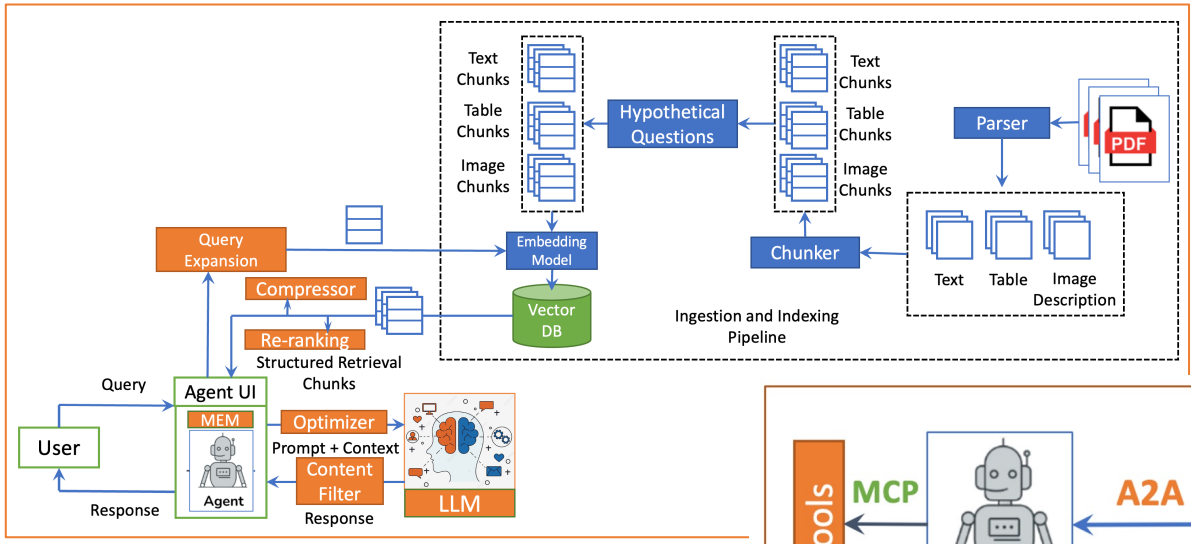
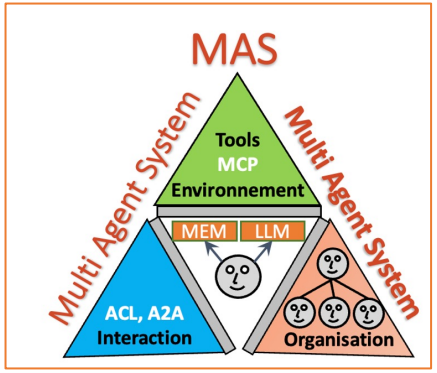
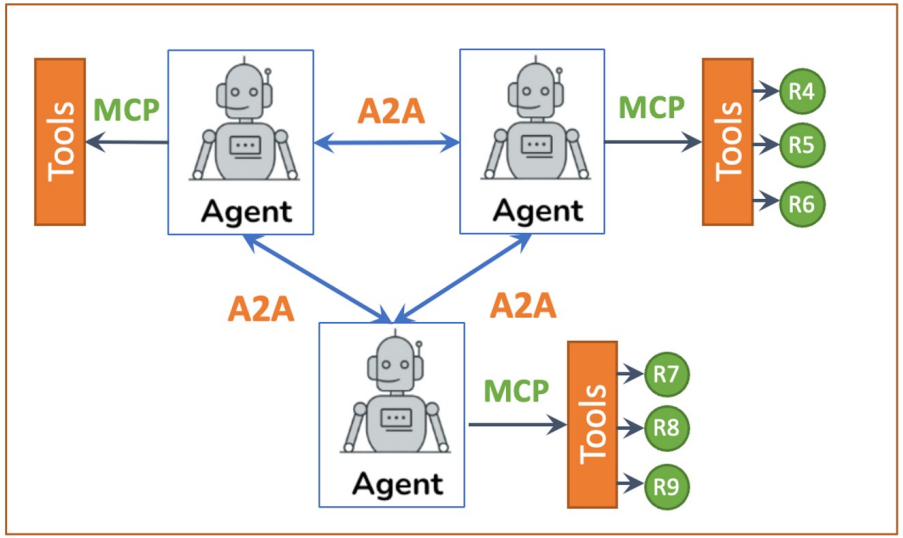
Advanced Multi Modal Agentic RAG

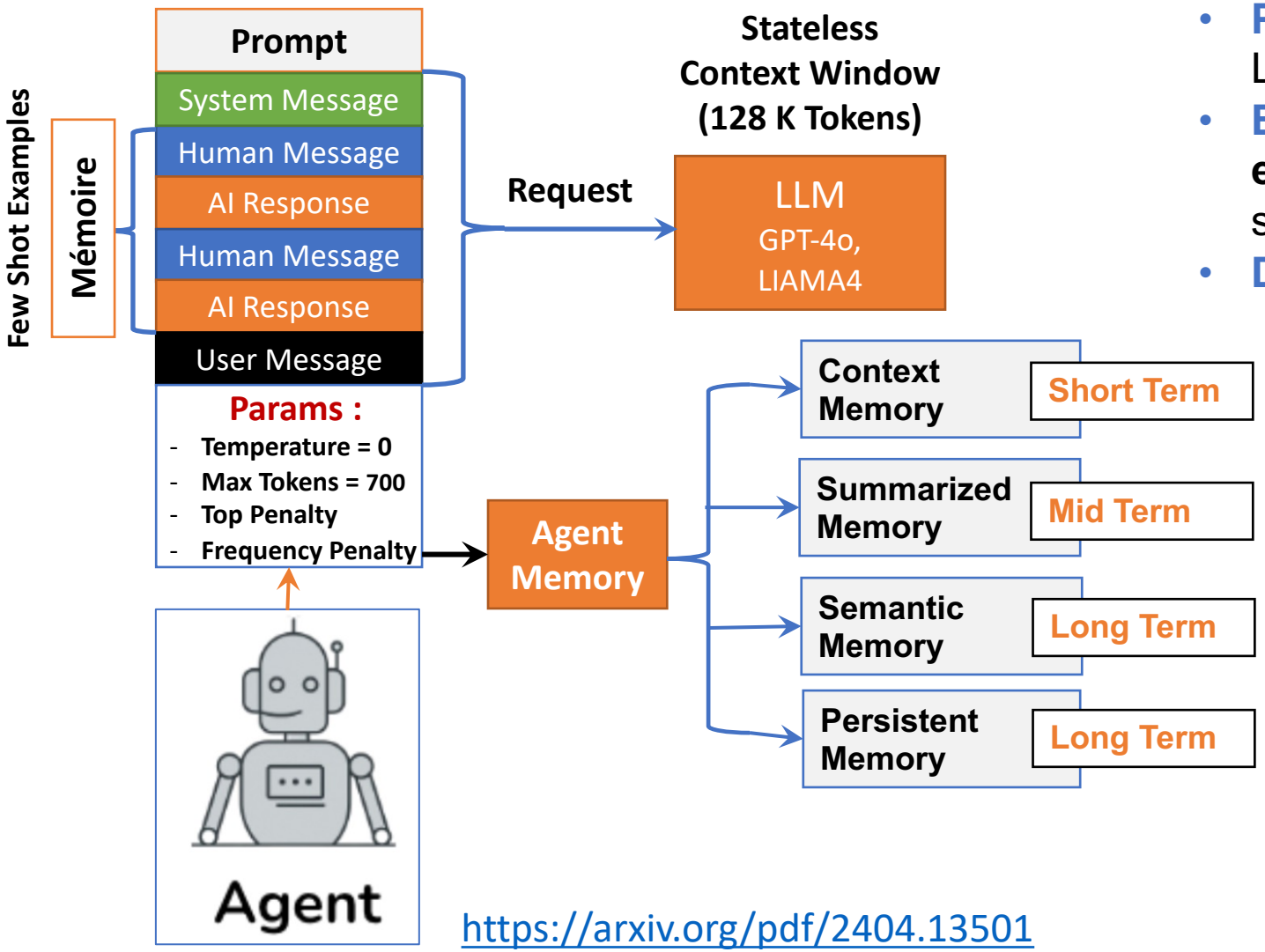
Mohamed Youssfi, Enseignant Chercheur, ENSET Mohammedia, Directeur Laboratoire Informatique, Intelligence Artificielle et Cyber Sécurité
https://www.youtube.com/@mohamedyoussfi







Prompt Engineering, Reasoning and Memorization



https://arxiv.org/pdf/2404.13501

Prompt Engineering:

- **Prompt** = Specific set of instructions sent to a LLM to accomplish a task
- **Engineering** = Process of **designing**, **evaluating** and **deploying** the prompt for specific tasks
- **Différents types de prompts :**
  - **Structure :**
    - **Zero Shot Prompt**
    - **Few Shot Prompt**
  - **Contenu :**
    - **Instruction prompt (Invite pédagogique)**
    - **Reasoning prompt (Invite de Raisonnement)**
    - **Induction prompt (Invite d'induction)**
    - **Paraphrasing prompt (Invite de paraphrase)**
    - **Chain-of-Thought Prompt (Invite de la chaine de pensée)**

Leverage Generative AI

How to leverage Generative AI

Few Shot Learning

Few Shot Prompt
{ System Message, Examples (DataSet), User Query }

User App

HTTP, JSON

Réponse

Generic LLM (Large Language Model)

Fine Tuning

Specific Data Set

LLM (Large Language Model)

Train

Specific LLM (Large Language Model)

Zero Shot Prompt
{ Context, User Query}

User App

HTTP, JSON

Réponse

RAG
Retrieval Augmented Generation

Chunks

PDF

Split

C 1
C 2
C 3

Embeddings

0.43	0.11	0.76	-0.5
0.43	0.11	0.76	-0.5
0.43	0.11	0.76	-0.5

Database

Query
Context

User App

Query
Response

RAG App

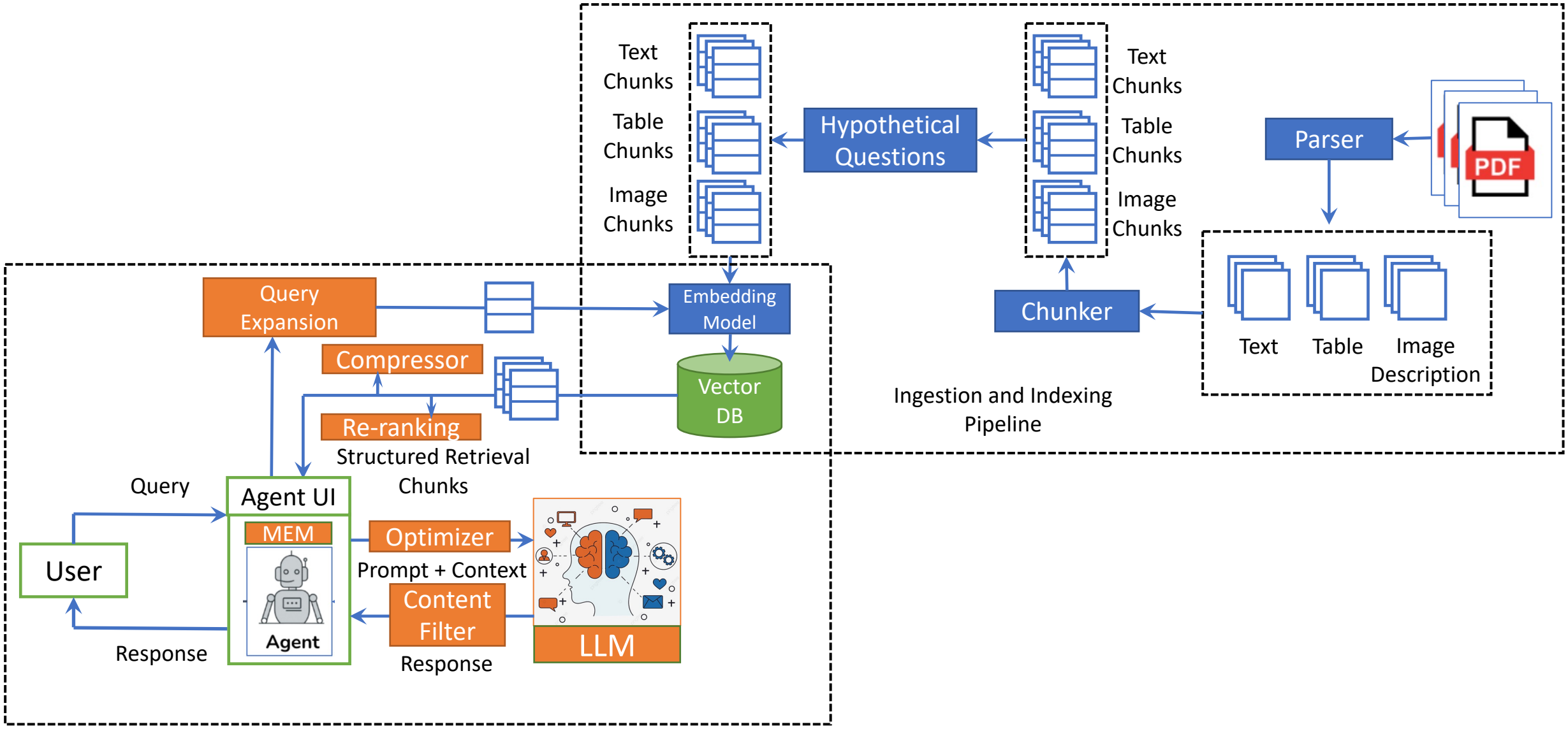
System Message + **Query** + **Context**

