



SetupAI

Développez votre croissance grâce à l'Intelligence Artificielle

AMBIENT°IT



Mila



Lanchain

-

Construire des Applications avec des LLMs



Première Journée

- I. Tour de table
- II. Prérequis: quelles notions de bases pour le cours
- III. Objectifs du cours
- IV. Outils et support
- V. Introduction
- VI. Cours - Partie 1



I - Tour de table

- Présentons-nous !
- Quel rôle occupez-vous au sein de la Société Générale?
- Quel est votre domaine d'expertise?
- A quoi ressemble votre routine de travail?
- Avez-vous déjà utilisé LangChain? Si, oui dans quel contexte?
- Quels sont vos attentes vis-à-vis de la formation?
- Quels sont vos objectifs à l'issue de cette formation?



II - Prérequis

- Connaissances de bases en Python
- Avoir déjà travailler sur des Notebook Jupyter/Google Colab
- Une clé OpenAI API (pour call notre LLM)
- Quelques notions en NLP (pas obligatoire)
- D'autres clé API externes marqués dans les prérequis Ambient-IT



III - Objectifs du cours

- Créer et déployer des agents
- Conception et personnalisation de Chatbots
- Création d'un agent personnalisé
- Maîtriser la compréhension du code et de l'écosystème LangChain





IV - Outils et support

- Python
- Google Colab ou Jupyter Notebook
- API OpenAI
- D'autres API dont on parlera après (Exa, Tavily, Pinecone, etc...)
- Documentation LanChain



V - Introduction

- Rappels sur les LLMs
- Limitation des LLMs
- Solution à ces limitations :  
- Pricing des APIs
- Composantes Clés



V - Introduction : Rappels sur les LLMs

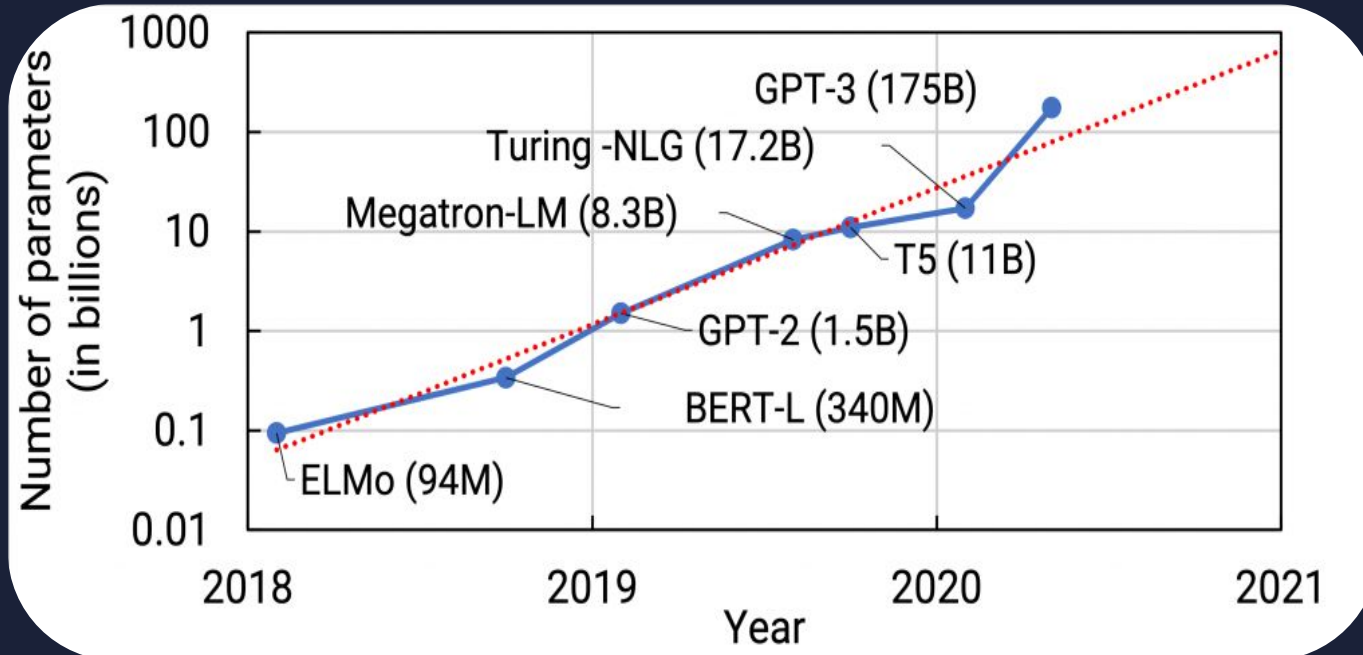
Qu'est-ce qu'un LM (Language Model):

- Modèle probabiliste de traitement du langage naturel.
- Ces modèles sont multitâches et peuvent effectuer différentes actions en interprétant l'entrée de l'utilisateur tel que:
 - Génération de texte
 - Traduction
 - Résumé de document
 - Classification de texte
 - etc...



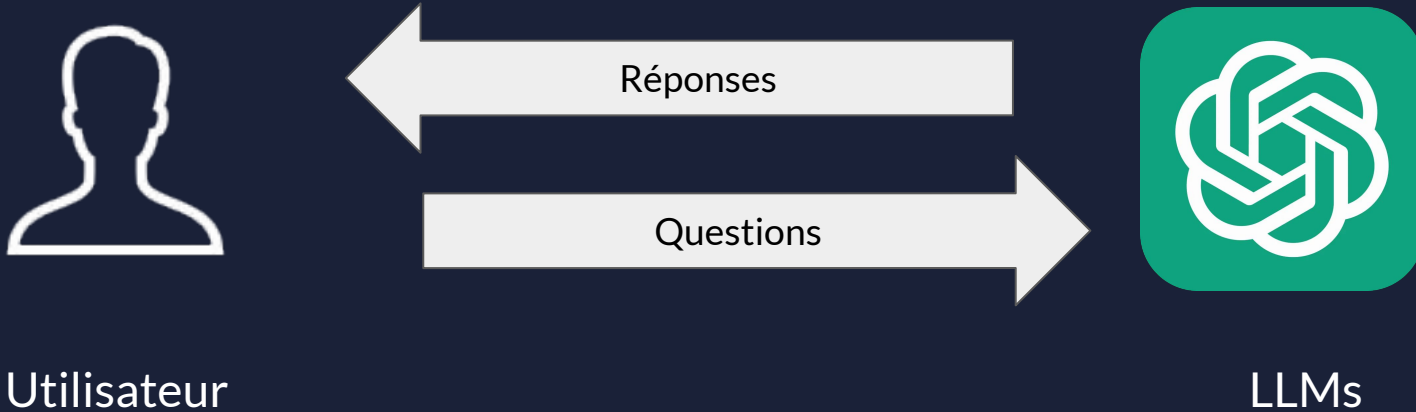
V - Introduction : Rappels sur les LLMs

Qu'est-ce qui les rends "Large" ?



