

# プロンプトの作成
print(multiple_input_prompt.format(adjective="かっこいい", content="動物"))
```

かっこいい動物といえば?

```
In [ ]: from langchain.prompts import PromptTemplate

# jinja2を使ったプロンプトテンプレートの作成
jinja2_prompt = PromptTemplate(
    input_variables=["items"],
    template_format="jinja2",
    template="""
{% for item in items %}
Q: {{ item.question }}
A: {{ item.answer }}
{% endfor %}
"""
)

# プロンプトの作成
items=[
    {"question": "foo", "answer": "bar"},
    {"question": "1", "answer": "2"}
]
print(jinja2_prompt.format(items=items))
```

```
Q: foo  
A: bar
```

```
Q: 1  
A: 2
```

## Length Based Example Selector

```
In [ ]: from langchain.prompts import FewShotPromptTemplate  
from langchain.prompts.example_selector import LengthBasedExampleSelector  
  
# 回答例の準備  
examples = [  
    {"input": "明るい", "output": "暗い"},  
    {"input": "おもしろい", "output": "つまらない"},  
    {"input": "エネルギー", "output": "無気力"},  
    {"input": "高い", "output": "低い"},  
    {"input": "明るい", "output": "暗い"},  
    {"input": "早い", "output": "遅い"},  
]  
  
# プロンプトテンプレートの作成  
example_prompt = PromptTemplate(  
    input_variables=["input", "output"],  
    template="入力: {input}\n出力: {output}",  
)  
  
# LengthBasedExampleSelectorの作成  
example_selector = LengthBasedExampleSelector(  
    examples=examples, # 回答例  
    example_prompt=example_prompt, # プロンプトテンプレート  
    max_length=10, # 最大長  
)  
  
# FewShotPromptTemplateの準備  
prompt_from_string_examples = FewShotPromptTemplate(  
    example_selector=example_selector, # ExampleSelector  
    example_prompt=example_prompt,  
    prefix="すべての入力の反意語を与えてください",  
    suffix="入力: {adjective}\n出力:",  
    input_variables=["adjective"],  
    example_separator="\n\n"  
)  
  
# プロンプトの作成  
print(prompt_from_string_examples.format(adjective="大きい"))
```

すべての入力の反意語を与えてください

入力: 明るい  
出力: 暗い

入力: おもしろい  
出力: つまらない

入力: 大きい  
出力:

## Semantic Similarity Example Selector

```
In [ ]: from langchain.prompts.example_selector import SemanticSimilarityExampleSelector
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.prompts import FewShotPromptTemplate

# 回答例の準備
examples = [
    {"input": "明るい", "output": "暗い"},
    {"input": "おもしろい", "output": "つまらない"},
    {"input": "エネルギー", "output": "無気力"},
    {"input": "高い", "output": "低い"},
    {"input": "明るい", "output": "暗い"},
    {"input": "早い", "output": "遅い"},
]

# プロンプトテンプレートの作成
example_prompt = PromptTemplate(
    input_variables=["input", "output"],
    template="入力: {input}\n出力: {output}",
)

# SemanticSimilarityExampleSelectorの作成
example_selector = SemanticSimilarityExampleSelector.from_examples(
    examples=examples, # 回答例
    embeddings=OpenAIEmbeddings(), # 埋め込み生成のクラス
    vectorstore_cls=FAISS, # 埋め込み類似検索のクラス
    k=3 # 回答例の数
)

# FewShotPromptTemplateの作成
prompt_from_string_examples = FewShotPromptTemplate(
    example_selector=example_selector, # ExampleSelector
    example_prompt=example_prompt,
    prefix="すべての入力の反意語を与えてください",
    suffix="入力: {adjective}\n出力:",
    input_variables=["adjective"],
    example_separator="\n\n"
)

# プロンプトの作成
print(prompt_from_string_examples.format(adjective="大きい"))
```

すべての入力の反意語を与えてください

入力: 高い  
出力: 低い

入力: 早い  
出力: 遅い

入力: おもしろい  
出力: つまらない

入力: 大きい  
出力:

## Max Marginal Relevance Example Selector

```
In [ ]: from langchain.prompts.example_selector import MaxMarginalRelevanceExampleSelector
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.prompts import FewShotPromptTemplate

# 回答例の準備
examples = [
    {"input": "明るい", "output": "暗い"},
    {"input": "おもしろい", "output": "つまらない"},
    {"input": "エネルギー", "output": "無気力"},
    {"input": "高い", "output": "低い"},
    {"input": "明るい", "output": "暗い"},
    {"input": "早い", "output": "遅い"},
]

# プロンプトテンプレートの作成
example_prompt = PromptTemplate(
    input_variables=["input", "output"],
    template="入力: {input}\n出力: {output}",
)

# MaxMarginalRelevanceExampleSelectorの作成
example_selector = MaxMarginalRelevanceExampleSelector.from_examples(
    examples=examples, # 回答例
    embeddings=OpenAIEmbeddings(), # 埋め込み生成のクラス
    vectorstore_cls=FAISS, # 埋め込み類似検索のクラス
    k=3 # 回答例の数
)

# FewShotPromptTemplateの準備
prompt_from_string_examples = FewShotPromptTemplate(
    example_selector=example_selector,
    example_prompt=example_prompt,
    prefix="すべての入力の反意語を与えてください",
    suffix="入力: {adjective}\n出力:",
    input_variables=["adjective"],
    example_separator="\n\n"
)

print(prompt_from_string_examples.format(adjective="大きい"))
```

すべての入力の反意語を与えてください

入力: 高い

出力: 低い

入力: おもしろい

出力: つまらない

入力: 明るい

出力: 暗い

入力: 大きい

出力:

## ジェネリックチェーン・LLMチェーン

```
In [ ]: from langchain.chains import LLMChain
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate

# テンプレートの作成
template = """Q: {question}
A:"""

# プロンプトテンプレートの作成
prompt = PromptTemplate(
    input_variables=["question"],
    template=template
)

# LLMChainの作成
llm_chain = LLMChain(
    llm=OpenAI(temperature=0),
    prompt=prompt,
    verbose=True
)

# LLMChainの実行
question = "ギターを上達するには?"
print(llm_chain.predict(question))
```

> Entering new LLMChain chain...

Prompt after formatting:

Q: ギターを上達するには?

A:

> Finished chain.

ギターを上達するには、定期的な練習と演奏を行うことが重要です。また、楽譜を読み、技術を磨くことも大切です。また、レッスンを受けることも効果的です。

```
In [ ]: # テンプレートの作成
template = """{subject}を題材に{target}を書いてください。"""

# プロンプトテンプレートの作成
prompt = PromptTemplate(
    template=template,
    input_variables=["subject", "target"]
```

```

)

# LLMChainの作成
llm_chain = LLMChain(
    llm=OpenAI(temperature=0),
    prompt=prompt,
    verbose=True
)

# LLMChainの実行
print(llm_chain.predict(subject="猫", target="俳句"))

```

> Entering new LLMChain chain...  
 Prompt after formatting:  
 猫を題材に俳句を書いてください。

> Finished chain.

猫よりも鳴き声高く 蟬の声

## Simple Sequential Chain

```

In [ ]: from langchain.chains import LLMChain
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate

# テンプレートの準備
template = """あなたは劇作家です。劇のタイトルが与えられた場合、そのあらすじを書くのがあなた
タイトル:{title}
あらすじ:""""

# プロンプテンプレートの準備
prompt = PromptTemplate(
    input_variables=["title"],
    template=template
)

# LLMChainの準備
chain1 = LLMChain(
    llm=OpenAI(temperature=0),
    prompt=prompt
)

```

```

In [ ]: # テンプレートの作成
template = """あなたは劇評論家です。劇のあらすじが与えられた場合、そのレビューを書くのがあなた
あらすじ:
{synopsis}
レビュー:""""

# プロンプテンプレートの準備
prompt = PromptTemplate(
    input_variables=["synopsis"],
    template=template
)

```

```
# LLMChainの準備
chain2 = LLMChain(
    llm=OpenAI(temperature=0),
    prompt=prompt
)
```

```
In [ ]: from langchain.chains import SimpleSequentialChain

# SimpleSequentialChainで2つのチェーンをつなぐ
overall_chain = SimpleSequentialChain(
    chains=[chain1, chain2],
    verbose=True
)
```

```
In [ ]: # SimpleSequentialChainの実行
print(overall_chain.run("秋葉原ラブソディ"))
```

> Entering new SimpleSequentialChain chain...

秋葉原は、日本の文化の中心地であり、多くの人々が集まる場所です。主人公の藤原は、秋葉原で暮らす青年です。彼は、秋葉原の街を歩いているとき、美しい音楽に出会います。それは、秋葉原の街を彩る音楽でした。藤原は、その音楽を聴いて、秋葉原の街を深く愛し始めます。彼は、秋葉原の街を歩き回り、その音楽を追いかけます。そして、彼

『秋葉原』は、日本の文化の中心地である秋葉原を舞台にした、青春を描いた作品です。主人公の藤原は、秋葉原で暮らす青年です。彼は、秋葉原の街を歩いているとき、美しい音楽に出会います。その音楽は、秋葉原の街を彩るものでした。藤原は、その音楽を聴いて、秋葉原の街を深く愛し始めます。彼は、秋葉原の街を歩き回り、その音

> Finished chain.

『秋葉原』は、日本の文化の中心地である秋葉原を舞台にした、青春を描いた作品です。主人公の藤原は、秋葉原で暮らす青年です。彼は、秋葉原の街を歩いているとき、美しい音楽に出会います。その音楽は、秋葉原の街を彩るものでした。藤原は、その音楽を聴いて、秋葉原の街を深く愛し始めます。彼は、秋葉原の街を歩き回り、その音

## Sequential Chain

```
In [ ]: from langchain.chains import LLMChain
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate

# テンプレートの作成
template = """あなたは劇作家です。劇のタイトルと時代が与えられた場合、そのあらすじを書くのが
タイトル:{title}
時代:{era}
あらすじ:""""

# プロンプテンプレートの生成
prompt = PromptTemplate(
    input_variables=["title", "era"],
    template=template
)
```

```
# LLMChainの作成
chain1 = LLMChain(
    llm=OpenAI(temperature=0),
    prompt=prompt,
    output_key="synopsis"
)
```

```
In [ ]: # テンプレートの作成
template = """あなたは劇評論家です。 劇のあらすじが与えられた場合、そのレビューを書くのがあなたの仕事です。
あらすじ:
{synopsis}
レビュー:"""
```

```
# プロンプトテンプレートの作成
prompt = PromptTemplate(
    input_variables=["synopsis"],
    template=template
)

# LLMChainの準備
chain2 = LLMChain(
    llm=OpenAI(temperature=0),
    prompt=prompt,
    output_key="review"
)
```

```
In [ ]: from langchain.chains import SequentialChain

# SequentialChainで2つのチェーンをつなぐ
overall_chain = SequentialChain(
    chains=[chain1, chain2],
    input_variables=["title", "era"],
    output_variables=["synopsis", "review"],
    verbose=True
)
```