HTTP, JSON

Response

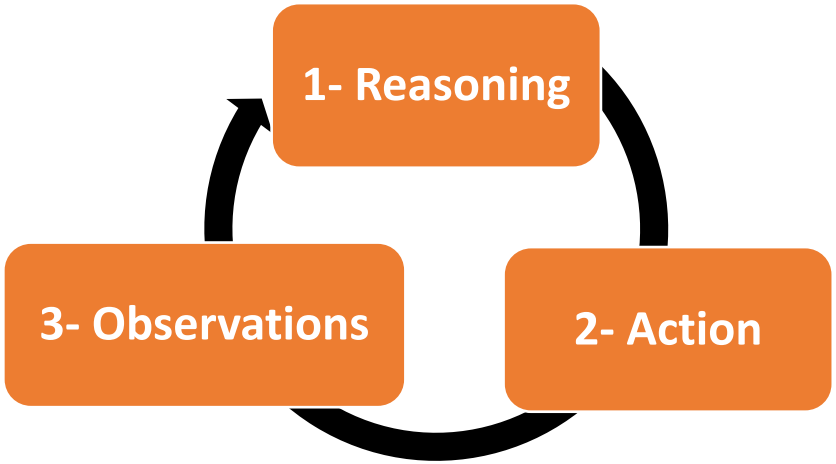
LLM

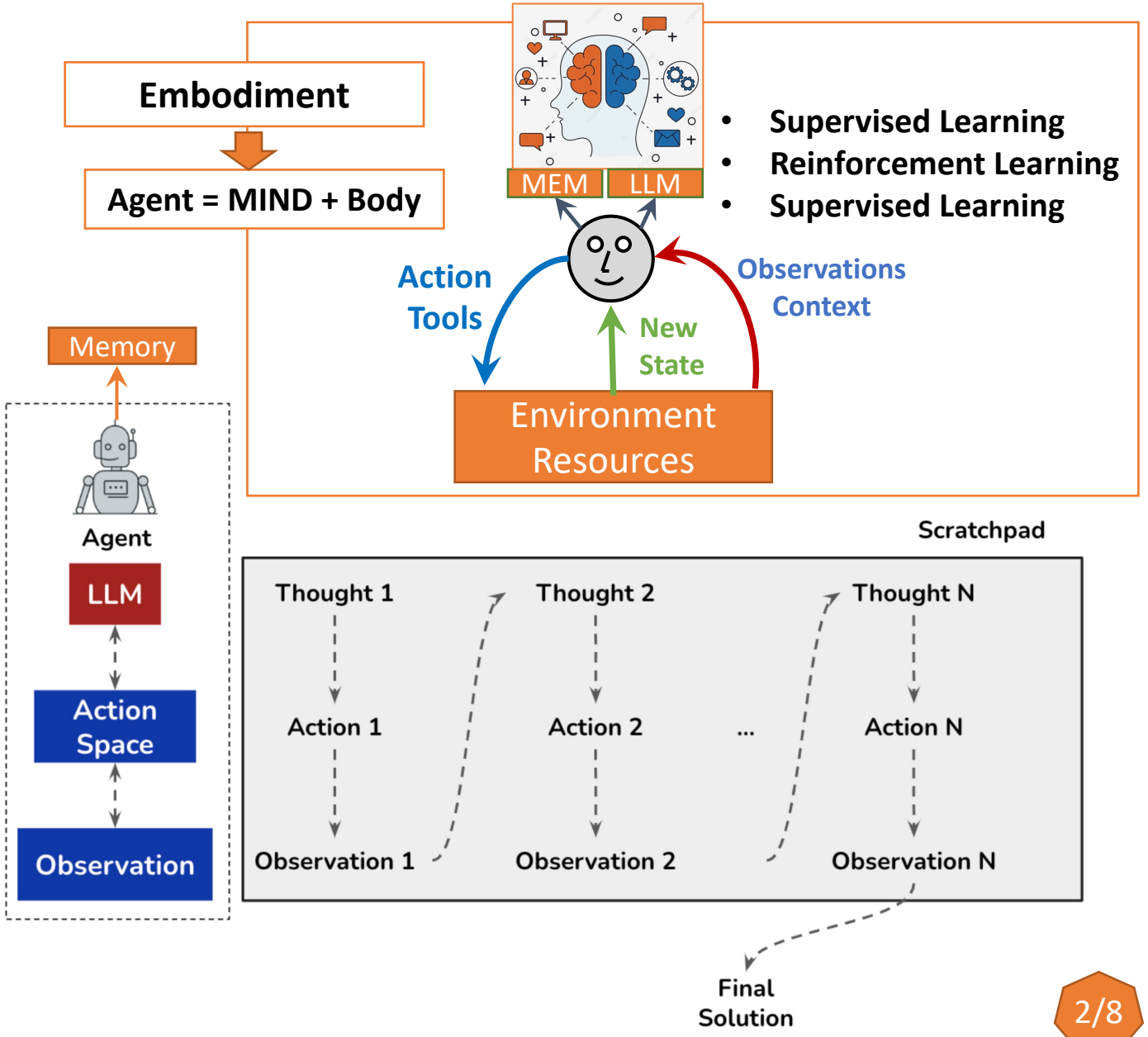
Advanced Multi Modal Agentic RAG



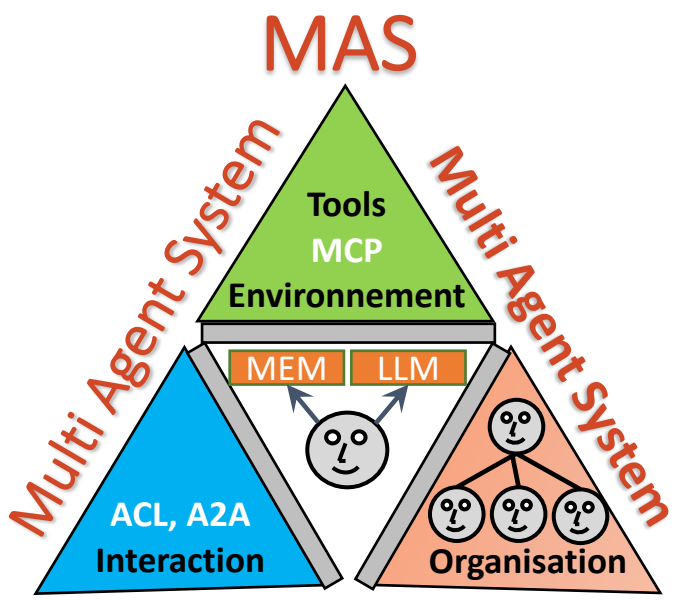
AI Agents

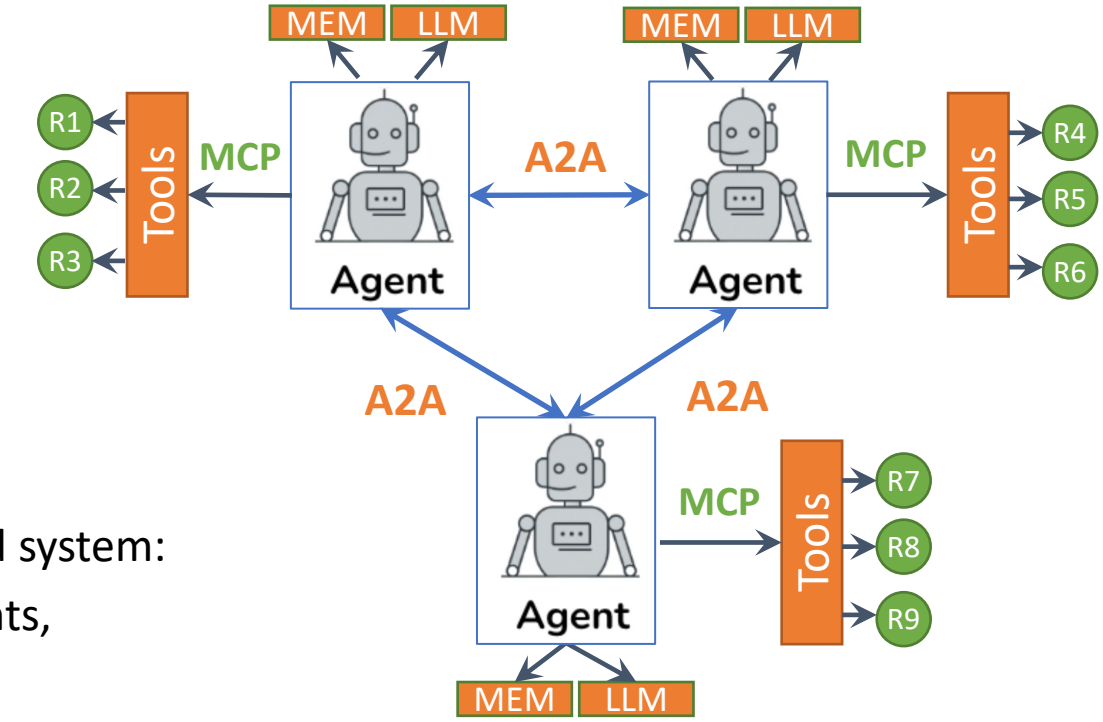
- An AI Agent, as described by the **ReaAct** Framework, is an autonomous entity with a goal that can:
  - **Reason** based on context (environmental abstraction)
  - **Act** autonomously on its environment
  - **Collect** observations about the environment





MAS And new Protocols : MCP et A2A



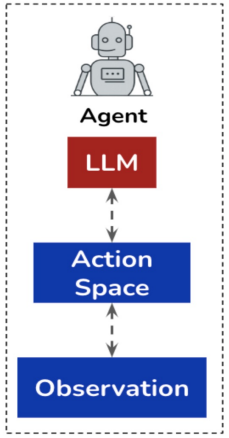


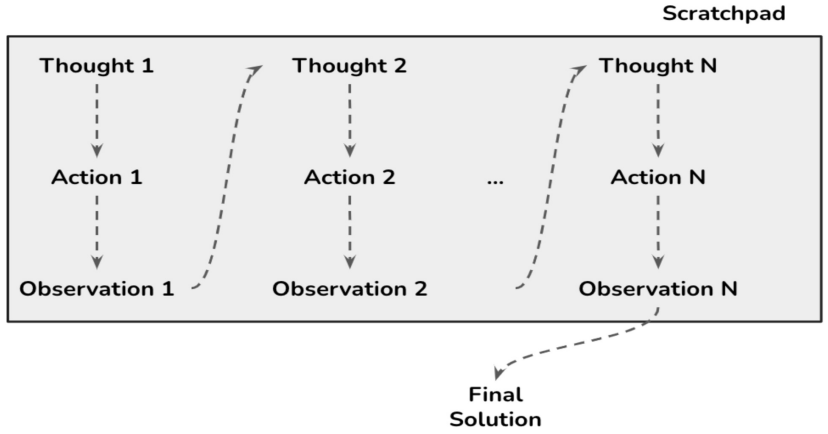
- A multi-agent system (MAS) is a distributed system:
  - Composed of a set of distributed agents,
  - Located in a specific environment
  - and Interacting according to specific organizational structures.
- An MAS solves complex problems by leveraging the collective intelligence of its component agents (DAI).