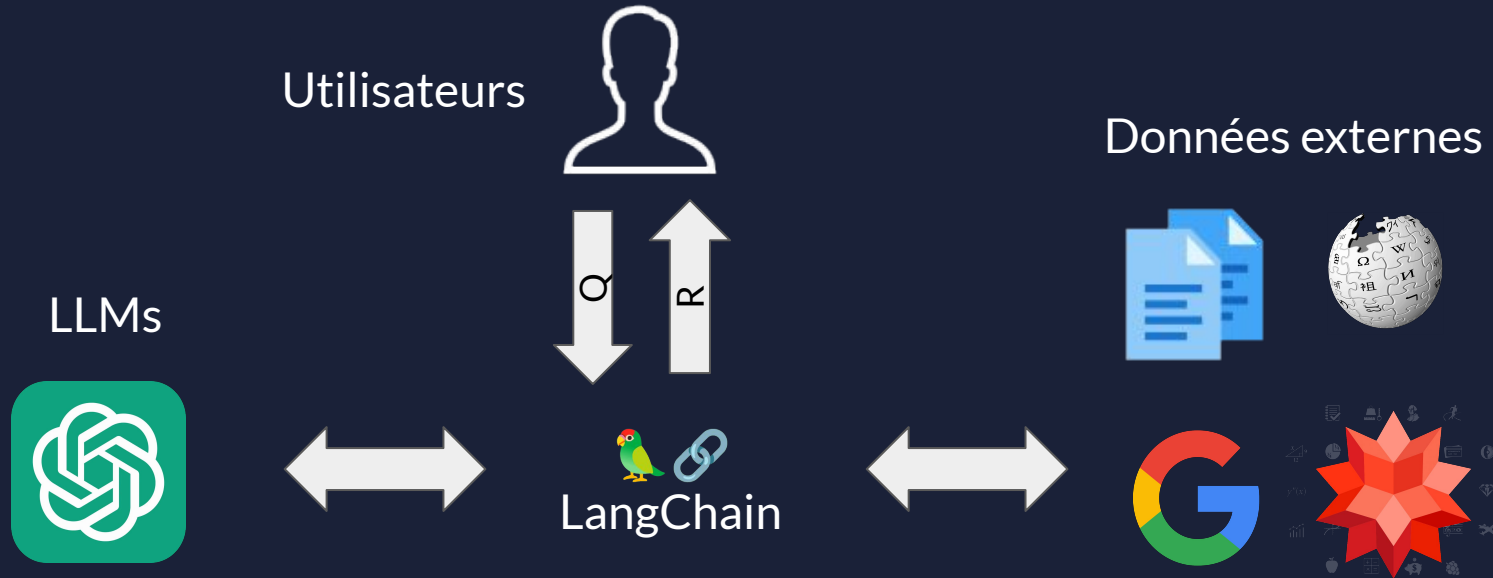
V - Introduction : Limitation des LLMs

Le LLM n'a aucun visuel sur les données externes.



V - Introduction : Solutions à ces limitations

LangChain permet au LLM de communiquer avec le monde extérieur.



V - Introduction : Pricing des API

À chaque création de compte, OpenAI vous donne 5\$ pour accéder à leur modèles.

Notez que GPT-4 est uniquement accessible aux utilisateurs ayant +10\$ sur leur compte API.

Model	Input	Output
gpt-3.5-turbo-0125	\$0.0005 / 1K tokens	\$0.0015 / 1K tokens
gpt-3.5-turbo-instruct	\$0.0015 / 1K tokens	\$0.0020 / 1K tokens



V - Introduction : Composantes clés

- LangChain est écrit en Python et JS
- On se concentrera sur la version Python dans ce cours
- Les principales applications du framework >
 - Analyse de documents
 - Résumé de documents
 - Chatbots Q/R



VI - Cours : Partie 1

- Chapitre 1 - Model I/O
- Chapitre 2 - Retriever
- Chapitre 3 - Chaînage
- Chapitre 4 - Agent
- Chapitre 5 - Memory
- Chapitre 6 - Callbacks



VI - Cours : Chapitre 1 - Model I/O

Dans ce chapitre nous allons voir comment utiliser l'interface LangChain avec les modèles de langage.

Cela inclut principalement une interface pour définir le modèle, des outils d'assistance pour construire les entrées des modèles, ainsi que des outils d'assistance pour travailler avec les sorties des modèles.

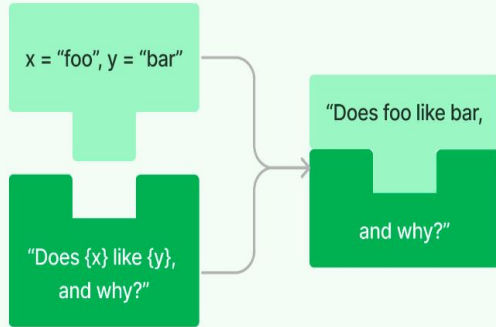
Nous allons voir comment désigner un Prompt sur LangChain, comment call notre LLM et comment formater la sortie de nos requêtes avec des Parsers.