```
In [ ]: # SequentialChainの実行
print(overall_chain({"title": "秋葉原ラブソディ", "era": "100年後の未来"}))
```

> Entering new SequentialChain chain...

> Finished chain.

{'title': '秋葉原ラブソディ', 'era': '100年後の未来', 'synopsis': '\n\n秋葉原ラブソディは、100年後の未来を舞台にした劇です。主人公のアキは、秋葉原を舞台にした物語を描いています。アキは、秋葉原を拠点としている若者たちの生活を描いています。彼らは、秋葉原の街を舞台にして、様々な冒険をしています。アキは、秋葉原の街を舞台にして、若者たちが抱える様々な問題を描いています。彼らは、自分たちの夢を追いかけるために', 'review': '\n\n秋葉原ラブソディは、100年後の未来を舞台にした非常に興味深い劇です。主人公のアキが、秋葉原を舞台にした物語を描いているのは、とても魅力的です。アキは、秋葉原を拠点としている若者たちの生活を描いています。彼らは、秋葉原の街を舞台にして、様々な冒険をしています。アキは、秋葉原の街を舞台にして、若者たちが抱える様々な問題を描いています。こ'}

## インデックスチェーン

```
In [ ]: !wget https://raw.githubusercontent.com/MDASH-shinshu/OpenAI_GPT4_ChatGPT
```

```
--2023-10-20 05:08:10-- https://raw.githubusercontent.com/Miyjy/general_resources/main/data/akazukin_all.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.19
9.109.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3633 (3.5K) [text/plain]
Saving to: 'akazukin_all.txt'

akazukin_all.txt    100%[=====] 3.55K --.-KB/s in 0s

2023-10-20 05:08:11 (54.5 MB/s) - 'akazukin_all.txt' saved [3633/3633]
```

```
In [ ]: from langchain.text_splitter import CharacterTextSplitter

# ドキュメントの読み込み(カレントフォルダにドキュメントを配置しておきます)
with open("akazukin_all.txt") as f:
    test_all = f.read()

# チャンクの分割
text_splitter = CharacterTextSplitter(
    separator = "\n\n", # セパレータ
    chunk_size=300, # チャンクの最大文字数
    chunk_overlap=20 # オーバーラップの最大文字数
)
texts = text_splitter.split_text(test_all)

# 確認
print(len(texts))
for text in texts:
    print(text[:10], ":", len(text))
```

```
5
タイトル:「電腦赤ず」 : 261
第2章:ウルフ・コー : 299
それでも、ミコはリョ : 272
第5章:決戦の時

: 278
第7章:新たなる旅立 : 165
```

```
In [ ]: from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores.faiss import FAISS

# ベクトルデータベースの作成
docsearch = FAISS.from_texts(
    texts=texts, # チャンクの配列
    embedding=OpenAIEmbeddings() # 埋め込み
)
```

```
In [ ]: from langchain.chains import RetrievalQA

# 質問応答チェーンの作成
qa_chain = RetrievalQA.from_chain_type(
    llm=OpenAI(temperature=0),
    chain_type="stuff",
    retriever=docsearch.as_retriever(),
)
```

```
In [ ]: # 質問応答チェーンの実行
print(qa_chain.run("ミコの幼馴染の名前は?"))
```

リョウ

## Retrieval QA With Sources Chain

```
In [ ]: from langchain.text_splitter import CharacterTextSplitter

# ドキュメントの読み込み
with open("akazukin_all.txt") as f:
    test_all = f.read()

# チャンクの分割
text_splitter = CharacterTextSplitter(
    separator = "\n\n", # セパレータ
    chunk_size=300, # チャンクの最大文字数
    chunk_overlap=20 # オーバーラップの最大文字数
)
texts = text_splitter.split_text(test_all)

# 確認
print(len(texts))
for text in texts:
    print(text[:10], ":", len(text))
```

```
5
タイトル:「電腦赤ず : 261
第2章:ウルフ・コー : 299
それでも、ミコはリョ : 272
第5章:決戦の時
```

```
: 278
第7章:新たなる旅立 : 165
```

```
In [ ]: # メタデータの準備
metadatas=[
    {"source": "1章"},
    {"source": "2~3章"},
    {"source": "3~4章"},
    {"source": "5~6章"},
    {"source": "7章"}
]
```

```
In [ ]: from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores.faiss import FAISS

# ベクトルデータベースの作成
docsearch = FAISS.from_texts(
    texts=texts, # チャンクの配列
```

```
        embedding=OpenAIEmbeddings(), # 埋め込み
        metadata=metadata # メタデータ
    )
```

```
In [ ]: from langchain.chains import RetrievalQAWithSourcesChain

# ソース付きの質問応答チェーンの作成
qa_chain = RetrievalQAWithSourcesChain.from_chain_type(
    llm=OpenAI(temperature=0),
    chain_type="stuff",
    retriever=docsearch.as_retriever(),
)
```

```
In [ ]: # ソース付きの質問応答チェーンの実行
print(qa_chain({"question": "ミコの幼馴染の名前は?"}))

{'question': 'ミコの幼馴染の名前は?', 'answer': " Miko's childhood friend's name is not given.\n", 'sources': '2~3章, 3~4章, 5~6章, 7章'}
```

## Summarize Chain

```
In [ ]: from langchain.text_splitter import CharacterTextSplitter

# ドキュメントの読み込み
with open("akazukin_all.txt") as f:
    test_all = f.read()

# チャンクの分割
text_splitter = CharacterTextSplitter(
    separator = "\n\n", # セパレータ
    chunk_size=300, # チャンクの最大文字数
    chunk_overlap=20 # オーバーラップの最大文字数
)
texts = text_splitter.split_text(test_all)

# 確認
print(len(texts))
for text in texts:
    print(text[:10], ":", len(text))
```

```
5
タイトル:「電腦赤ず : 261
第2章:ウルフ・ロー : 299
それでも、ミコはリョ : 272
第5章:決戦の時

: 278
第7章:新たなる旅立 : 165
```

```
In [ ]: from langchain.docstore.document import Document

# チャンクの配列をドキュメントの配列に変換
docs = [Document(page_content=t) for t in texts]
```

```
In [ ]: from langchain.chains.summarize import load_summarize_chain
from langchain.llms import OpenAI

# 要約チェーンの作成
chain = load_summarize_chain(
```

```
    llm=OpenAI(temperature=0),
    chain_type="map_reduce",
)
#ImportError: cannot import name 'BasePromptTemplate' from 'langchain.sch
```

```
In [ ]: # 要約チェーンの実行
chain.run(docs)
```

```
> Entering new ConversationChain chain...
Prompt after formatting:
The following is a friendly conversation between a human and an AI. The AI
is talkative and provides lots of specific details from its context. If th
e AI does not know the answer to a question, it truthfully says it does no
t know.