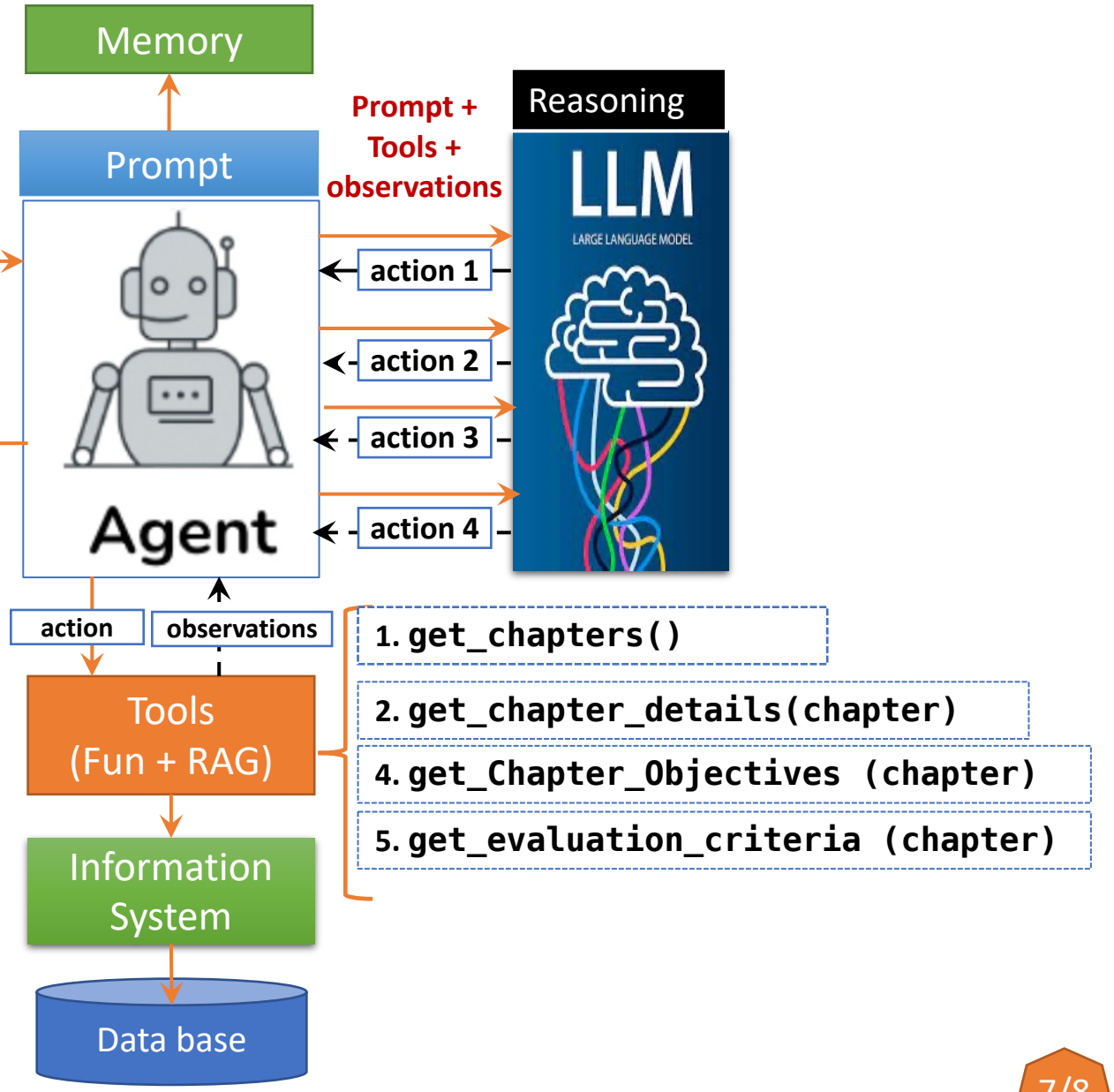
Architecture 1

Je voudrais passer une évaluation de positionnement concernant le chapitre chaîne d'énergie (fonctions alimenter, distribuer et convertir)

Q1. À partir du schéma fourni :
a. Quel composant correspond à la fonction **Alimenter** ?
b. Quel composant assure la fonction **Distribuer** ?
c. Quel composant a pour rôle de **Convertir** l'énergie ?

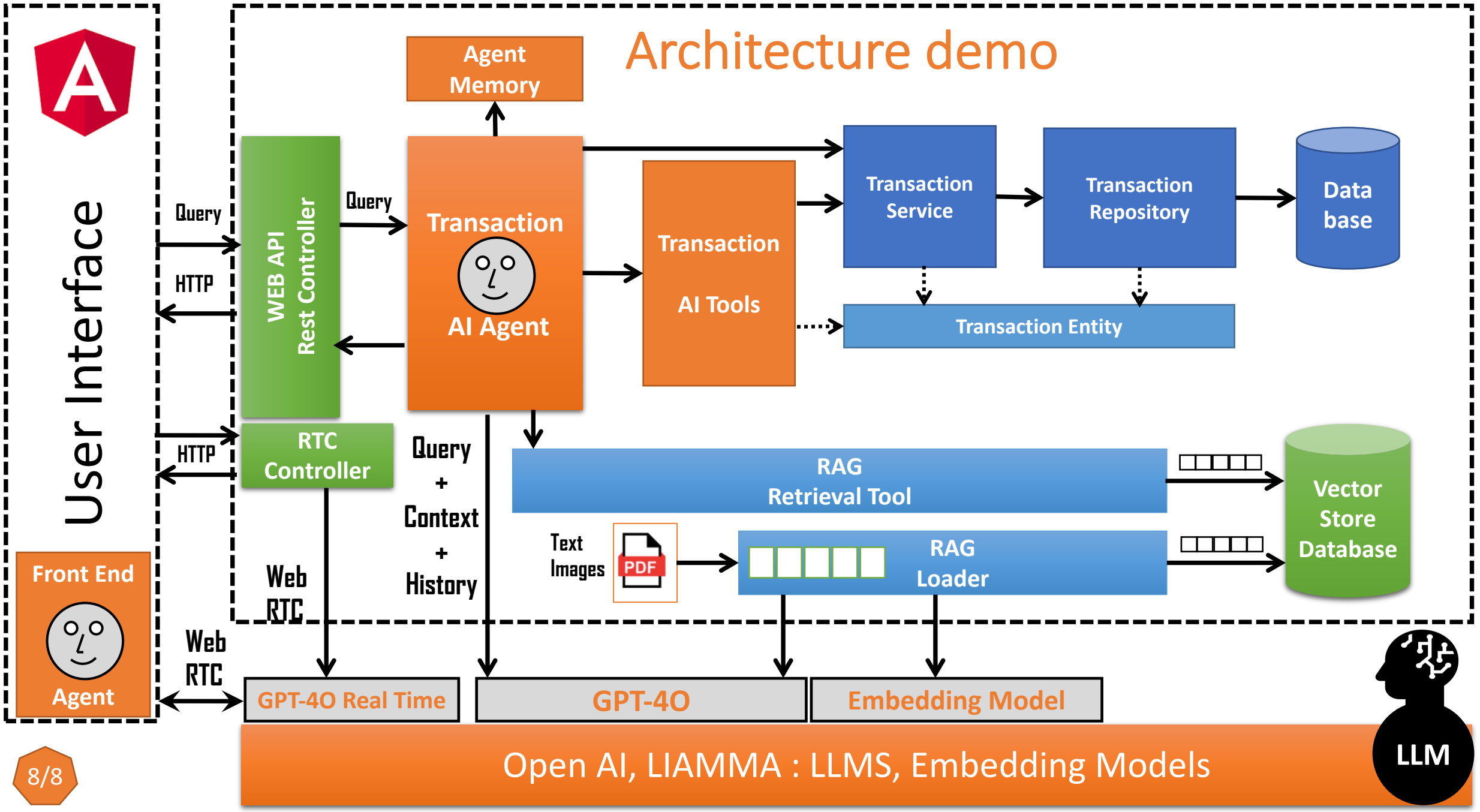






Démo1 : https://www.youtube.com/watch?v=sxlDvVBzX5A

Démo2 : https://www.youtube.com/watch?v=5aS35FFqY18


Architecture demo
User Interface
Agent Memory
WEB API Rest Controller
Query
HTTP
Query
Transaction
AI Agent
Transaction
AI Tools
Transaction Service
Transaction Repository
Data base
Transaction Entity
RTC Controller
HTTP
Query + Context + History
RAG Retrieval Tool
Vector Store Database
Text Images
PDF
RAG Loader
Front End
Agent
Web RTC
Web RTC
GPT-4O Real Time
GPT-4O
Embedding Model
Open AI, LIAMMA : LLMS, Embedding Models
LLM

AI Agents Frameworks

- **OpenAI Agent SDK** : https://openai.github.io/openai-agents-python/
- **CrewAI** : https://docs.crewai.com/introduction
- **LangChain** : https://python.langchain.com/docs/tutorials/agents/
- **AutoGen** : https://microsoft.github.io/autogen/dev//user-guide/agentchat-user-guide/tutorial/agents.html
- **Pydantic AI Agent** : https://ai.pydantic.dev/
- **Phidata** : https://docs.phidata.com/introduction
- **PraisonAiAgents** : https://docs.praison.ai/framework/praisonaiagents
- …

OpenAI Agents SDK

- The OpenAI Agents SDK aims to build agentic AI apps in a lightweight, easy-to-use package with very few abstractions.
- It's a production-ready upgrade of our previous experimentation for agents, Swarm.
- The Agents SDK has a very small set of primitives:
  - **Agents**, which are LLMs equipped with instructions and tools
  - **Handoffs**, which allow agents to delegate to other agents for specific tasks
  - **Guardrails**, which enable the inputs to agents to be validated
  - Support **Model Context Protocol (MCP)**

Installation

```
pip install openai-agents
```

Hello world example

```
from agents import Agent, Runner

agent = Agent(name="Assistant", instructions="You are a helpful assistant")

result = Runner.run_sync(agent, "Write a haiku about recursion in programming.")
print(result.final_output)

# Code within the code,
# Functions calling themselves,
# Infinite loop's dance.
```

```
export OPENAI_API_KEY=sk-...
```

DEMO : Agentic AI

Source code and resources : https://github.com/mohamedYoussfi/agentic-ai

```
> pip install uv
> uv init
> uv venv
> source .venv/bin/activate (MACOS)
> .venv\Scripts\activate.bat (Windows)
> uv add ipykernel
> uv add "openai-agents[viz]"
> uv add pydantic
> uv add python-dotenv
> uv add streamlit
```

- **pyproject.toml**

```
[project]
name = "test"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.11"
dependencies = [
    "duckduckgo-search>=8.0.1",
    "ipykernel>=6.29.5",
    "openai-agents[viz]>=0.0.14",
    "pydantic>=2.11.4",
    "python-dotenv>=1.1.0",
    "streamlit>=1.45.0",
]
```

First Agent Using OpenAI Model

```
from dotenv import load_dotenv
import os
from IPython.display import Markdown
```

✓ 0.0s

```
load_dotenv()
```

✓ 0.0s

```
api_key = os.environ.get('OPENAI_API_KEY')
if not api_key:
    raise("API KEY not available")
```

✓ 0.0s

.env

```
OPENAI_API_KEY=sk-proj-…
```

```
from agents import Agent, Runner, OpenAIChatCompletionsModel, AsyncOpenAI
```

✓ 0.5s

```
cost_medico_agent = Agent(
    name="Doctor Agent",
    model="gpt-4o",
    instructions="""
    Donne moi des conseils concernant la pathologie fournie par l'utilisateur
    """
    )
result =  await(Runner.run(cost_medico_agent,"Cardiologie"))
```

```
display(Markdown(result.final_output))
```

En ce qui concerne la cardiologie, il est important de prendre soin de la santé cardiovasculaire en suivant plusieurs conseils généraux:

1. **Alimentation équilibrée** : Adopte un régime riche en fruits, légumes, grains entiers, protéines maigres et acides gras oméga-3 (présents dans le poisson, les noix et les graines).
2. **Activité physique régulière** : Vise au moins 150 minutes d'exercice modéré par semaine, comme la marche rapide, la natation ou le vélo.
3. **Gestion du stress** : Pratiques comme le yoga, la méditation ou des exercices de respiration peuvent aider à réduire le stress.
4. **Arrêt du tabac** : Si tu fumes, chercher de l'aide pour arrêter peut drastiquement améliorer ta santé cardiaque.
5. **Contrôle de la pression artérielle et du cholestérol** : Effectue des contrôles réguliers et prends les mesures nécessaires si les niveaux sont élevés.
6. **Maintien d'un poids sain** : Un poids adéquat peut réduire le risque de maladies cardiovasculaires.
7. **Modération de la consommation d'alcool** : Limite ta consommation d'alcool si tu choisis de boire.
8. **Suivi médical** : Effectue des bilans de santé réguliers pour détecter précocement les problèmes.