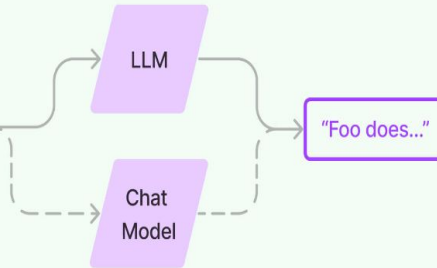
VI - Cours : Chapitre 1 - Model I/O

Model I/O

Format



Predict



Parse



VI - Cours : Chapitre 1 - Model I/O

Prompt :

```
prompt="""Réponds à la question grâce au contexte si dessous. S'il n'y a pas de réponses possible à la question avec les informations fournies, réponds simplement "jsp".  
Context:Les grands modèles de langage (LLM) sont les modèles les plus récents utilisés dans le domaine du NLP.  
Leurs performances supérieures à celles des modèles plus petits les ont rendus incroyablement utiles pour les développeurs d'applications NLP.  
Ces modèles sont accessibles via la bibliothèque `transformers` de Hugging Face, via OpenAI en utilisant la bibliothèque `openai`,  
et via Cohere en utilisant la bibliothèque `cohere`.  
Question: Quelle bibliothèque et fournisseurs de modèles proposent des LLMs?  
Réponse: """
```



VI - Cours : Chapitre 1 - Model I/O

Prompt Engineering: Bonne pratique

Un bon Prompt d'entrée contient (souvent) au moins un des points suivants:

- Instructions
- Contexte extérieur
- Entrée utilisateur
- Indicateur de l'output



VI - Cours : Chapitre 1 - Model I/O

Prompt Template

```
from langchain import PromptTemplate
```

```
template = """Réponds à la question grâce au contexte si dessous. S'il n'y a pas de réponses possible à la question  
avec les informations fournies,  
réponds simplement "jsp".
```

Context: Les grands modèles de langage (LLM) sont les modèles les plus récents utilisés dans le domaine du NLP. Leurs performances supérieures à celles des modèles plus petits les ont rendus incroyablement utiles pour les développeurs d'applications NLP.

Ces modèles sont accessibles via la bibliothèque `transformers` de Hugging Face, via OpenAI en utilisant la bibliothèque `openai`, et via Cohere en utilisant la bibliothèque `cohere`.

```
Question: {query}
```

```
Answer: """
```

```
prompt_template = PromptTemplate(  
    input_variables=["query"],  
    template=template  
)
```



VI - Cours : Chapitre 1 - Model I/O

Prompt Template

Il est possible de donner plusieurs paramètres à son prompt:

```
from langchain.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(
    "Raconte-moi une blague {adjective} sur {content}."
)
```



VI - Cours : Chapitre 1 - Model I/O

Prompt Template

Les PromptTemplate peuvent faciliter le prompt engineering au call

```
print(openai.invoke(prompt_template.format(adjective="drole",  
                                          content="la police")))
```

Pourquoi les policiers aiment-ils les escaliers ?

Parce qu'ils peuvent monter les marches sans être vus !



VI - Cours : Chapitre 1 - Model I/O

Few-Shot PromptTemplate

Avantages: Calquer le comportement du LLM sur des exemples

```
from langchain import FewShotPromptTemplate

# create our examples
examples = [
    {
        "query": "Comment ca-va?",
        "answer": "Je ne suis pas à plaindre me je le fais quand-même."
    }, {
        "query": "Quel heure est-il?",
        "answer": "L'heure de vous acheter une montre."
    }, {
        "query": "À quoi sert un baromètre?",
        "answer": "À feur."
    }
]
```



VI - Cours : Chapitre 1 - Model I/O

Few-Shot PromptTemplate

```
# create a example template
example_template = """
User: {query}
AI: {answer}
"""

# create a prompt example from above template
example_prompt = PromptTemplate(
    input_variables=["query", "answer"],
    template=example_template
)

# now break our previous prompt into a prefix and suffix
# the prefix is our instructions
prefix = """Voici des extraits de conversations avec un
assistant d'IA.
L'assistant est généralement sarcastique et plein d'esprit,
produisant des réponses créatives et amusantes aux
questions des utilisateurs.
Voici quelques exemples:
"""

# and the suffix our user input and output indicator
suffix = """
User: {query}
AI: """
```

```
# now create the few shot prompt
template
few_shot_prompt_template =
FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix=prefix,
    suffix=suffix,
    input_variables=["query"],
    example_separator="\n\n"
)
```



VI - Cours : Chapitre 1 - Model I/O

FewShotPromptTemplate

Le cas d'utilisation peut être variable en fonction de l'historique de données:

```
query = "Quel est le sens de la vie?"  
openai.invoke(few_shot_prompt_template.format(query=query))
```

```
"42, selon les Monty Python. Mais je pense que c'est plus compliqué que ça."
```



VI - Cours : Chapitre 1 - Model I/O

LLMs et ChatModel

Les composants clés de 🦜🔗 LangChain



VI - Cours : Chapitre 1 - Model I/O

LLMs

Il existe de nombreux fournisseurs de LLM (OpenAI, Cohere, Hugging Face, ...)



cohere



...



VI - Cours : Chapitre 1 - Model I/O

LLM

```
from langchain_openai import OpenAI

llm = OpenAI(
    model_name="gpt-3.5-turbo-instruct",
    openai_api_key=OPENAI_API_KEY,
    temperature=0.9
)
text = "Écris-moi un plan d'exposé sur la boxe"
#prompt
print(llm.invoke(text))
```

- I. Introduction
 - A. Présentation de la boxe
 - B. Origines et évolution de la boxe
 - C. Objectif de l'exposé
- II. Les règles de la boxe
 - A. Les catégories de poids
 - B. Les équipements nécessaires
 - C. Déroulement d'un combat de boxe
- III. Les différentes techniques de boxe
 - A. Le jab
 - B. Le crochet
 - C. L'uppercut
 - D. Les coups de pieds
- IV. Les bienfaits de la boxe
 - A. Pour la santé physique
 - B. Pour la santé mentale
 - C. La pédagogie et la discipline
- V. La boxe dans l'histoire
 - A. Les grands noms de la boxe
 - ...
 - C. Les revenus des boxeurs



VI - Cours : Chapitre 1 - Model I/O

ChatModel

Prends en entrée une suite de messages et retourne un message.

Restructure le Prompt.

```
messages = [  
    SystemMessage(content="Tu es un assistant IA efficace donnant des réponses courtes à  
l'utilisateurs."),  
    HumanMessage(content="J'aime la montagne. Où devrais-je aller pour les vacances d'hiver?"),  
    AIMessage(content="Les Alpes pour des vacances au ski semblent être une bonne idée"),  
    HumanMessage(content="J'aimerais connaître le nombre d'espèces d'arachnides connues existant sur  
Terre.")  
]  
chat.invoke(messages)
```



VI - Cours : Chapitre 1 - Model I/O

Output Parser:

Les 'Parseurs' de sortie sont chargés de prendre la sortie d'un LLM et de la transformer dans un format plus approprié.