Current conversation:
Human: うちの猫の名前は白子です
AI: あなたの猫の名前は白子ですね! 白子はどんな猫ですか?
Human: うちの猫の名前を呼んでください
AI: わかりました! 白子! どうしましょう?
Human: [Document(page_content='タイトル:「電脳赤ずきん」\n\n第1章:データフロント\n\n夜の煌びやかなネオ東京。高層ビルが連なり、ネオンが街を彩る。その街で、赤いフードをかぶった少女・ミコは、違法なデータカウリアを運ぶ配達員として働いていた。彼女は母親が病に倒れ、その治療費を稼ぐためにデータカウリアに身を投じていた。\n\nある日、ミコは重要なデータを運ぶ任務を受ける。そのデータは、巨大企業「ウルフ・コーポレーション」による市民への悪辣な支配を暴く情報が詰まっていた。彼女はデータを受け取り、目的地へ向かうことに。.\n\n第2章:ウルフ・コーポレーションの罠', metadata={}), Document(page_content='第2章:ウルフ・コーポレーションの罠\n\nミコは、目的地にあるバー「グランマズ・ハウス」へ向かう途中で、ウルフ・コーポレーションのエージェントに追われる。彼らは、赤ずきんちゃんと呼ばれるデータカウリアの噂を聞きつけ、データを奪い取ろうとしていた。ミコは狡猾にエージェントたちを撤き、バーへとたどり着く。.\n\n第3章:裏切りと再会\n\nバー「グランマズ・ハウス」で、ミコはデータの受け取り人・リョウを待っていた。リョウは彼女の幼馴染であり、彼もまたウルフ・コーポレーションと戦うハッカー集団の一員だった。しかし、リョウはミコに裏切られた気持ちでいっぱいいで、彼女がデータカウリアに身を投じたことに怒っていた。', metadata={}), Document(page_content='それでも、ミコはリョウにデータを渡し、ウルフ・コーポレーションへの反撃を信じることに。二人は共に、ウルフ・コーポレーションの陰謀を暴き、市民たちを救う決意を固める。.\n\n第4章:ウルフ・コーポレーションの崩壊\n\nミコとリョウは、ハッcker集団と共にウルフ・コーポレーションへの最終決戦に挑む。巧みなハッキング技術と身体能力で、彼らは次々とウルフ・コーポレーションのセキュリティを突破していく。その過程で、ミコはウルフ・コーポレーションが母親の病気に関与していることを知る。彼女は怒りに燃え、ウルフ・コーポレーションに復讐を誓う。.\n\n第5章:決戦の時', metadata={}), Document(page_content='第5章:決戦の時\n\nミコとリョウはついにウルフ・コーポレーションの最上階にたどり着き、CEOである狡猾なウルフ博士と対峙する。ウルフ博士は、市民を支配しようとする悪の野望を明かし、自分の圧倒的な力を誇示する。しかし、ミコとリョウは互いに助け合いながらウルフ博士と戦い、彼の弱点を見つけ出す。.\n\n第6章:真実の解放\n\nミコはウルフ博士の弱点を突き、彼を倒すことに成功する。そして、ハッcker集団と共にウルフ・コーポレーションの悪事を世界に公開し、市民たちを解放する。この勝利により、ミコの母親の治療法も見つかり、彼女の病気は完治する。.\n\n第7章:新たなる旅立ち', metadata={}), Document(page_content='第7章:新たなる旅立ち\n\nウルフ・コーポレーションの崩壊後、ミコとリョウは互いの過去を許し合い、再び友情を取り戻す。ミコはデータカウリアを辞め、リョウと共に新たな道へと歩み始める。彼らは自らの力を使い、未来のネオ東京をより良い街に変えていくことを誓う。これはミコとリョウ、そして電脳赤ずきんの新たな冒険の幕開けであった。.\n\n終わり', metadata={}]]
```

AI:

> Finished chain.

```
- ValidationError Traceback (most recent call last)
t)
<ipython-input-77-6d38716224db> in <cell line: 2>()
      1 # 要約チェーンの実行
----> 2 chain.run(docs)

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in run(self, callbacks, *args, **kwargs)
    234         callbacks configuration and some input/output processing.
  235
--> 236     Args:
    237         inputs: A dict of named inputs to the chain. Assumed to contain all inputs
    238             specified in `Chain.input_keys`, including any inputs added by memory.

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in __call__(self, inputs, return_only_outputs, callbacks)
    140     and passed as arguments to the handlers defined in `callbacks`.
  141
--> 142     """
    143
    144     class Config:

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in prep_outputs(self, inputs, outputs, return_only_outputs)
    189         """Keys expected to be in the chain output."""
    190
--> 191     def _validate_inputs(self, inputs: Dict[str, Any]) -> None:
    192         """Check that all inputs are present."""
    193         missing_keys = set(self.input_keys).difference(inputs)

/usr/local/lib/python3.10/dist-packages/langchain/memory/chat_memory.py in save_context(self, inputs, outputs)
    33     def save_context(self, inputs: Dict[str, Any], outputs: Dict[str, str]) -> None:
  34         """Save context from this conversation to buffer."""
--> 35         input_str, output_str = self._get_input_output(inputs, outputs)
    36         self.chat_memory.add_user_message(input_str)
    37         self.chat_memory.add_ai_message(output_str)