Si tu as des soucis spécifiques ou un diagnostic, il est important de consulter un cardiologue pour un plan de traitement adapté à ton cas.

First Agent Using OpenAI Llamma 3.2 Model

```python
lamma_model = OpenAIChatCompletionsModel(
    model="llama3.2",
    openai_client= AsyncOpenAI(base_url="http://localhost:11434/v1")
)
```

```python
free_medico_agent = Agent(
    name="Doctor Agent",
    model=lamma_model,
    instructions="""
    Donne moi des conseils concernant la pathologie fournie par l'utilisateur
    """
    )
result =  await(Runner.run(free_medico_agent,"Cardiologie"))
```

```python
display(Markdown(result.final_output))
```

Voici quelques informations générales et conseils relatifs à la cardiologie :

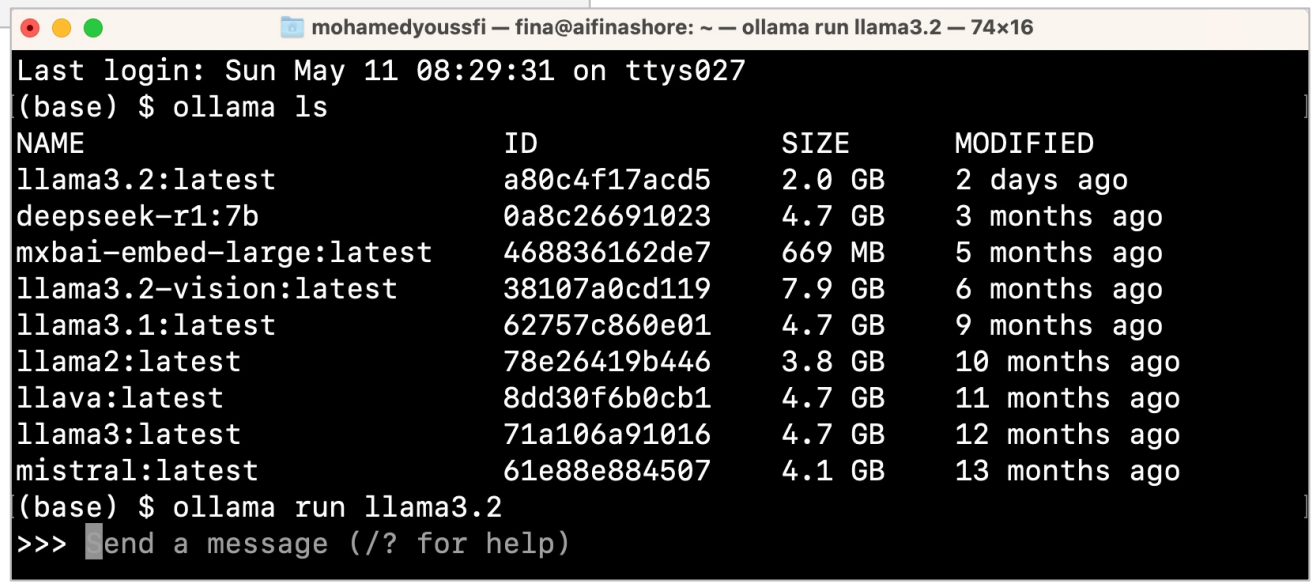
Qu'est-ce que le cœur ?

Le cœur est un organe essentiel de notre corps qui frappe sur environ 100 000 fois par jour. Il est responsable de pousser l' oxygène à travers les artères et d'amener à retourner au cœur le déchet, comme les déchets du cerveau. Le cœur est établi à la base du torso.

Les troubles cardiaques

Il existe une large gamme de maladies cardiovasculaires qui affectent millions de personnes dans le monde entier. Voici quelques exemples :

1. **Insuffisance cardiaque**: Lorsqu'elles ne fonctionnent pas avec suffisamment pour fournir les doses nécessaires à tout le corps, les organes peuvent s'empirer.
2. **Coronaropathie** : Cette maladie est due au stress des artères. Tout ce que vous avez besoin de faire, c'est que votre système d'ongulates réagisse très bien avec une pression.
3. **Aortique et pulmonaire hypertension**: le risque de stent ou d'assister à un cedran
4. **Arérodite hémorragique** : sache qu'il y a peut-être des problèmes en cas d'hémorragie
5. **Anomalies du coeur (AESC)**



Agent and Structured Output

```python
from pydantic import BaseModel
```

```python
class Recipe(BaseModel):
    title : str
    ingredients : list[str]
    cooking_time : int
    servings : int
```

```python
recipe_agent = Agent(
    name = "Recipe Agent",
    model="gpt-4o",
    instructions= """
    You are an agent for creating Recipes
    You will be given the name of the food to create
    You job is to create this food as an actual detailed recipe
    The cooking time should be in minutes
    """,
    output_type= Recipe
)
```

```python
response = await Runner.run(recipe_agent,"Tajine Marocain en Arabe")
recipe = response.final_output
```

```python
print("*"*40)
print(f"Title : {recipe.title}")
print(f"Time : {recipe.cooking_time} minutes")
print(f"Servings : {recipe.servings}")
for ing in recipe.ingredients:
    print(f"- {ing}")
```

```
****************************************
Title : طاجين مغربي
Time : 120 minutes
Servings : 4
- 1 كغ من لحم الغنم (أو الدجاج حسب الرغبة)
- 2 بصلات متوسطة الحجم، مقطعة إلى شرائح
- 3 فصوص ثوم مفرومة
- 4 ملاعق كبيرة من زيت الزيتون
- 1 ملعقة صغيرة من الزنجبيل المطحون
- 1 ملعقة صغيرة من الكركم
- 1 ملعقة صغيرة من القرفة
- ملح وفلفل حسب الرغبة
- 100 غرام من الزبيب
- 100 غرام من اللوز المقلي
- عصير نصف ليمونة
- 2 كوب من الماء
- حفنة من الكزبرة الطازجة، مفرومة للزينة
- 200 غرام من الخضروات الموسمية (مثل الجزر والبطاطس)، مقطعة
```

Agent and Tools

```
response = await Runner.run(weather_agent, "The temperature and weather in Marrakech")
response.final_output
```

```
The tool get_weather is used by the agent to get the weather of the Marrakech
The tool get_temperature is used by the agent to get the temperature of the Ma

'The weather in Marrakech is sunny, with a temperature of 50°F.'
```

```
from agents import Agent, function_tool, WebSearchTool
```

```
@function_tool
def get_weather(city : str) -> str:
    """
    Get the weather of a given city
    """
    print(f"The tool get_weather is used by the agent to get the weather of the {city}")
    return f"The weather of {city} is sunny"
```

```
response = await Runner.run(weather_agent, "Quel temps fera t-il demain à Tanger")
response.final_output
```

```
The tool get_weather is used by the agent to get the weather of the Tanger

'Demain à Tanger, le temps sera ensoleillé. ☀️'
```

```
@function_tool
def get_temperature(city : str) -> str:
    """
    Get the teperature in a given city
    """
    print(f"The tool get_temperature is used by the agent to get the temperature of the {city}")
    return f"it is 50° In {city}"
```

```
weather_agent = Agent (
    name="The Weather Agent",
    instructions="""
                As the local weather agent, you will be asked to give information
                about weather of a given city,
                Your output should include weather information asked by the user"
                """,
    tools=[get_weather, get_temperature]
)
```

Agent and Tools

```
search_agent = Agent(
    name= "Search Agent",
    instructions= """
    You are a news reporter,
    your job is to search new recent articles using internet about a given country
    """,
    tools=[WebSearchTool()]
)
```

```
search_agent = Agent(
    name= "Search Agent",
    instructions= """
    You are a news reporter,
    your job is to search new recent articles using internet about a given country
    """,
    tools=[get_news_articles]
    )
```

```
from duckduckgo_search import DDGS
from datetime import datetime
```

```
now = datetime.now().strftime("%Y-%m-%d")
now
```

```
@function_tool
def get_news_articles(topic:str):
    print(f"Running DuckDuckSearch for news in topic : {topic}")
    ddgs_api = DDGS()
    results = ddgs_api.text(f"{topic}, {now}", max_results=5)
    return results
```

```
response = await Runner.run(search_agent, "Les actualités au Maroc")
```

```
display(Markdown(response.final_output))
```

```
Running DuckDuckSearch for news in topic : المغرب
```

```
display(Markdown(response.final_output))
```

✓ 0.0s

آخر الأخبار حول المغرب

1. **حكومة بيدرو سانشيز تمنح المغرب 340 مليون يورو لبناء مشروع محطة تحلية المياه**
   - الرابط: اقرأ المزيد
   - مقتطف: حكومة بيدرو سانشيز منحت المغرب مبلغ 340 مليون يورو لتمويل مشروع محطة لتحلية المياه...
2. **التقويم والأعياد الرسمية للمغرب سنة 2025**
   - الرابط: اقرأ المزيد
   - مقتطف: تشمل العطل الرسمية في المغرب عام 2025 رأس السنة الجديدة وذكرى بيان الاستقلال...
3. **المملكة المغربية لسنة 2025 - تفاصيل العطل**

Agent and Handoffs

```python
from IPython.display import Markdown
from agents.extensions.visualization import draw_graph
```

```python
from agents import Agent
from pydantic import BaseModel
```

```python
class Tutorial(BaseModel):
    outline : str
    tutorial : str
```

```python
tutorial_response = await Runner.run(outline_builder_agent,
                                     "Boucles en Java avec des explications en Français")
tuto = tutorial_response.final_output
```

```python
tutorial_generator_agent = Agent(
    name="Tutorial Generator Agent",
    handoff_description= "Used to generate tutorial for a given outline",
    instructions="""
        Given a programming topic and an outline,
        your job is to generate code snippets for each section of the outline,
        Format the tutorial in Markdown using a mix of text explaination and code snippets examples",
        Include comments in the code snippets to further explain the code"
    """,
    output_type=Tutorial
)
```

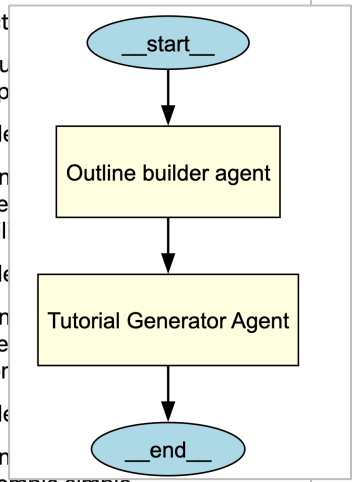
```python
display(Markdown("### Outline :"))
display(Markdown(tuto.outline))
print("*"*60)
display(Markdown("### Totorial :"))
display(Markdown(tutorial_response.final_output.tutorial))
```

```python
draw_graph(outline_builder_agent)
```

```python
outline_builder_agent = Agent(
    name= "Outline builder agent",
    instructions="""
        Given a particular programming topic, your joc is to generate a tutorial by crafting an ouline
        After making the outline, hand it to the tutorial generator agent"
    """,
    handoffs=[tutorial_generator_agent]
)
```