En plus de disposer d'une large collection de différents types d'analyseurs de sortie, l'un des avantages distinctifs des LangChain `OutputParsers` est que nombre d'entre eux prennent en charge la diffusion en continu "Streaming".



VI - Cours : Chapitre 1 - Model I/O

Quelques exemples de format:

- CSV
- JSON
- XML
- Markdown
- Etc...



VI - Cours : Chapitre 1 - Model I/O

Exemple en Output de DataFrame

```
import pprint
from typing import Any, Dict
import pandas as pd
from langchain.output_parsers import PandasDataFrameOutputParser
from langchain.prompts import PromptTemplate
from langchain_openai import ChatOpenAI
model = ChatOpenAI(temperature=0,api_key=OPENAI_API_KEY)
# Solely for documentation purposes.
def format_parser_output(parser_output: Dict[str, Any]) -> None:
    """
    Print la valeur de la requête sans print le dictionnaire
    """
    for key in parser_output.keys():
        parser_output[key] = parser_output[key].to_dict()
    return pprint.PrettyPrinter(width=4, compact=True).pprint(parser_output)
# Define your desired Pandas DataFrame.
df = pd.DataFrame(
    {
        "num_legs": [2, 4, 8, 0],
        "num_wings": [2, 0, 0, 0],
        "num_specimen_seen": [10, 2, 1, 8],
    }
)
# Set up a parser + inject instructions into the prompt template.
parser = PandasDataFrameOutputParser(dataframe=df)
```



VI - Cours : Chapitre 1 - Model I/O

Exemple en Output de DataFrame

```
# Here's an example of a column operation being performed.
df_query = "Donne-moi la colonne 'num_wings'."

# Set up the prompt.
prompt = PromptTemplate(
    template="Réponds à la requête de l'utilisateur.\n{format_instructions}\n{query}\n",
    input_variables=["query"],
    partial_variables={"format_instructions": parser.get_format_instructions()},
)

chain = prompt | model | parser
parser_output = chain.invoke({"query": df_query})

format_parser_output(parser_output)
```



VI - Cours : Chapitre 1 - Model I/O

Exemple en Output de DataFrame

```
{'num_wings': {0: 2,  
              1: 0,  
              2: 0,  
              3: 0}}
```

	num_legs	num_wings	num_specimen_seen
0	2	2	10
1	4	0	2
2	8	0	1
3	0	0	8



VI - Cours : Chapitre 1 - Model I/O

TP:

Tâche 1 : Prompt Engineering

- Créez des Prompt Templates et comprenez comment fournir des instructions au LLM.

Tâche 2 : Analyse de sortie

- Explorez les analyseurs de sortie et comprenez comment transformer la réponse brute du modèle de langage dans un format structuré.
- Apprenez à utiliser les analyseurs de sortie pour rendre la sortie du modèle plus exploitable pour une utilisation en aval.



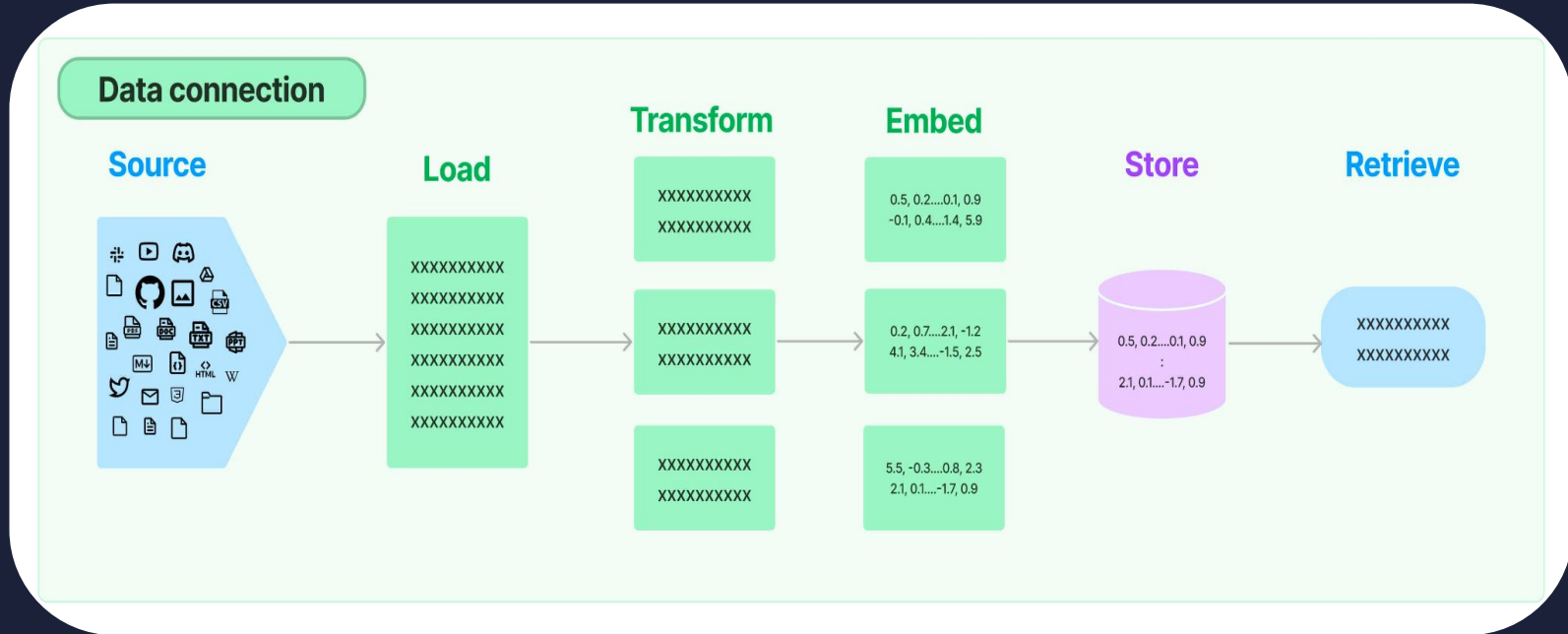
VI - Cours : Chapitre 2 - Retriever

Dans ce chapitre nous allons voir les éléments constitutifs des applications RAG avec le **Retriever**.