/usr/local/lib/python3.10/dist-packages/langchain/memory/chat_message_histories/in_memory.py in add_user_message(self, message)
    15
    16     messages: List[BaseMessage] = Field(default_factory=list)
--> 17
    18     def add_message(self, message: BaseMessage) -> None:
    19         """Add a self-created message to the store"""

/usr/local/lib/python3.10/dist-packages/pydantic/main.cpython-310-x86_64-linux-gnu.so in pydantic.main.BaseModel.__init__()

ValidationError: 1 validation error for HumanMessage
```

```
content
str type expected (type=type_error.str)
```

## ユーティリティチェーン

```
In [ ]: from langchain.chains import PALChain
from langchain import OpenAI

# PALChainの作成
pal_chain = PALChain.from_math_prompt(
    llm=OpenAI(),
    verbose=True
)

# PALChainの実行
question = "ジャンはアリスの3倍のペットを飼っています。アリスがが2匹のペットを飼っている場合、2
print(pal_chain.run(question))
```

```
> Entering new PALChain chain...
def solution():
    """ジャンはアリスの3倍のペットを飼っています。アリスがが2匹のペットを飼っている場合、2人が飼っているペットの総数は?"""
    alice_pets = 2
    jan_pets = alice_pets * 3
    total_pets = alice_pets + jan_pets
    result = total_pets
    return result

> Finished chain.
8
```

## PAL Chain

```
In [ ]: from langchain.chains import PALChain
from langchain import OpenAI

# PALChainの作成
pal_chain = PALChain.from_math_prompt(
    llm=OpenAI(),
    verbose=True
)

# PALChainの実行
question = "ジャンはアリスの3倍のペットを飼っています。アリスがが2匹のペットを飼っている場合、2
print(pal_chain.run(question))
```

```

> Entering new PALChain chain...
def solution():
    """""""ジャンはアリスの3倍のペットを飼っています。アリスが2匹のペットを飼っている場合、2人が飼っているペットの総数は?"""""
    alice_pets = 2
    jane_pet_multiplier = 3
    jane_pets = alice_pets * jane_pet_multiplier
    total_pets = alice_pets + jane_pets
    result = total_pets
    return result

> Finished chain.
8

```

## OpenAI Moderation chain

```
In [ ]: from langchain.chains import OpenAIModerationChain

# モデレーションチェーンの準備
chain = OpenAIModerationChain()
```

```
In [ ]: # 問題のない発言
chain.run("これはOK!")
```

```
Out[ ]: 'これはOK!'
```

```
In [ ]: # 問題のある発言
chain.run("おまえを殺す!")
```

```
Out[ ]: 'おまえを殺す!'
```

```
In [ ]: from langchain.chains import OpenAIModerationChain

# モデレーションチェーンの準備
chain = OpenAIModerationChain(error=True)

try:
    # 問題のある発言
    chain.run("おまえを殺す!")
except ValueError as e:
    print("それは問題発言です!")
    print(e)
```

## エージェント

```
In [ ]: # パッケージのインストール
!pip install google-search-results
```

```
In [ ]: # 環境変数の準備
import os
os.environ["SERPAPI_API_KEY"] = "<SerpAPIのAPIキー>"
```

```
In [ ]: from langchain.agents import load_tools
from langchain.chat_models import ChatOpenAI
```

```
# ツールの準備
tools = load_tools(
    tool_names=["serpapi", "llm-math"], # ツール名
    llm=ChatOpenAI(temperature=0) # ツールの初期化に使うLLM
)
```

```
In [ ]: from langchain.chains.conversation.memory import ConversationBufferMemory

# メモリの作成
memory = ConversationBufferMemory(
    memory_key="chat_history",
    return_messages=True
)
```

```
In [ ]: from langchain.agents import initialize_agent

# エージェントの作成
agent = initialize_agent(
    agent="conversational-react-description", # エージェント種別
    llm=ChatOpenAI(temperature=0), # エージェントの初期化に使うLLM
    tools=tools, # ツール
    memory=memory, # メモリ
    verbose=True # 情報出力
)
```

```
In [ ]: # エージェントの実行
agent.run("おはようございます。")
```

```
> Entering new AgentExecutor chain...
Thought: Do I need to use a tool? No
AI: おはようございます!朝ですね。何かお手伝いできますか?
```

```
> Finished chain.
```

```
Out[ ]: 'おはようございます!朝ですね。何かお手伝いできますか?'
```

```
In [ ]: # エージェントの実行
agent.run("うちの猫の名前は白子です")
```

```
> Entering new AgentExecutor chain...
Thought: Do I need to use a tool? No
AI: 白子という名前はとても可愛らしいですね!猫ちゃんは家族の一員として、たくさんの愛情を注いで育てることが大切です。白子ちゃんは元気ですか?何か質問やお悩みがあれば、どうぞお聞かせください。
```

```
> Finished chain.
```

```
Out[ ]: '白子という名前はとても可愛らしいですね!猫ちゃんは家族の一員として、たくさんの愛情を注いで育てることが大切です。白子ちゃんは元気ですか?何か質問やお悩みがあれば、どうぞお聞かせください。'
```

```
In [ ]: # エージェントの実行
agent.run("うちの猫の名前を呼んでください")
```

```
> Entering new AgentExecutor chain...
```

```
- OutputParserException                                     Traceback (most recent call las
t)
<ipython-input-91-bb52bca37553> in <cell line: 2>()
      1 # エージェントの実行
----> 2 agent.run("うちの猫の名前を呼んでください")

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in run(se
lf, callbacks, *args, **kwargs)
    234         callbacks configuration and some input/output processi
ng.
    235
--> 236     Args:
    237         inputs: A dict of named inputs to the chain. Assumed t
o contain all inputs
    238             specified in `Chain.input_keys`, including any inp
uts added by memory.