Outline :

1. Introduct
   - Pou
   - Typ
2. La boucle
   - Syn
   - Exe
   - Util
3. La boucle
   - Syn
   - Exe
   - Cor
4. La boucle
   - Syn
   - Exemple simple
   - Différences avec `while`



Triage Agent

```python
from agents import Agent, handoff, RunContextWrapper, function_tool
from agents.extensions.visualization import draw_graph
```

```python
@function_tool
def get_teacher_name(course : str):
    """
    Get The teacher name if course (Mathematics, History)
    """
    if(course=='Mathe
        return "Pr. H
    else:
        return "Pr. Najlaa"
```

(parameter) course: str

course (Mathematics, History)

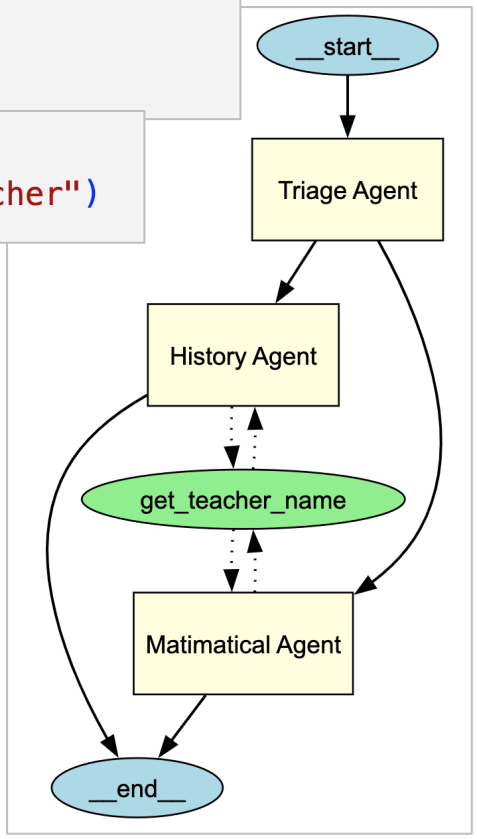
```python
triage_agent = Agent(
    name= "Triage Agent",
    instructions="""
        Answer the user question by delegating the task to the apropriate agent,
        If neither agent is relevant, give a general response
    """,
    handoffs=[history_agent, math_agent]
)
```

```python
history_agent = Agent(
    name= "History Agent",
    handoff_description= "Specialist Agent for historical questions",
    instructions="""
        You provide asistant with the historical queries
        Explain important events and context clearlly
    """,
    tools= [get_teacher_name]
)
```

```python
math_agent = Agent(
    name= "Matimatical Agent",
    handoff_description= "Specialist Agent for matimatical questions",
    instructions= """
        You provide asistant with the matimatical queries
        Explain your reasoning at each step and include examples
        """,
    tools= [get_teacher_name]
)
```

```python
trial_response = await Runner.run(triage_agent,
        "How do i multiply X by y and give me the name of teacher")
```

```python
draw_graph(triage_agent)
```



Triage Agent

```python
from agents import Agent, handoff, RunContextWrapper, function_tool
from agents.extensions.visualization import draw_graph
```

```python
@function_tool
def get_teacher_name(course : str):
    """
    Get The teacher name if course (Mathematics, History)
    """
    if(course=='Mathe
        return "Pr. H
    else:
        return "Pr. Najlaa"
```

(parameter) course: str

course (Mathematics, History)

```python
triage_agent = Agent(
    name= "Triage Agent",
    instructions="""
        Answer the user question by delegating the task to the apropriate agent,
        If neither agent is relevant, give a general response
    """,
    handoffs=[history_agent, math_agent]
)
```

```python
history_agent = Agent(
    name= "History Agent",
    handoff_description= "Specialist Agent for historical questions",
    instructions="""
        You provide asistant with the historical queries
        Explain important events and context clearlly
    """,
    tools= [get_teacher_name]
)
```

```python
math_agent = Agent(
    name= "Matimatical Agent",
    handoff_description= "Specialist Agent for matimatical questions",
    instructions= """
        You provide asistant with the matimatical queries
        Explain your reasoning at each step and include examples
        """,
    tools= [get_teacher_name]
)
```

```python
trial_response = await Runner.run(triage_agent,
        "How do i multiply X by y and give me the name of teacher")
```

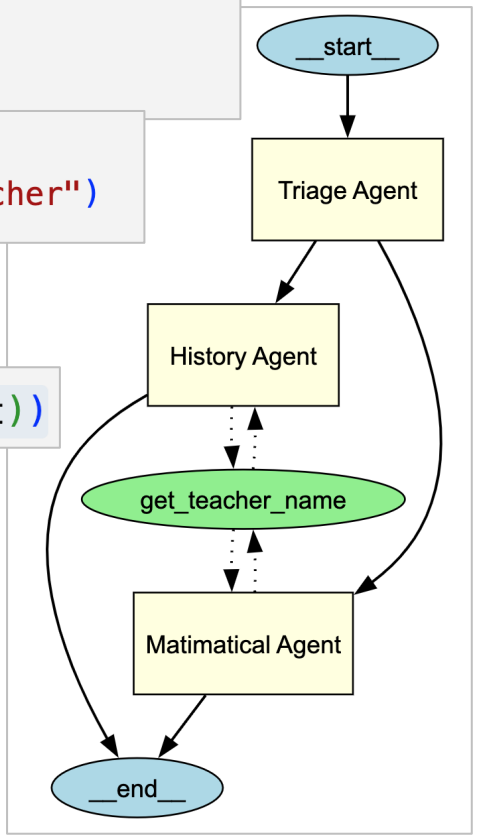
```python
draw_graph(triage_agent)
```

```python
display(Markdown(trial_response.final_output))
```

To multiply two numbers, (x) and (y), follow these steps:

1. **Understand the Multiplication:** Multiplication is a shortcut for repeated a
2. **Perform the Multiplication:**
   - If both (x) and (y) are positive integers, multiply them directly.
   - For example, if (x = 3) and (y = 4), then (3 \times 4 = 12).
3. **Use a Calculator for Complex Numbers:**
   - For large or decimal numbers, you might want to use a calculator.
4. **Special Properties:**
   - **Zero Property:** Any number multiplied by 0 is 0.
     - E.g., (5 \times 0 = 0).
   - **Identity Property:** Any number multiplied by 1 remains unchanged.
     - E.g., (7 \times 1 = 7).

The Mathematics teacher is Pr. Hassan.



Triage Agent

```
def on_history_handoff(ctx : RunContextWrapper[None]):
    print ("="*60)
    print ("History Agent is Invoqued")
    print ("="*60)

def on_math_handoff(ctx : RunContextWrapper[None]):
    print ("*"*60)
    print ("Math Agent is Invoqued")
    print ("*"*60)
```

```
triage_agent = Agent(
    name= "Triage Agent",
    instructions="""
        Answer the user question by delegating the task to the apropriate agent,
        If neither agent is relevant, give a general response
    """,
    handoffs=[
        handoff(history_agent, on_handoff=on_history_handoff),
        handoff(math_agent, on_handoff=on_math_handoff)
            ]
)
```

```
trial_response = await Runner.run(triage_agent,"HISTORY OF Rabat")
```

```
============================================================
History Agent is Invoqued
============================================================
```

```
trial_response = await Runner.run(triage_agent,"C'est quoi l'ENSET")
display(Markdown(trial_response.final_output))
```

L'ENSET (École Normale Supérieure de l'Enseignem
disciplines techniques et professionnelles. Elle est s
futurs enseignants à transmettre des connaissance
programmes peuvent inclure des études théorique

```
trial_response = await Runner.run(triage_agent,"How do i multiply X by y")
```

```
************************************************************
Math Agent is Invoqued
************************************************************
```

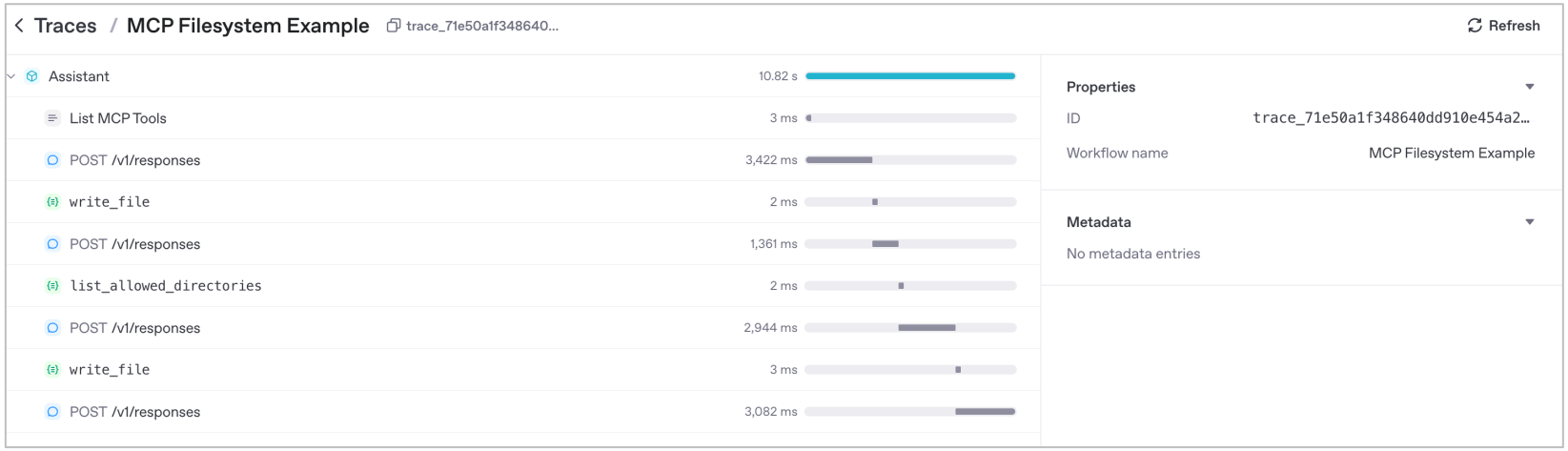
Agent Traces

https://platform.openai.com/traces

```
from agents import trace
```

```
with trace("Triage Work Flow"):
    trial_response = await Runner.run(triage_agent,"Hitoire de Rabat")
    trial_response.final_output
```

```
==========================================================
History Agent is Invoqued
==========================================================
```



Agent Streaming

```python
from openai.types.responses import ResponseTextDeltaEvent
joke_agent = Agent(
    name="My Agent",
    model='gpt-4o',
    instructions="You are a joke teller. you are giving a topic and you will tell 10 jokes about it"
)
result =  Runner.run_streamed(joke_agent,"Food")
async for event in result.stream_events():
    if event.type=='raw_response_event' and isinstance(event.data, ResponseTextDeltaEvent):
        print(event.data.delta, end="", flush=True)
```

```
Sure thing! Here are 10 food jokes for you:

1. Why did the tomato turn red?
   Because it saw the salad dressing!

2. What did the taco say to the burrito?
   "I'm nacho friend anymore!"

3. Why don't eggs tell jokes?
   They might crack up!

4. What do you call cheese that's not yours?
   Nacho cheese!
```