Ce chapitre couvre ce qui a trait à l'étape d'extraction, c'est-à-dire la récupération des données. Bien que cela paraisse simple, cela peut être subtilement complexe. Elle englobe plusieurs modules clés.



VI - Cours : Chapitre 2 - Retriever



VI - Cours : Chapitre 2 - Retriever

Retrieval: Composantes:

- Document Loader
- Découpage de texte (Splitter)
- Embedding de texte
- Stockage de vecteurs
- Retrievers



VI - Cours : Chapitre 2 - Retriever

Document Loader:

Permet de charger vos documents au retriever.

Loader prends en charge plusieurs format: " Markdown, Txt, CSV, Site internet, etc..."

```
from langchain_community.document_loaders import TextLoader

loader = TextLoader("./index.md")
loader.load()
```



VI - Cours : Chapitre 2 - Retriever

Splitter

```
from langchain.text_splitter import MarkdownHeaderTextSplitter

with open('doc.md', 'r') as file:
    markdown_doc = file.read()

headers_to_split_on = [
    ("#", "Header 1"),
    ("##", "Header 2"),
    ("###", "Header 3"),
    ("####", "Header 4"),
]

markdown_splitter =
MarkdownHeaderTextSplitter(headers_to_split_on=headers_to_split_on)
md_header_splits = markdown_splitter.split_text(markdown_doc)
md_header_splits
```



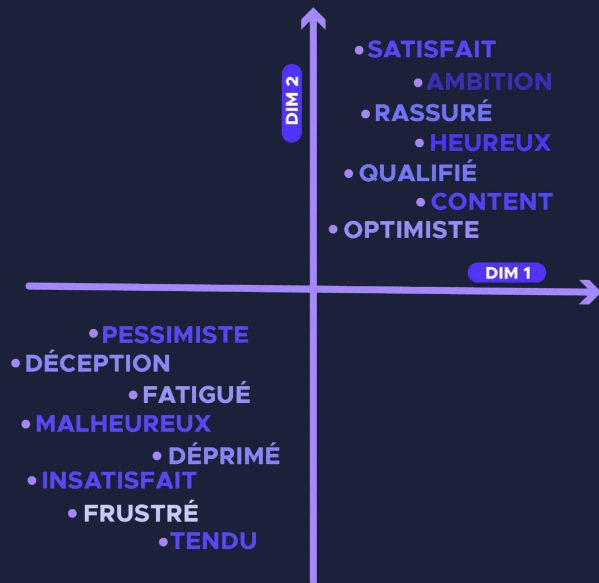
VI - Cours : Chapitre 2 - Retriever

Embedding de texte



VI - Cours : Chapitre 2 - Retriever

Embedding de texte



```
from langchain_openai import OpenAIEmbeddings

model_name = 'text-embedding-3-small'

embed = OpenAIEmbeddings(
    model=model_name,
    openai_api_key=OPENAI_API_KEY
)
```

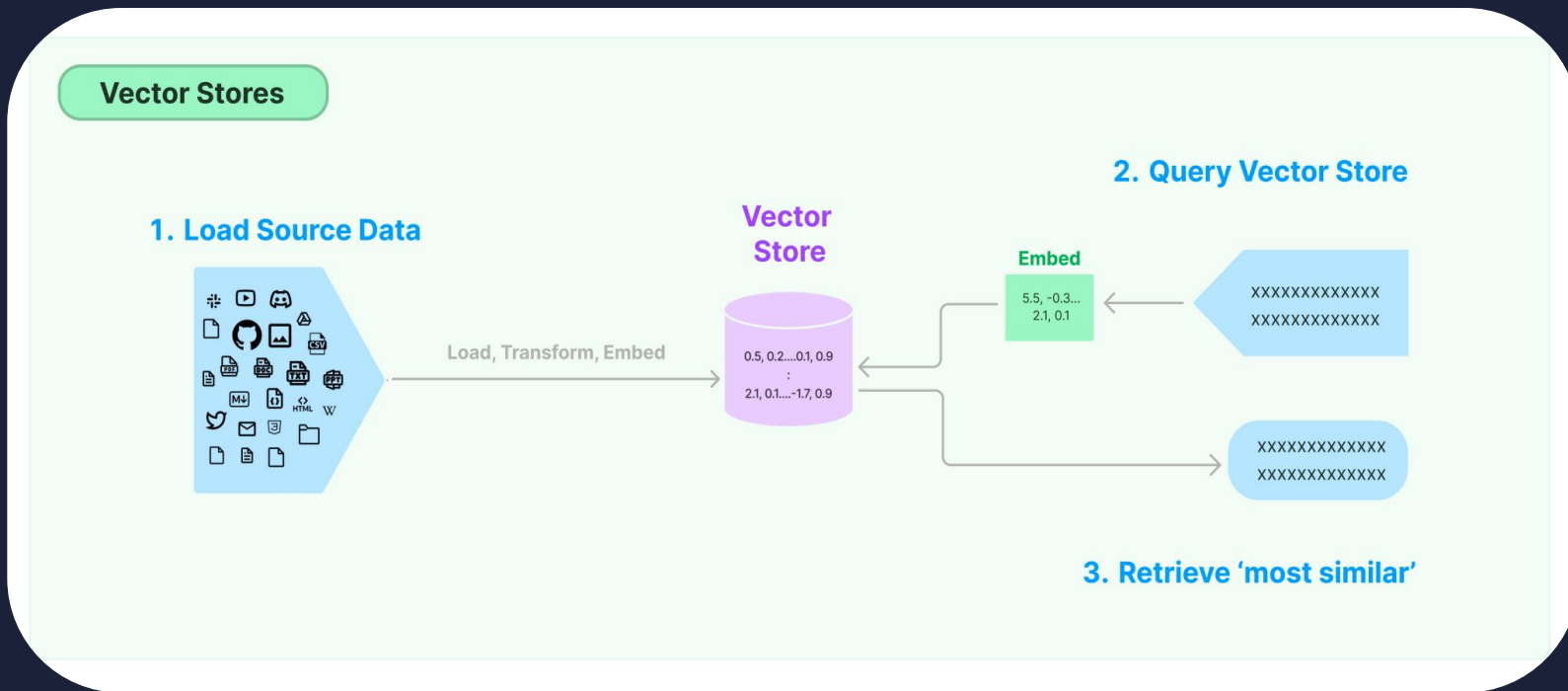
```
texts = [
    'Ceci est un morceau de texte',
    'puis ceci est un autre bout de texte'
]

print(len(texts), len(texts[0]))
res = embed.embed_documents(texts)
print(len(res), len(res[0]))
```