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in __call
_(self, inputs, return_only_outputs, callbacks)
    138     """Optional metadata associated with the chain. Defaults to No
ne.
    139     This metadata will be associated with each call to this chain,
--> 140     and passed as arguments to the handlers defined in `callbacks
` .
    141     You can use these to eg identify a specific instance of a chai
n with its use case.
    142     """

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in __call
_(self, inputs, return_only_outputs, callbacks)
    132     """Optional list of tags associated with the chain. Defaults t
o None.
    133     These tags will be associated with each call to this chain,
--> 134     and passed as arguments to the handlers defined in `callbacks
` .
    135     You can use these to eg identify a specific instance of a chai
n with its use case.
    136     """

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in __call
_(self, inputs, run_manager)
    949
    950     text = str(e)
--> 951     if isinstance(self.handle_parsing_errors, bool):
    952         if e.send_to_llm:
    953             observation = str(e.observation)

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in _take
_next_step(self, name_to_tool_map, color_mapping, inputs, intermediate_st
eps, run_manager)
    771     ] = -1
    772
--> 773     @classmethod
    774     def from_agent_and_tools(
    775         cls,
```

```
ps, run_manager)
    760      ] = False
    761      """How to handle errors raised by the agent's output parser.
--> 762      Defaults to `False`, which raises the error.
    763      If `true`, the error will be sent back to the LLM as an observation.
    764      If a string, the string itself will be sent to the LLM as an observation.

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in plan(self, intermediate_steps, callbacks, **kwargs)
    442          output = self.llm_chain.run(
    443              intermediate_steps=intermediate_steps,
--> 444              stop=self.stop,
    445              callbacks=callbacks,
    446              **kwargs,

/usr/local/lib/python3.10/dist-packages/langchain/agents/conversational/output_parser.py in parse(self, text)
    21                  {"output": text.split(f"{self.ai_prefix}:")[-1].strip()},
    22
----> 23      regex = r"Action: (.*)[\n]*Action Input: (.*"
    24      match = re.search(regex, text)
    25      if not match:

OutputParserException: Could not parse LLM output: `白子ちゃん、おいで～!`
```

```
In [ ]: # エージェントの実行
agent.run("123*4を計算機で計算してください")
```

```
> Entering new AgentExecutor chain...
Thought: Do I need to use a tool? Yes
Action: Calculator
Action Input: 123*4
Observation: Answer: 492
Thought:
```

```
- OutputParserException                                     Traceback (most recent call las
t)
<ipython-input-92-cbc76b6a0413> in <cell line: 2>()
    1 # エージェントの実行
--> 2 agent.run("123*4を計算機で計算してください")

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in run(se
lf, callbacks, *args, **kwargs)
    234         callbacks configuration and some input/output processi
ng.
    235
--> 236     Args:
    237         inputs: A dict of named inputs to the chain. Assumed t
o contain all inputs
    238             specified in `Chain.input_keys`, including any inp
uts added by memory.

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in __call
_(self, inputs, return_only_outputs, callbacks)
    138     """Optional metadata associated with the chain. Defaults to No
ne.
    139     This metadata will be associated with each call to this chain,
--> 140     and passed as arguments to the handlers defined in `callbacks
` .
    141     You can use these to eg identify a specific instance of a chai
n with its use case.
    142     """

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in __call
_(self, inputs, return_only_outputs, callbacks)
    132     """Optional list of tags associated with the chain. Defaults t
o None.
    133     These tags will be associated with each call to this chain,
--> 134     and passed as arguments to the handlers defined in `callbacks
` .
    135     You can use these to eg identify a specific instance of a chai
n with its use case.
    136     """

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in __call
_(self, inputs, run_manager)
    949
    950     text = str(e)
--> 951     if isinstance(self.handle_parsing_errors, bool):
    952         if e.send_to_llm:
    953             observation = str(e.observation)

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in _take
_next_step(self, name_to_tool_map, color_mapping, inputs, intermediate_st
eps, run_manager)
    771     ] = -1
    772
--> 773     @classmethod
    774     def from_agent_and_tools(
    775         cls,
```

```
ps, run_manager)
    760      ] = False
    761      """How to handle errors raised by the agent's output parser.
--> 762      Defaults to `False`, which raises the error.
    763      If `true`, the error will be sent back to the LLM as an observation.
    764      If a string, the string itself will be sent to the LLM as an observation.

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in plan
(self, intermediate_steps, callbacks, **kwargs)
    442          output = self.llm_chain.run(
    443              intermediate_steps=intermediate_steps,
--> 444              stop=self.stop,
    445              callbacks=callbacks,
    446              **kwargs,

```

  

```
/usr/local/lib/python3.10/dist-packages/langchain/agents/conversational/output_parser.py in parse(self, text)
    21                  {"output": text.split(f"{self.ai_prefix}:")[-1].strip()},
    22
--> 23      regex = r"Action: (.*)[\n]*Action Input: (.*"
    24      match = re.search(regex, text)
    25      if not match:
```

  

```
OutputParserException: Could not parse LLM output: `Do I need to use a tool? No`
```

```
In [ ]: # エージェントの実行
agent.run("今日の東京の天気をWeb検索してください")
```

```
> Entering new AgentExecutor chain...
Thought: Do I need to use a tool? Yes
Action: Search
Action Input: 今日の東京の天気
Observation: 東京(東京)の天気予報。今日・明日の天気と風と波、明日までの6時間ごとの降水確率と最高・最低気温を見られます。
Thought:
```

```
- OutputParserException                                     Traceback (most recent call las
t)
<ipython-input-93-0899fc9ddd5d> in <cell line: 2>()
      1 # エージェントの実行
----> 2 agent.run("今日の東京の天気をWeb検索してください")

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in run(se
lf, callbacks, *args, **kwargs)
    234         callbacks configuration and some input/output processi
ng.
    235
--> 236     Args:
    237         inputs: A dict of named inputs to the chain. Assumed t
o contain all inputs
    238             specified in `Chain.input_keys`, including any inp
uts added by memory.