Agent MCP : Model Context Protocol

```
## Cretae agent_mcp.py File
## Then Exceute this application in console : > python
agent_mcp.py
import asyncio
import os
import shutil
from agents import Agent, Runner, gen_trace_id, trace
from agents.mcp import MCPServer, MCPServerStdio
from dotenv import load_dotenv

load_dotenv()

async def run(mcp_server: MCPServer):
    agent = Agent(
        name="Assistant",
        instructions="Use the tools to read the filesystem and
answer questions based on those files.",
        mcp_servers=[mcp_server],
    )

    # List the files it can read
    message = "Read the files and list them."
    print(f"Running: {message}")
    result = await Runner.run(starting_agent=agent, input=message)
    print(result.final_output)
```

```
# Ask about books
    message = "What is my #1 favorite book?"
    print(f"\n\nRunning: {message}")
    result = await Runner.run(starting_agent=agent, input=message)
    print(result.final_output)

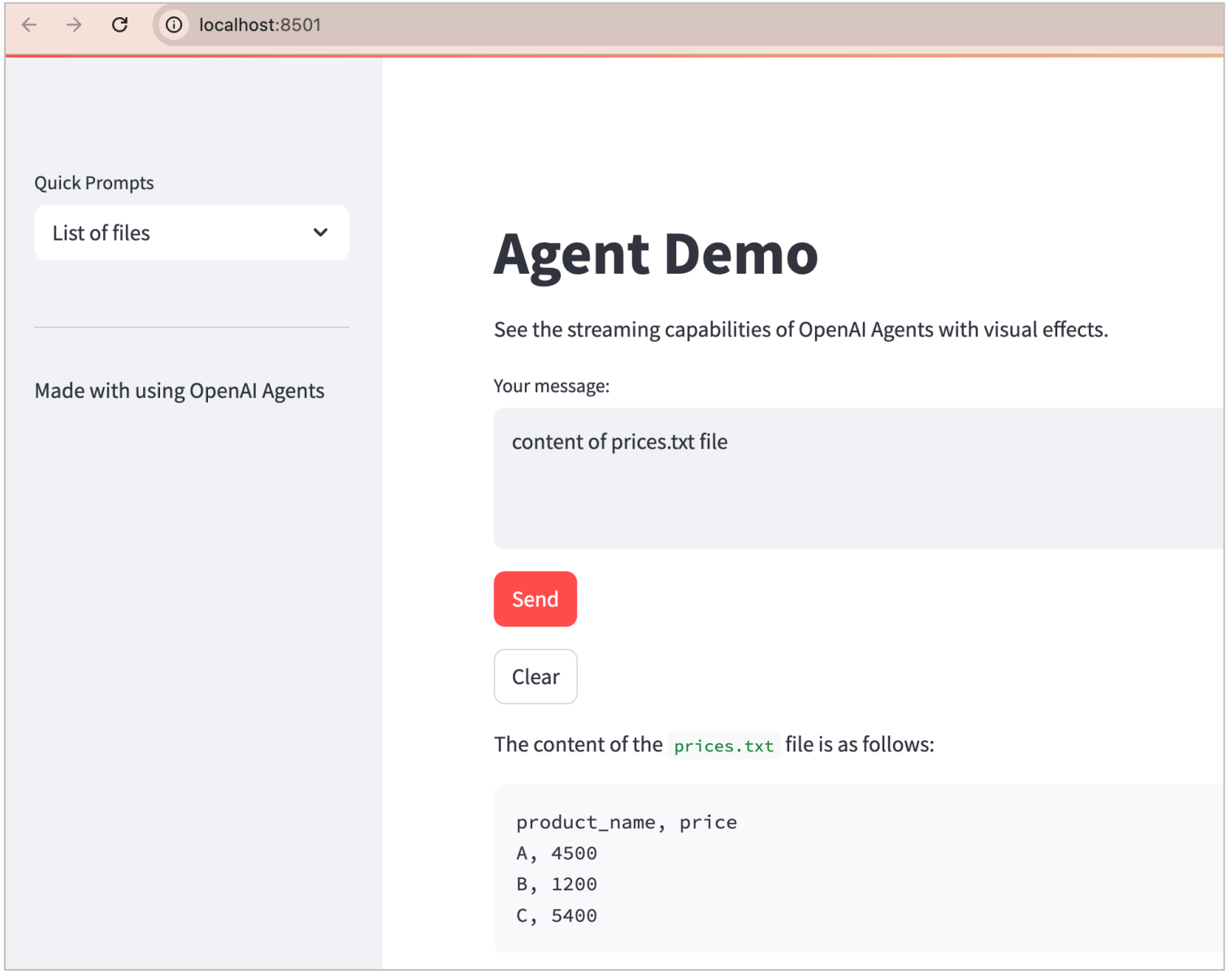
    # Ask a question that reads then reasons.
    message = "Look at my favorite songs. Suggest one new song that I might
like."
    print(f"\n\nRunning: {message}")
    result = await Runner.run(starting_agent=agent, input=message)
    print(result.final_output)

async def main():
    current_dir = os.path.dirname(os.path.abspath(__file__))
    samples_dir = os.path.join(current_dir, "sample_files")
    os.makedirs(samples_dir, exist_ok=True)
    async with MCPServerStdio(
        name="Filesystem Server, via npx",
        params={
            "command": "npx",
            "args": ["-y", "@modelcontextprotocol/server-filesystem",
samples_dir],
        },
    ) as server:
        trace_id = gen_trace_id()
        with trace(workflow_name="MCP Filesystem Example", trace_id=trace_id):
            print(
                f"View trace:
https://platform.openai.com/traces/trace?trace_id={trace_id}\n"
            )
            await run(server)

if __name__ == "__main__":
    # Let's make sure the user has npx installed
    if not shutil.which("npx"):
        raise RuntimeError(
            "npx is not installed. Please install it with `npm install -g npx`."
        )

    asyncio.run(main())
```

Agent MCP : Model Context Protocol with Streamlit UI



Agent MCP : Model Context Protocol with Streamlit UI

```
import asyncio
import streamlit as st
from openai.types.responses import ResponseTextDeltaEvent
from agents.mcp import MCPServer, MCPServerStdio, MCPServerSse
import sys
import os
from dotenv import load_dotenv
load_dotenv()
api_key = os.environ.get("OPENAI_API_KEY")
if not api_key:
    raise ("API KEY not available")
# Add the parent directory to the path so we can import the agents
module
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__fil
e__))))
from agents import Agent, Runner
st.set_page_config(
    page_title="Agent Demo",
    page_icon=".",
    layout="wide",
)
st.title("Agent Demo")
st.write("See the streaming capabilities of OpenAI Agents with visual
effects.")
```

```
with st.sidebar:
    demo_options = [
        "List of files",
        "if not created, create new file data.txt",
        "content of data.txt file",
        "Look at my favorite songs. Suggest one new song that I might like.",
    ]
    demo_prompt = st.selectbox("Quick Prompts", demo_options)
    st.markdown("---")
    st.markdown("Made with  using OpenAI Agents")
# User input area
user_input = st.text_area("Your message:", value=demo_prompt, height=100)
send_button = st.button("Send", type="primary")
# Response area with custom styling
response_container = st.container()
message_placeholder = st.empty()
```

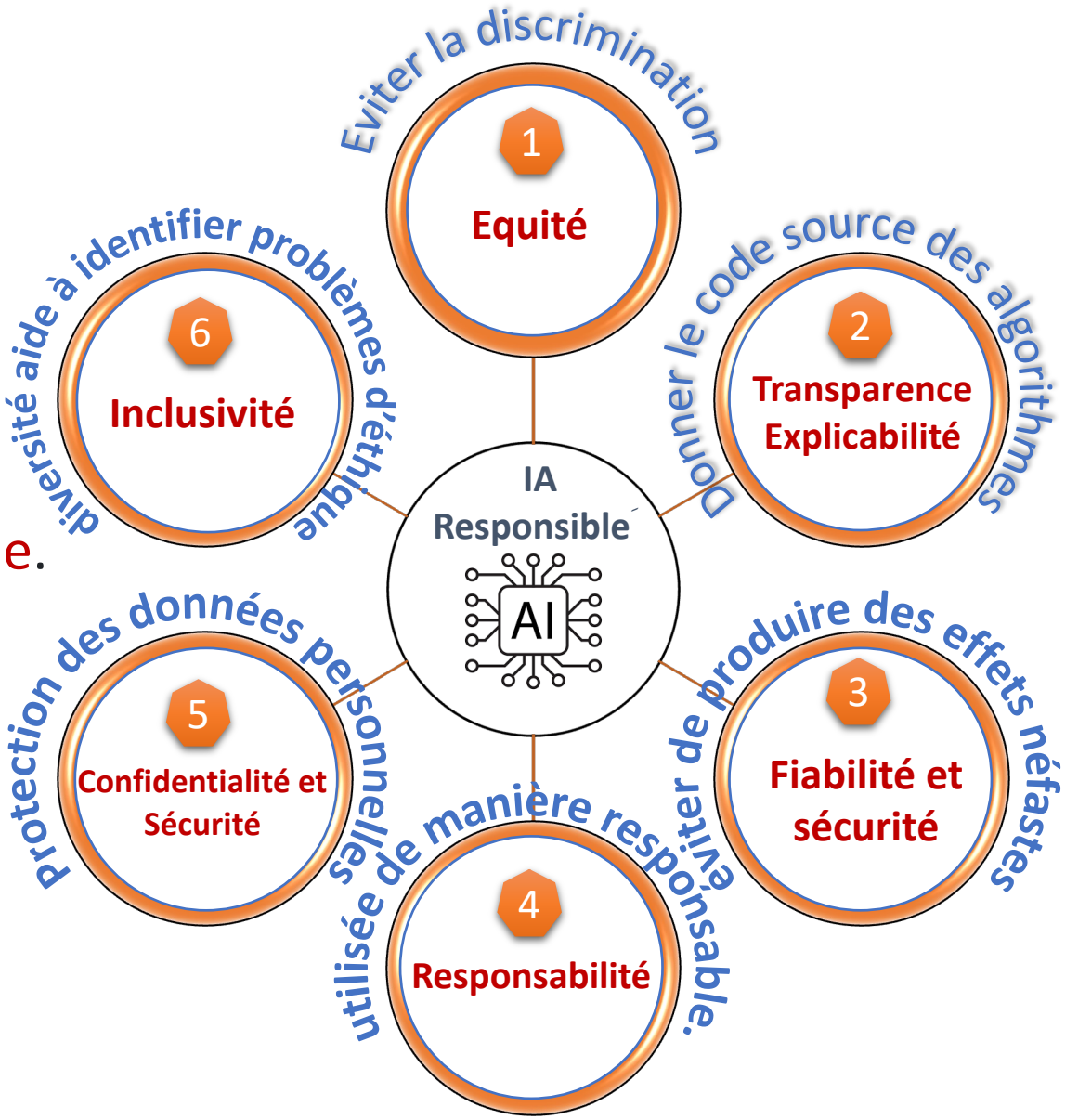
Agent MCP : Model Context Protocol with Streamlit UI

```
import asyncio
import os
import shutil
from agents import Agent, Runner, gen_trace_id, trace
from agents.mcp import MCPServer, MCPServerStdio
from dotenv import load_dotenv
load_dotenv()
async def run(mcp_server: MCPServer):
    agent = Agent(
        name="Assistant",
   instructions="answer the user question using provided tools",
        mcp_servers=[mcp_server],
    )
    if send_button and user_input:
        with response_container:
            with st.spinner("Thinking..."):
                result = await Runner.run(starting_agent=agent,
input=user_input)
                print(result.final_output)
                message_placeholder.markdown(result.final_output)
            # Add clear button
            if st.button("Clear"):
                st.experimental_rerun()
```

```
# Instructions
    if not send_button:
        with response_container:
        st.info("👆 Enter your message above and click 'Send' to see the streaming response.")
            st.markdown(
                """
            ### Tips:
            - Choose from the quick prompts or enter your own
            """)
async def main():
    current_dir = os.path.dirname(os.path.abspath(__file__))
    samples_dir = os.path.join(current_dir, "sample_files")
    os.makedirs(samples_dir, exist_ok=True)
    async with MCPServerStdio(
        name="Filesystem Server, via npx",
        params={
            "command": "npx",
            "args": ["-y", "@modelcontextprotocol/server-filesystem", samples_dir],
        },
    ) as server:
        trace_id = gen_trace_id()
        with trace(workflow_name="MCP Filesystem Example", trace_id=trace_id):
            print(
                f"View trace: https://platform.openai.com/traces/trace?trace_id={trace_id}\n"
            )
            await run(server)
if __name__ == "__main__":
    # Let's make sure the user has npx installed
    if not shutil.which("npx"):
        raise RuntimeError(
            "npx is not installed. "
        )
    asyncio.run(main())
```

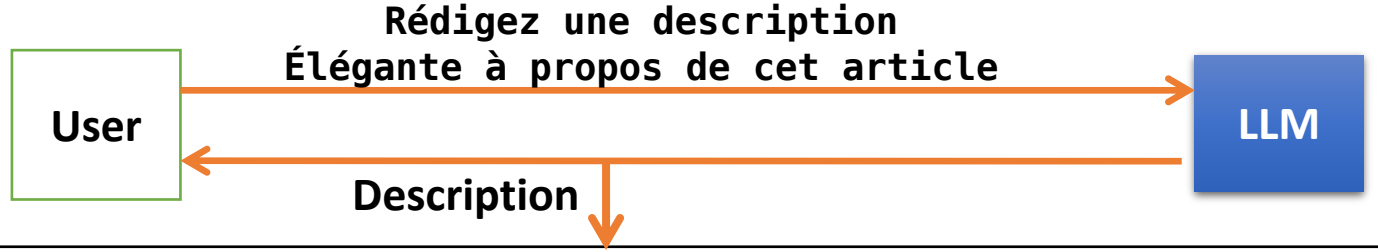
Facing a crazy world of agents
Principals of Responsible AI

L'IA
responsable est
une approche du
développement et du
déploiement de l'intelligence artificielle
sous un angle éthique et juridique.
L'objectif consiste à employer l'IA d'une
manière qui soit sûre,
fiable et éthique. Ainsi,
son utilisation gagnerait en
transparence et contribuerait à réduire les
problèmes tels que les biais (التحيزات).