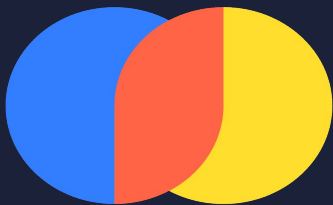
VI - Cours : Chapitre 2 - Retriever

Stockage de vecteurs



VI - Cours : Chapitre 2 - Retriever

Stockage de vecteurs - Vector DB



Pinecone



VI - Cours : Chapitre 3 - Chaînage

Dans ce chapitre nous allons voir le concept de **chaînage**.

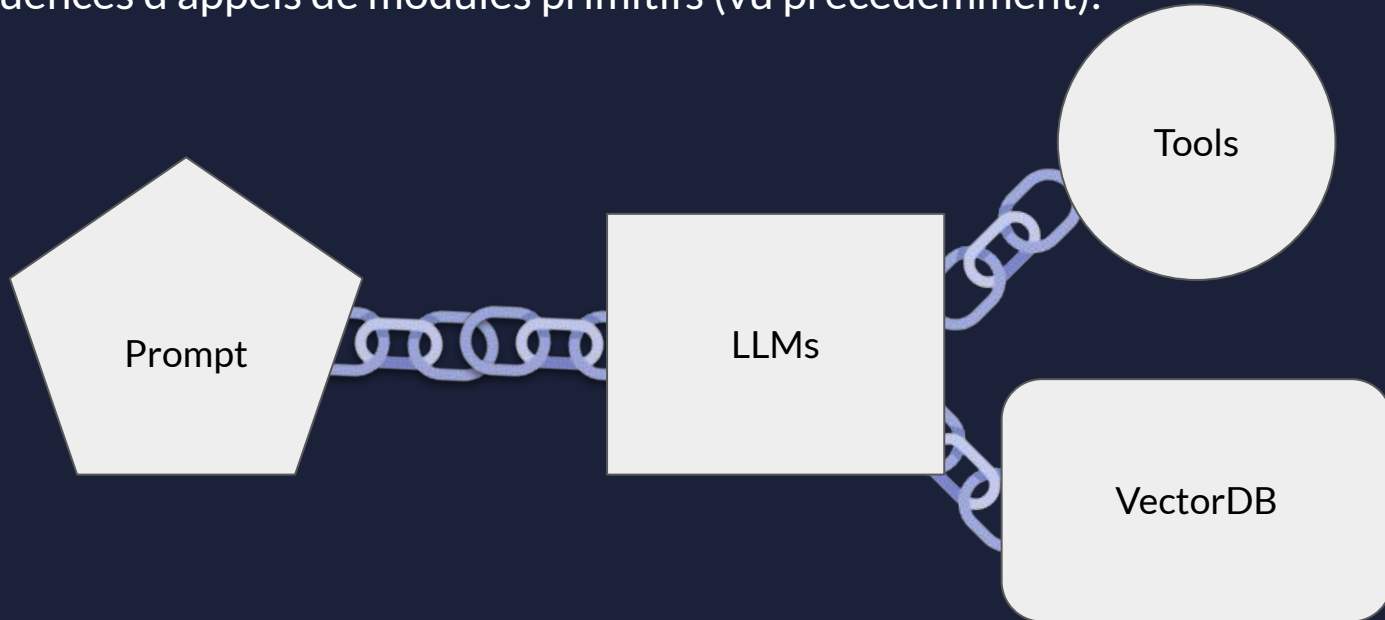
Les chaînes font référence à des séquences d'appels, que ce soit à un LLM, à un outil ou à une étape de prétraitement des données.



VI - Cours : Chapitre 3 - Chaînage

Chain

Séquences d'appels de modules primitifs (vu précédemment):



VI - Cours : Chapitre 3 - Chaînage

LangChain Expression Language (LCEL):

Normalisation de la composition de chaînes.

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

prompt = ChatPromptTemplate.from_template("Raconte-moi une blague courte sur {topic}.")
model = ChatOpenAI()
output_parser = StrOutputParser()

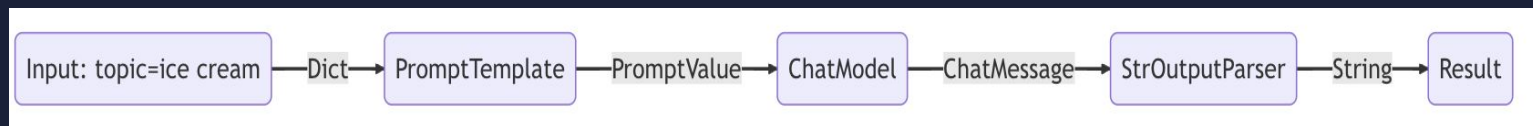
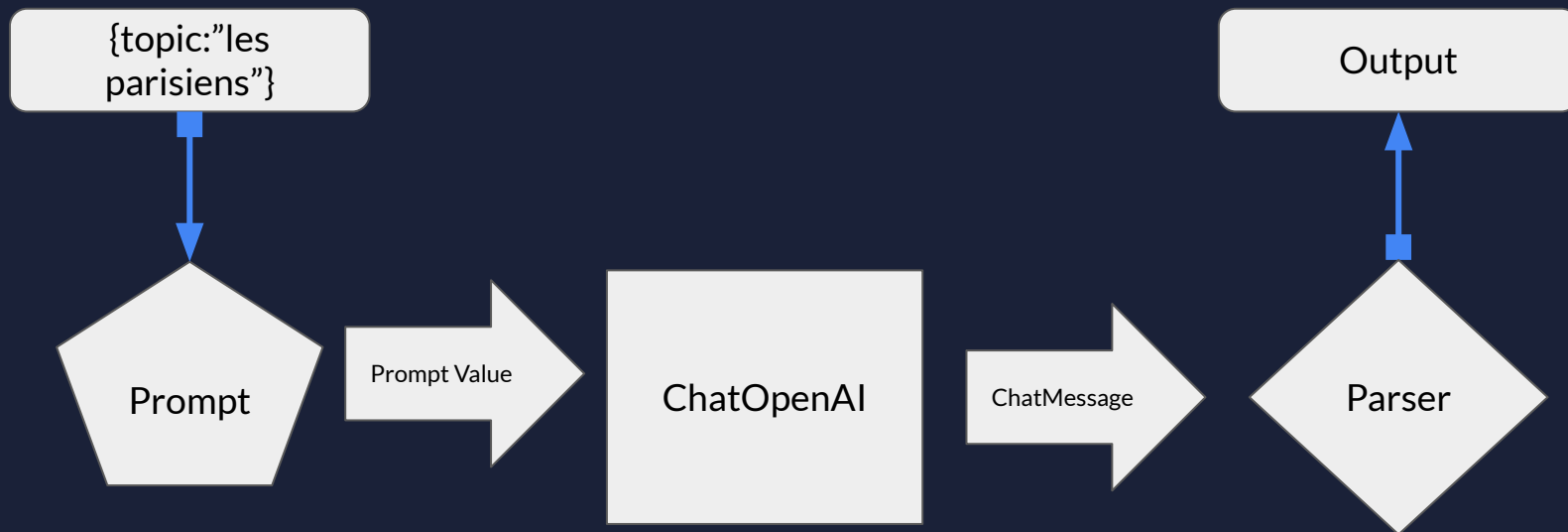
chain = prompt | model | output_parser