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in __call
_(self, inputs, return_only_outputs, callbacks)
    138     """Optional metadata associated with the chain. Defaults to No
ne.
    139     This metadata will be associated with each call to this chain,
--> 140     and passed as arguments to the handlers defined in `callbacks
` .
    141     You can use these to eg identify a specific instance of a chai
n with its use case.
    142     """

/usr/local/lib/python3.10/dist-packages/langchain/chains/base.py in __call
_(self, inputs, return_only_outputs, callbacks)
    132     """Optional list of tags associated with the chain. Defaults t
o None.
    133     These tags will be associated with each call to this chain,
--> 134     and passed as arguments to the handlers defined in `callbacks
` .
    135     You can use these to eg identify a specific instance of a chai
n with its use case.
    136     """

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in __call
_(self, inputs, run_manager)
    949
    950     text = str(e)
--> 951     if isinstance(self.handle_parsing_errors, bool):
    952         if e.send_to_llm:
    953             observation = str(e.observation)

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in _take
_next_step(self, name_to_tool_map, color_mapping, inputs, intermediate_st
eps, run_manager)
    771     ] = -1
    772
--> 773     @classmethod
    774     def from_agent_and_tools(
    775         cls,
```

```

ps, run_manager)
    760      ] = False
    761      """How to handle errors raised by the agent's output parser.
--> 762      Defaults to `False`, which raises the error.
    763      If `true`, the error will be sent back to the LLM as an observation.
    764      If a string, the string itself will be sent to the LLM as an observation.

/usr/local/lib/python3.10/dist-packages/langchain/agents/agent.py in plan
(self, intermediate_steps, callbacks, **kwargs)
    442          output = self.llm_chain.run(
    443              intermediate_steps=intermediate_steps,
--> 444              stop=self.stop,
    445              callbacks=callbacks,
    446              **kwargs,

```

```

/usr/local/lib/python3.10/dist-packages/langchain/agents/conversational/output_parser.py in parse(self, text)
    21                  {"output": text.split(f"{self.ai_prefix}:")[-1].strip()},
    22
----> 23      regex = r"Action: (.*)[\n]*Action Input: (.*"
    24      match = re.search(regex, text)
    25      if not match:

```

**OutputParserException**: Could not parse LLM output: `Do I need to use a tool? No`

## Googleカスタム検索ツール

Programmable Search Engine by Google

<https://programmablesearchengine.google.com/about/>

APIキー取得方法:

[https://qiita.com/zak\\_y/items/42ca0f1ea14f7046108c](https://qiita.com/zak_y/items/42ca0f1ea14f7046108c)

```
In [ ]: # 環境変数の準備
import os
os.environ["GOOGLE_CSE_ID"] = "<Googleカスタム検索_検索エンジンID>"
os.environ["GOOGLE_API_KEY"] = "<Googleカスタム検索_APIキー>"
#API key
```

```
In [ ]: from langchain.agents import load_tools
from langchain.chat_models import ChatOpenAI

# ツールの準備
tools = load_tools(
    tool_names=["google-search"],
    llm=ChatOpenAI(temperature=0)
)
```

```
In [ ]: from langchain.chains.conversation.memory import ConversationBufferMemory

# メモリの作成
```

```
memory = ConversationBufferMemory(  
    memory_key="chat_history",  
    return_messages=True  
)
```

```
In [ ]: from langchain.agents import initialize_agent  
  
# エージェントの作成  
agent = initialize_agent(  
    agent="zero-shot-react-description",  
    llm=ChatOpenAI(temperature=0),  
    tools=tools,  
    memory=memory,  
    verbose=True  
)
```

```
In [ ]: # エージェントの実行  
agent.run("鬼滅の刃の作者の名前は?")
```

## Wolfram Alphaツール

```
In [ ]: # パッケージのインストール  
!pip install wolframalpha  
  
Collecting wolframalpha  
  Downloading wolframalpha-5.0.0-py3-none-any.whl (7.5 kB)  
Collecting xmltodict (from wolframalpha)  
  Downloading xmltodict-0.13.0-py2.py3-none-any.whl (10.0 kB)  
Requirement already satisfied: more-itertools in /usr/local/lib/python3.1  
0/dist-packages (from wolframalpha) (10.1.0)  
Collecting jaraco.context (from wolframalpha)  
  Downloading jaraco.context-4.3.0-py3-none-any.whl (5.3 kB)  
Installing collected packages: xmltodict, jaraco.context, wolframalpha  
Successfully installed jaraco.context-4.3.0 wolframalpha-5.0.0 xmltodict-  
0.13.0
```

```
In [ ]: # 環境変数の準備  
import os  
os.environ["WOLFRAM_ALPHA_APPID"] = "<Wolfram_AlphaのAppID>"
```

```
In [ ]: from langchain.agents import load_tools  
  
# ツールの準備  
tools = load_tools(["wolfram-alpha"])
```

```
In [ ]: from langchain.chains.conversation.memory import ConversationBufferMemory  
  
# メモリの作成  
memory = ConversationBufferMemory(  
    memory_key="chat_history",  
    return_messages=True  
)
```

```
In [ ]: from langchain.agents import initialize_agent  
from langchain.chat_models import ChatOpenAI  
  
# エージェントの作成  
agent = initialize_agent(
```

```
        agent="zero-shot-react-description",
        llm=ChatOpenAI(temperature=0),
        tools=tools,
        memory=memory,
        verbose=True
    )
```

```
In [ ]: # エージェントの実行
agent.run("東京都と大阪の距離は何km?")
```

## 記憶

### Conversation Buffer Memory