Instructional prompt (Prompt pédagogique)

- Les invites pédagogiques sont un partenaire précieux dans les tâches créatives où les idées initiales sont difficiles à trouver.
- Par exemple, imaginez que vous soyez chargé de proposer une description qui figurera sur la page produit du [Clavier de jeu Razer Ornata V3 X] sur Amazon.
- Voici une invite qui accomplit cette tâche ; varier la température générera une grande variété de descriptions de produits facilitant le processus créatif.




```
instuction_prompt = """

Vous êtes marketeur pour la société de jeux Razer.
Vous trouverez ci-dessous les métadonnées sur le clavier de jeu Razer Ornata V3 X :
- Marque : Razer
- Série : Ornata V3 X
- Numéro de modèle de l'article : RZ03-04470200-R3U1.
- Plateforme matérielle : PC
- Système d'exploitation : Microsoft Windows
- Poids de l'article : 2,97 livres
- Dimensions du produit : 17,46 x 5,68 x 1,23 pouces
- Dimensions de l'article LxlxH : 17,46 x 5,68 x 1,23 pouces
- Couleur : noir classique.
- Fabricant : Razer
- Langue: français
- ASIN : B09X6GJ691
- Caractéristiques spéciales : touches à profil bas, résistantes aux éclaboussures,
repose-poignet ergonomique, éclairage Chroma RVB, interrupteurs à membrane
silencieux, options de routage des câbles

Avec ces informations, rédigez une description élégante « À propos de cet article »
qui sera utilisée sur sa page produit Amazon.

Utilisez des puces pour délimiter les principales fonctionnalités mentionnées dans
la description.
"""
```

Reasoning Prompt with ReAct Framework

- Une meilleure méthode pour utiliser les capacités de raisonnement de GPT consiste à utiliser le Framework **ReAct** (**Reasoning and Action**).
- Avec ce Framework, nous considérons GPT comme un agent intelligent et nous codifions explicitement les actions disponibles pour le modèle.
- Cela incite le modèle à peser sur les alternatives disponibles avant d'agir.

```
prompt = """
ABC Ecommerce …….
1. Diversifier les fournisseurs et les produits.
2. Améliorer la visibilité de la chaîne d'approvisionnement et la collaboration .
3. Explorez les capacités de fabrication internes ou les opportunités de
partenariat.

Recommander un plan d'action à l'entreprise et créer un plan étape par
étape que l'entreprise devra suivre dans cette situation.
Présentez votre réponse sous le format suivant :
{
  "question": <la question d'entrée à laquelle il faut répondre>,
  "reflection": <réfléchissez à vos options et au plan d'action pour
chaque option>,
  "action": <l'action à entreprendre parmi les trois options
présentées">,
  "raison": <le raisonnement derrière votre réponse>
  "étapes" : <plan étape par étape mettant en œuvre l'action>
}
"""
```

{
"question": "Quel plan d'action ABC Ecommerce devrait-il prendre pour surmonter le problème de pénurie de fournisseurs sur sa principale gamme de produits, les smartphones ?",
"reflection ": "En considérant les options disponibles, la diversification des fournisseurs pourrait contribuer à réduire la dépendance à l'égard d'un fournisseur unique et réduire le risque de pénurie, mais cela pourrait également accroître la complexité opérationnelle. Améliorer la visibilité et la collaboration de la chaîne d'approvisionnement pourrait aider à mieux planifier et gérer les stocks, mais cela pourrait ne pas résoudre le problème de pénurie d'approvisionnement. Explorer les opportunités de fabrication en interne ou de partenariat. pourrait également être une option, mais cela pourrait nécessiter des investissements et du temps importants.",
"action": "Diversifier les fournisseurs et les produits.",
"raison": "La diversification des fournisseurs contribuerait à atténuer le risque de pénurie de fournisseurs. Dans le cas où un fournisseur est confrontée à des problèmes, l'entreprise ne serait pas entièrement dépendante de ce fournisseur et pourrait obtenir le même produit auprès d'autres fournisseurs. Cela lui donnerait également un meilleur pouvoir de négociation. La diversification des produits contribuerait à réduire la dépendance à l'égard d'une seule gamme de produits et pourrait ouvrir de nouvelles sources de revenus..
"étapes": [
"Étape 1 : Réaliser une étude de marché pour identifier de nouveaux fournisseurs potentiels capables de répondre aux exigences de qualité et de quantité de l'entreprise.",
"Étape 2 : Contacter les fournisseurs potentiels et négocier les conditions.",
"Étape 3 : Changer progressivement certaines des commandes aux nouveaux fournisseurs pour tester leur fiabilité.",
"Étape 4 : Réaliser une étude de marché pour identifier les nouveaux produits potentiels qui sont en demande et s'aligner sur le portefeuille de produits existant de l'entreprise.",
"Étape 5 : Ajouter les nouveaux produits au portefeuille de l'entreprise et les promouvoir auprès des clients.",
"Étape 6 : Surveiller les performances des nouveaux fournisseurs et produits et ajuster la stratégie si nécessaire."] }

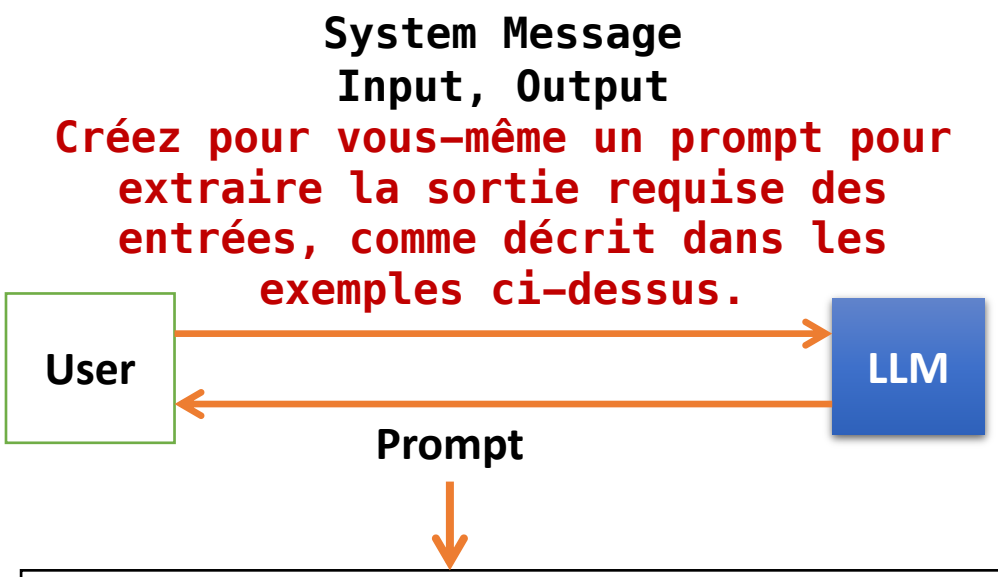
Induction Prompt

- Une approche pour générer des prompts pour un modèle de langage consiste à :
  - Présenter des paires d'entrées-sorties
  - et à inviter le modèle à générer une prompt qui obtient la sortie donnée à partir des entrées fournies.
- Cette technique peut servir de base d'idées d'incitation, contribuant ainsi à relancer le processus de génération de prompts.

```
Vous êtes un assistant qui aide l'équipe marketing d'une entreprise
d'électronique à mieux comprendre les avis de ses clients.
Les exemples d'entrées-sorties suivants ont été collectés à titre de
suggestions pour que vous puissiez en tirer des leçons.
- Entrée : Gris commandé qui annonce un éclairage vert, quand on opte pour
une esthétique cheap, c'est bouleversant. La -souris fonctionne bien.
- Sortie : Souris
- Entrée : j'en ai acheté un pour les jeux sur PC. J'ai adoré, puis j'en ai
acheté une autre pour le travail. Cette souris n'est pas à la hauteur des
souris haut de gamme comme la série Logitech MX Master, mais à 1/5-/8ème du
prix, je ne m'attendais pas à ce niveau de qualité. Il fonctionne bien, la
molette de la souris semble lourde, les boutons latéraux sont bien placés
avec différentes textures pour que vous puissiez les distinguer. La souris
semble plutôt plastique et bon marché, mais pour le prix, c'est à peu près ce
à quoi je m'attendais. J'aime une souris filaire pour éviter que le
pointeur/le jeu ne saute à cause de la latence. Fil long également, ce qui
minimise les problèmes d'accrochage. Très bon rapport qualité/prix dans
l'ensemble.
- Sortie : souris, Logitech MX Master, boutons DPI, molette de la souris, fil

Créez pour vous-même un prompt pour extraire la sortie requise des entrées,
comme décrit dans les exemples ci-dessus.

Créez l'invite pour qu'elle contienne des exemples généralisés à partir de
ceux présentés ci-dessus.
N'oubliez pas que l'invite doit contenir des instructions que vous pouvez
comprendre et générer le résultat attendu compte tenu de l'entrée.
"""
```



Prompt:
Lisez attentivement l'avis du client et identifiez le principal produit électronique en question. Notez également toutes les caractéristiques ou composants spécifiques du produit mentionnés, ainsi que tout autre produit électronique mentionné à des fins de comparaison. Le résultat doit être une liste de ces éléments.

Paraphrasing Prompt

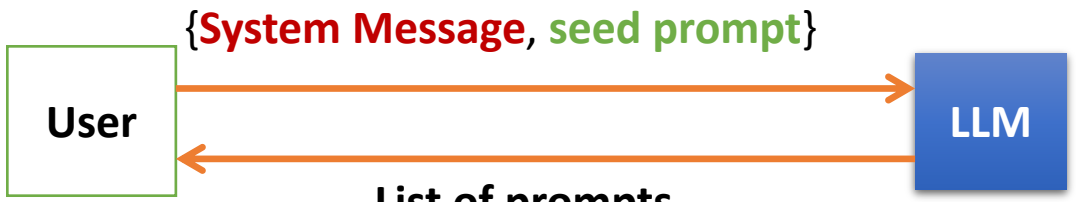
- Le Prompte de paraphrase est une technique qui consiste à générer un ensemble de promptes à l'aide d'un LLM à partir d'un Prompt de départ,
- Puis à évaluer les performances de chaque prompte sur un ensemble de test.
- Choisir le prompt le plus performant

```
paraphrase_prompt = """

Un promprt de départ vous sera présentée,
délimitée par trois backticks, c'est-à-dire
```.
Ce prompt de départ sera présentée à un grand
modèle de langage qui génère une sortie
requise spécifique.