chain.invoke({"topic": "les parisiens"})
```



VI - Cours : Chapitre 3 - Chaînage

Pipeline du LCEL



VI - Cours : Chapitre 4 - Agent

Dans ce chapitre nous allons parcourir le concept **d'Agent**.

L'idée centrale des agents est d'utiliser un LLM pour choisir une séquence d'actions à entreprendre.

Dans les chaînes, une séquence d'actions est codée en dur (dans le code).

Dans les agents, un modèle de langage est utilisé comme moteur de raisonnement pour déterminer les actions à entreprendre et leur ordre.



VI - Cours : Chapitre 4 - Agent

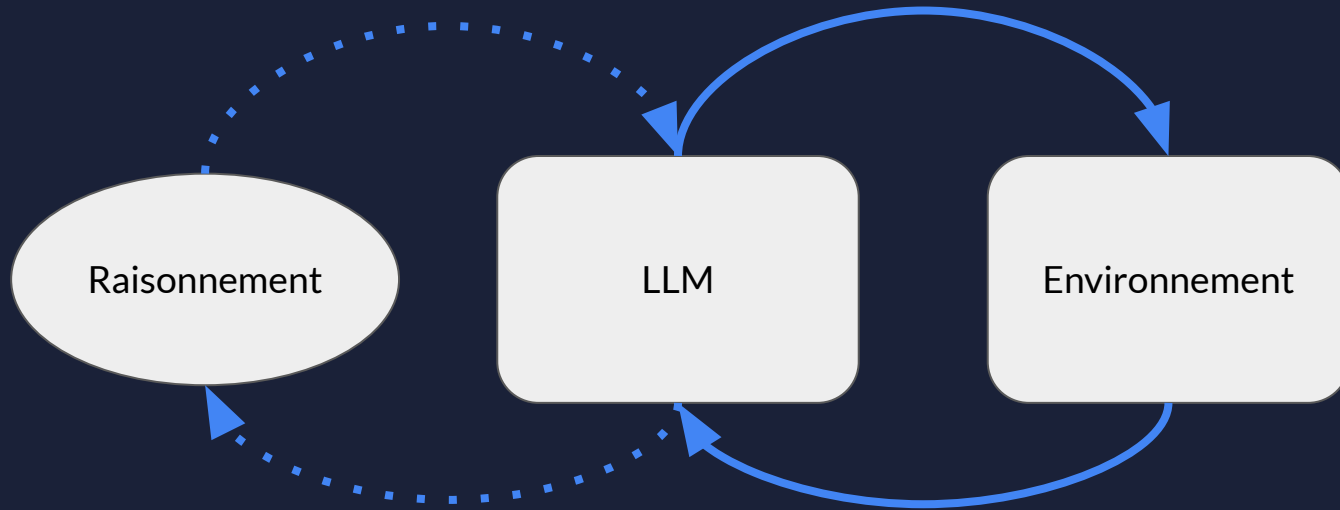
Agent:

Un agent communique avec différents outils (tools) pour donner du contexte au LLM call via l'API.



VI - Cours : Chapitre 4 - Agent

Framework d'un Agent



VI - Cours : Chapitre 4 - Agent

Les outils (tools):

Les outils de LangChain permettent aux agents d'intégrer des services externes, en offrant des interfaces modulaires et des capacités telles que:

- Chargeurs de documents
- Base de données de vecteurs
- Outils de calculs
- Cartes
- etc...



VI - Cours : Chapitre 4 - Agent

TP:

Tâche 1 : Mise en œuvre du modèle

- Mettez en œuvre une chaîne LLM simple et une chaîne de récupération à l'aide de LangChain.
- Utilisez le LangChain Expression Language (LCEL) pour définir une interface Runnable unifiée pour les modules.



VI - Cours : Chapitre 4 - Agent

TP:

Tâche 2 : Mise en œuvre d'un agent

- Mettez en œuvre un agent qui utilise un LLM comme moteur de raisonnement pour prendre des décisions sur les actions à entreprendre.
- Définissez un agent, créez des outils et utilisez l'AgentExecutor pour exécuter l'agent et observer le processus de prise de décision basé sur les réponses du LLM



VI - Cours : Chapitre 5 - Memory

Dans ce chapitre nous allons voir comment fonctionne le module **Memory**.

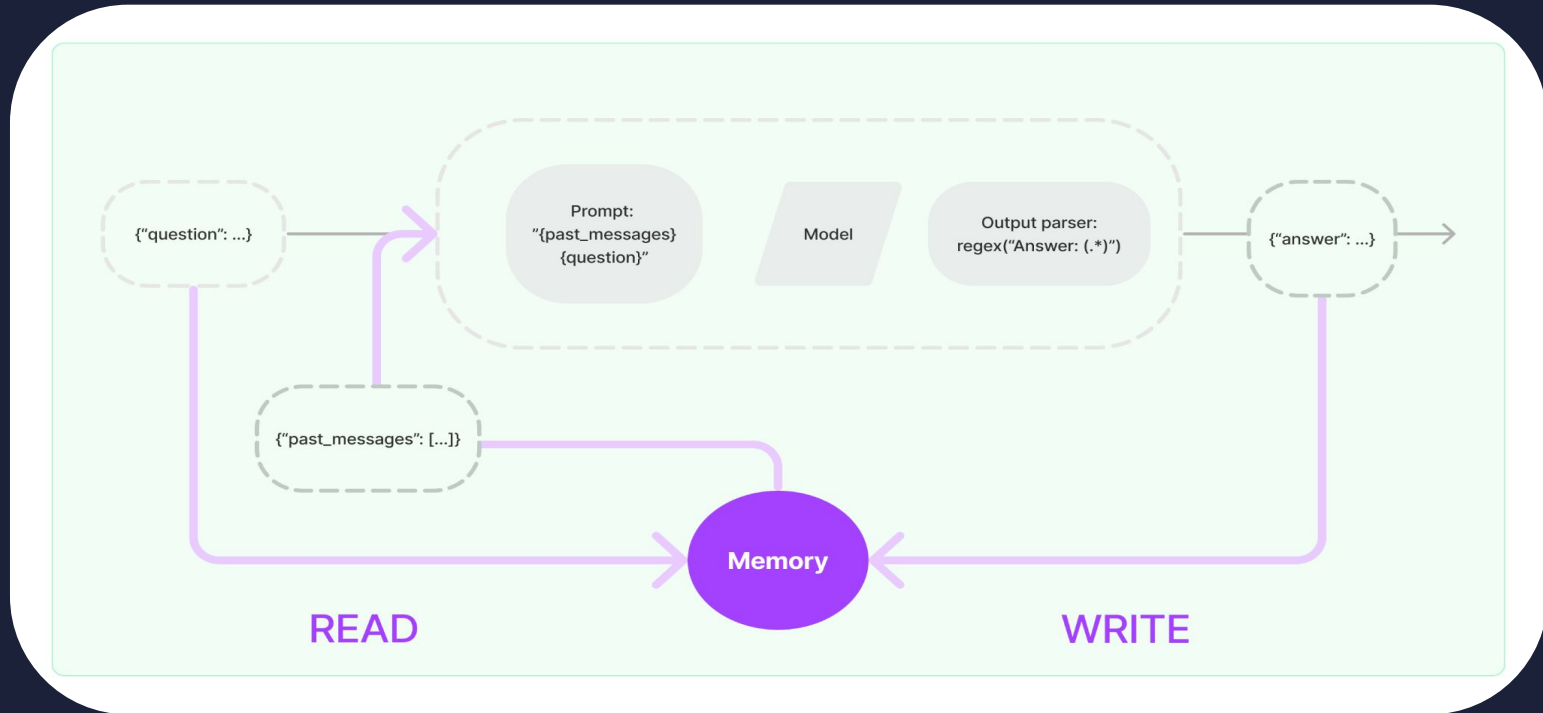
Un système de mémoire doit prendre en charge deux actions de base : la lecture et l'écriture. Rappelons que chaque chaîne définit une logique d'exécution de base qui attend certaines entrées.

Certaines de ces entrées proviennent directement de l'utilisateur, mais d'autres peuvent provenir de la mémoire des échanges précédents.



VI - Cours : Chapitre 5 - Memory

Memory



VI - Cours : Chapitre 5 - Memory

Ajouter l'historique des messages: ChatMessageHistory