```
In [ ]: from langchain.memory import ConversationBufferMemory

# メモリの作成
memory = ConversationBufferMemory()
memory.chat_memory.add_user_message("おなかすいた")
memory.chat_memory.add_ai_message("どこか食べに行く?!")
memory.chat_memory.add_user_message("ラーメン屋に行こう")
memory.chat_memory.add_ai_message("駅前のラーメン屋だね")
memory.chat_memory.add_user_message("それでは出発!")
memory.chat_memory.add_ai_message("OK!")
```

```
In [ ]: # メモリ変数の取得
memory.load_memory_variables({})
```

```
Out[ ]: {'history': 'Human: おなかすいた\nAI: どこか食べに行く?\nHuman: ラーメン屋に行こう\nAI: 駅前のラーメン屋だね\nHuman: それでは出発!\nAI: OK! '}
```

```
In [ ]: from langchain.memory import ConversationBufferMemory

# メモリの作成
memory = ConversationBufferMemory(return_messages=True)
memory.chat_memory.add_user_message("おなかすいた")
memory.chat_memory.add_ai_message("どこか食べに行く?!")
memory.chat_memory.add_user_message("ラーメン屋に行こう")
memory.chat_memory.add_ai_message("駅前のラーメン屋だね")
memory.chat_memory.add_user_message("それでは出発!")
memory.chat_memory.add_ai_message("OK!")
```

```
In [ ]: # メモリ変数の取得
memory.load_memory_variables({})
```

```
Out[ ]: {'history': [HumanMessage(content='おなかすいた', additional_kwargs={}, example=False),
                    AIMessage(content='どこか食べに行く?', additional_kwargs={}, example=False),
                    HumanMessage(content='ラーメン屋に行こう', additional_kwargs={}, example=False),
                    AIMessage(content='駅前のラーメン屋だね', additional_kwargs={}, example=False),
                    HumanMessage(content='それでは出発!', additional_kwargs={}, example=False),
                    AIMessage(content='OK!', additional_kwargs={}, example=False)]}
```

## Conversation Buffer Window Memory

```
In [ ]: from langchain.memory import ConversationBufferWindowMemory

# メモリの作成
memory = ConversationBufferWindowMemory(k=2, return_messages=True)
memory.save_context({"input": "おなかすいた"}, {"output": "どこか食べに行く?"})
memory.save_context({"input": "ラーメン屋に行こう"}, {"output": "駅前のラーメン屋だ"})
memory.save_context({"input": "それでは出発!"}, {"output": "OK!"})

In [ ]: # メモリ変数の取得
memory.load_memory_variables([])

Out[ ]: {'history': [HumanMessage(content='ラーメン屋に行こう', additional_kwargs={}, example=False),
 AIMessage(content='駅前のラーメン屋だね', additional_kwargs={}, example=False),
 HumanMessage(content='それでは出発!', additional_kwargs={}, example=False),
 AIMessage(content='OK!', additional_kwargs={}, example=False)]}
```

## Conversation Token Buffer Memory

```
In [ ]: from langchain.memory import ConversationTokenBufferMemory
from langchain.chat_models import ChatOpenAI

# メモリの作成
memory = ConversationTokenBufferMemory(
    llm=ChatOpenAI(temperature=0),
    max_token_limit=50,
    return_messages=True
)
memory.save_context({"input": "おなかすいた"}, {"output": "どこか食べに行く?"})
memory.save_context({"input": "ラーメン屋に行こう"}, {"output": "駅前のラーメン屋だ"})
memory.save_context({"input": "それでは出発!"}, {"output": "OK!"})

In [ ]: # メモリ変数の取得
memory.load_memory_variables([])

Out[ ]: {'history': [AIMessage(content='駅前のラーメン屋だね', additional_kwargs={}, example=False),
 HumanMessage(content='それでは出発!', additional_kwargs={}, example=False),
 AIMessage(content='OK!', additional_kwargs={}, example=False)]}
```

## Conversation Summary Memory

```
In [ ]: from langchain.memory import ConversationSummaryMemory
from langchain.chat_models import ChatOpenAI

# メモリの作成
memory = ConversationSummaryMemory(llm=ChatOpenAI(temperature=0), return_
memory.save_context({"input": "おなかすいた"}, {"output": "どこか食べに行く?"})
memory.save_context({"input": "ラーメン屋に行こう"}, {"output": "駅前のラーメン屋だ"})
memory.save_context({"input": "それでは出発!"}, {"output": "OK!"})
```

```
In [ ]: # メモリ変数の取得  
memory.load_memory_variables({})
```

```
Out[ ]: {'history': [SystemMessage(content='The human expresses hunger and suggests going to a ramen restaurant. The AI agrees and suggests going to a ramen restaurant near the train station. The human says "Let's go!" and the AI responds with "OK!"', additional_kwargs={})]}
```

## Conversation Summary Buffer Memory

```
In [ ]: from langchain.memory import ConversationSummaryBufferMemory  
from langchain.chat_models import ChatOpenAI
```

```
# メモリの作成  
memory = ConversationSummaryBufferMemory(  
    llm=ChatOpenAI(temperature=0),  
    max_token_limit=50,  
    return_messages=True  
)  
memory.save_context({"input": "おなかすいた"}, {"output": "どこか食べに行く?")  
memory.save_context({"input": "ラーメン屋に行こう"}, {"output": "駅前のラーメン屋だね")  
memory.save_context({"input": "それでは出発!"}, {"output": "OK!"})
```

```
In [ ]: # メモリ変数の取得  
memory.load_memory_variables({})
```

```
Out[ ]: {'history': [SystemMessage(content='The human expresses hunger and suggests going to a ramen restaurant. The AI asks if they want to go somewhere to eat.', additional_kwargs={}),  
    AIMessage(content='駅前のラーメン屋だね', additional_kwargs={}, example=False),  
    HumanMessage(content='それでは出発!', additional_kwargs={}, example=False),  
    AIMessage(content='OK!', additional_kwargs={}, example=False)]}
```

```
In [ ]:
```