```
from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_core.chat_history import BaseChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory

store = {}

def get_session_history(session_id: str) -> BaseChatMessageHistory:
    if session_id not in store:
        store[session_id] = ChatMessageHistory()
    return store[session_id]

with_message_history = RunnableWithMessageHistory(
    runnable,
    get_session_history,
    input_messages_key="input",
    history_messages_key="history",
)
```



VI - Cours : Chapitre 5 - Memory

ChatMessageHistory

```
with_message_history.invoke(  
    {"ability": "math", "input": "Qu'est-ce que l'hypothalamus?"},  
    config={"configurable": {"session_id": "abc123"}},  
)
```

```
AIMessage(content="L'hypothalamus est une région du cerveau qui régule diverses fonctions corporelles telles que la température et l'appétit.")
```



VI - Cours : Chapitre 5 - Memory

ChatMessageHistory

```
# Remembers
with_message_history.invoke(
    {"ability": "math", "input": "Reformule en une dizaine de mots."},
    config={"configurable": {"session_id": "abc123"}},
)
```

```
AIMessage(content='Région du cerveau qui contrôle fonctions corporelles comme température et appétit.')
```



VI - Cours : Chapitre 6 - Callback

Dans ce chapitre nous allons voir le modules callbacks.

LangChain fournit un système de callbacks qui vous permet de vous connecter aux différentes étapes de votre application LLM. Ce système est utile pour la journalisation, la surveillance, la diffusion en continu et d'autres tâches.



VI - Cours : Chapitre 6 - Callback

- Utile pour la supervision et le tracking des chaînes.
- Permet de tracker les call à la requête
- *N'inclut pas le LCEL*



VI - Cours : Chapitre 6 - Callback

```
from langchain.callbacks import StdOutCallbackHandler
from langchain.chains import LLMChain
from langchain_openai import OpenAI
from langchain.prompts import PromptTemplate

handler = StdOutCallbackHandler()
llm = OpenAI(api_key=OPENAI_API_KEY)
prompt = PromptTemplate.from_template("1 + {number} = ")

# Constructor callback :
#Tout d'abord, définissons explicitement le StdOutCallbackHandler lors de l'initialisation de notre chaîne
chain = LLMChain(llm=llm, prompt=prompt, callbacks=[handler])
chain.invoke({"number":2})
```



VI - Cours : Chapitre 6 - Callback

> Entering new LLMChain chain...

Prompt after formatting:

1 + 2 =

> Finished chain.

```
{'number': 2, 'text': '3\n\n3 is the sum of 1 and 2.'}
```