Veuillez générer 5 variantes du prompt de
départ en gardant intacte l'intention de
l'invite de départ.

```
Veuillez analyser les avis clients suivants
et identifier les produits et fonctionnalités
qui sont mentionnés.
Fournissez une liste de produits et de
fonctionnalités mentionnés dans chaque avis.
```
"""
```




```
``` Analyzez les commentaires des clients fournis ci-dessous et identifiez les produits et
les fonctionnalités respectives qui sont discutés. Compilez une liste de ces produits et de
leurs fonctionnalités correspondantes mentionnées dans chaque avis. ```

``` Examinez les témoignages de clients fournis et déterminez les produits spécifiques ainsi
que leurs caractéristiques auxquels il est fait référence. Créez une liste détaillant le
produit et ses fonctionnalités associées mentionnées dans chaque avis. ```

``` Veuillez examiner les évaluations des clients ci-dessous et identifier les produits
explicites et leurs fonctionnalités qui sont en discussion. Créez une liste des produits et
fonctionnalités mentionnés dans chaque avis individuel. ```

``` Pouvez-vous analyser les retours clients suivants et reconnaître les produits et les
caractéristiques sur lesquels ils mettent l'accent ? Établissez une liste comprenant les
produits et les fonctionnalités mentionnés dans chacun des avis. ```

``` Inspectez les évaluations des clients qui s'ensuivent et discernez les produits et leurs
attributs qui sont mis en avant. Compilez une liste détaillant les produits et les
fonctionnalités respectives mentionnées dans chaque avis. ```
```

CoT Prompt : Chain-of-Thought Prompting :

Chain-of-Thought Prompting : CoT Prompt

- Pour l'invite CoT, nous ajoutons des instructions détaillées étape par étape au message système demandant au modèle de réfléchir attentivement avant de décider du résultat à générer
- En dehors de cet ajout, il n'y a aucun autre changement par rapport à prompt standard

```
cot_system_message = """
Classify the sentiment of movie reviews presented
in the input as 'positive' or 'negative'.
Movie reviews will be delimited by triple backticks
in the input.
Answer only 'positive' or 'negative'. Do not
explain your answer.

Instructions:
1. Carefully read the text of the review and think
through the options for sentiment provided
2. Consider the overall sentiment of the review and
estimate the probability of the review being
positive

To reiterate, your answer should strictly only
contain the label: positive or negative.
"""
```

Standard Prompt

- Q : Ahmed has 5 tennis balls, he buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls doas he have now?
- A : **The answer is 11**
- Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more. How many apples do they have?

The cafeteria now has 9 apples.

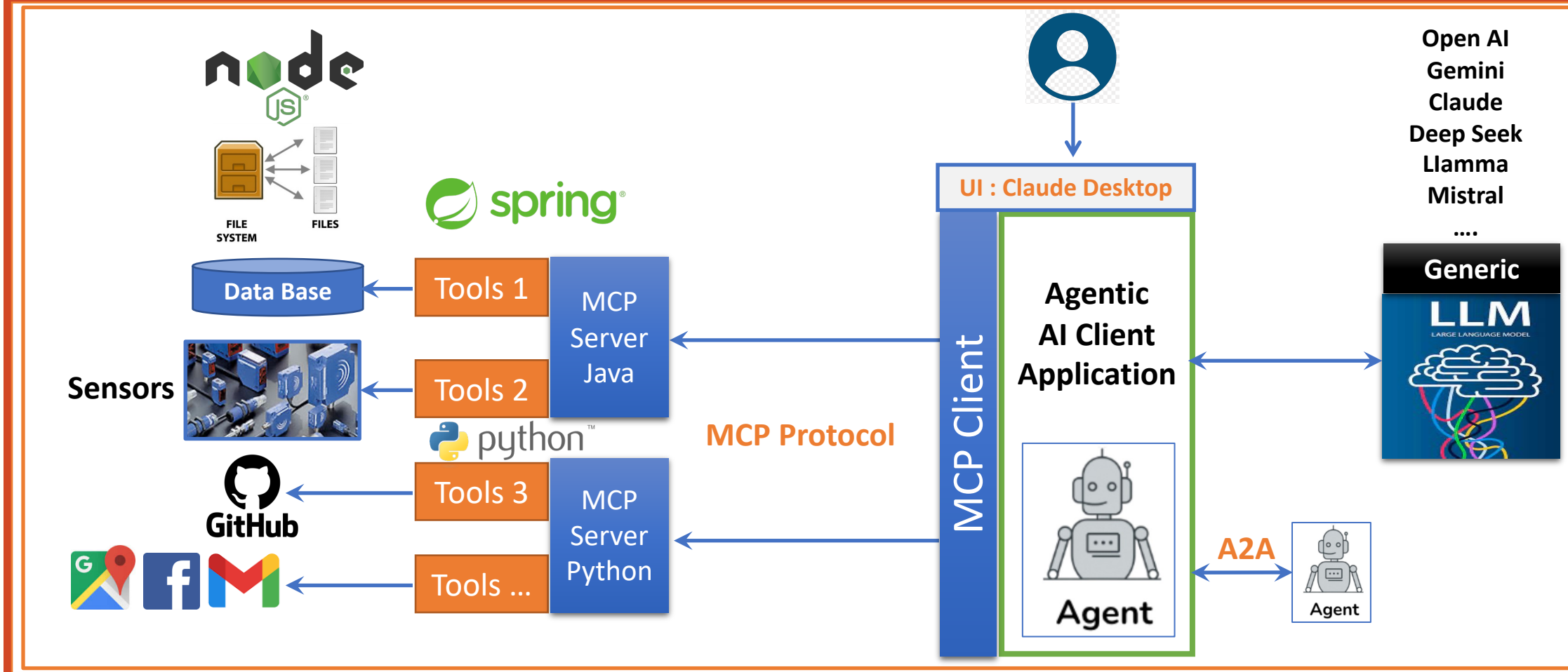
CoT Prompt

- Q : Ahmed has 5 tennis balls, he buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls doas he have now?
- A : **Ahmed started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5+6=11. The answer is 11**
- Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more. How many apples do they have?

The cafeteria started with 23 apples. They used 20, so they had 23–20=3 apples left. Then they bought 6 more, so they now have 3+6=9 apples.

Model Context Protocol (MCP)

Mohamed Youssfi, Enseignant Chercheur, ENSET Mohammedia, Directeur Laboratoire Informatique, Intelligence Artificielle et Cyber Sécurité



Example of Agent Implementation with Phidata Framework

```python
from phi.agent import Agent
from phi.model.openai import OpenAIChat
from phi.tools.duckduckgo import DuckDuckGo
import json
```

```python
def get_company_infos(company: str) -> str:
    """Get The number Information about the company : Name, Turnover, number of employees"""
    return json.dumps(
        {"name": company, "Turnover": 36000000, "number_of_employees": 9000}
    )
```

```python
def get_last_stock_market(company: str) -> str:
    """Get the stock market of the company in the last 5 days"""
    return json.dumps([400, 5450, 500, 600, 700])
```

```python
my_agent = Agent(
    name="Financial analyst",
    model=OpenAIChat(id="gpt-4o"),
    tools=[DuckDuckGo(), get_company_infos, get_last_stock_market],
    instructions=[
        "Effectuer une analyse financière en se basant sur les information de
l'entreprise et la valeurs des actions en bourse pendant les 5 derniers jours"
    ],
    show_tool_calls=True,
    markdown=True,
    structured_outputs=True,
)
```

```python
my_agent.print_response(
    "Donne moi une analyse financière de l'entreprise Maroc Telecom pour l'année 2024",
    stream=True,
)
```

Example of Agent Implementation with Phidata Framework

Message

Donne moi une analyse financière de l'entreprise Maroc Telecom pour l'année 2024

Response (21.2s)

Running:

- get_company_infos(company=Maroc Telecom)
- get_last_stock_market(company=Maroc Telecom)

Pour réaliser une analyse financière de l'entreprise Maroc Telecom pour l'année 2024, nous devons examiner plusieurs aspects, notamment les informations sur la société et la performance de ses actions en bourse. Voici une récapitulation des données disponibles :

Informations sur Maroc Telecom :

- **Nom de l'entreprise :** Maroc Telecom
- **Chiffre d'affaires :** 36 000 000 (36 millions de dollars ou MAD, le montant exact n'étant pas précisé)
- **Nombre d'employés :** 9 000

Performance des actions en bourse (sur les 5 derniers jours) :

1. 1er jour : 400
2. 2ème jour : 5 450
3. 3ème jour : 500
4. 4ème jour : 600
5. 5ème jour : 700

Analyse Financière :

1. **Chiffre d'Affaires :**
   - Maroc Telecom a généré un chiffre d'affaires de 36 millions, ce qui donne une première indication sur la taille et la capacité de l'entreprise à générer des revenus. Ceci est important pour évaluer sa position sur le marché et sa stabilité financière.
2. **Nombre d'Employés :**
   - Avec 9 000 employés, Maroc Telecom semble avoir une structure d'entreprise substantielle. Cela pourrait indiquer une opération étendue en termes de portée géographique et de gamme de services.
3. **Performance des Actions :**
   - La variation des actions sur les cinq derniers jours montre une volatilité importante, avec une valeur très élevée le deuxième jour (5 450) par rapport aux autres jours. Cela pourrait être dû à des événements temporaires ou à des fluctuations du marché.
   - La restitution à une plage de valeurs entre 400 et 700 après cette hausse soudaine nécessite une analyse plus approfondie pour comprendre si cela représente une correction normale du marché ou d'autres facteurs.

Conclusion :

- **Stabilité Financière :** Maroc Telecom semble bien établie avec un chiffre d'affaires et un effectif importants, ce qui lui permet de maintenir une opération relativement stable.
- **Volatilité du Marché :** L'énorme pic observé dans la valeur des actions du deuxième jour pourrait être exceptionnel et pourrait nécessiter une enquête plus poussée pour en déterminer la cause. Il est essentiel d'examiner les nouvelles récentes ou les publications financières pour comprendre cette

Example of Agent Implementation with Phidata Framework

```python
from phi.agent import Agent
from phi.model.openai import OpenAIChat
from phi.tools.duckduckgo import DuckDuckGo
import json
```

```python
def get_company_infos(company: str) -> str:
    """Get The number Information about the company : Name, Turnover, number of employees"""
    return json.dumps(
        {"name": company, "Turnover": 91277000000, "number_of_employees": 20000}
    )
```

```python
my_agent = Agent(
    name="Web Agent",
    model=OpenAIChat(id="gpt-4o"),
    tools=[DuckDuckGo(), get_company_infos],
    instructions=["Always include sources"],
    show_tool_calls=True,
    markdown=True,
)
```

```python
my_agent.print_response(
    "What is the number of employees and the turnover of OCP",
stream=True
)
```

Message

What is the number of employees and the turnover of OCP

Response (3.8s)

Running:

- get_company_infos(company=OCP)

OCP has a turnover of approximately 91.28 billion and employs around 20,000 people.

AI Agents Frameworks

```
my_agent.print_response(
    "C'est qui Mohamed YOUSSFI de l'ENSET",
    stream=True,
)
```

Message

C'est qui Mohamed YOUSSFI de l'ENSET

Response (8.0s)

Pour obtenir des informations précises et à jour sur Mohamed Youssfi de l'ENSET, utilisons DuckDuckGo pour une recherche en ligne. Je vais chercher des détails à son sujet. Running:

- duckduckgo_search(query=Mohamed YOUSSFI ENSET, max_results=5)

Mohamed Youssfi est un professeur et chercheur en informatique à l'Université Hassan II de Casablanca, plus précisément à l'ENSET (École Normale Supérieure de l'Enseignement Technique). Voici quelques détails le concernant :

- **Profil Académique:** Il est reconnu pour ses recherches en informatique, notamment dans les domaines du calcul parallèle et distribué, le Grid Computing et les systèmes Middleware source.
- **Publications:** Il est un auteur consulté et a publié de nombreux travaux dans le domaine, comptant plus de 190 publications source.
- **Activités Pédagogiques:** Il s'est impliqué dans l'enseignement de l'utilisation des frameworks de machines et d'apprentissage profond pour les applications mobiles et embarquées source.

Pour plus d'informations sur ses publications et son parcours académique, vous pouvez visiter ses profils sur ResearchGate et Google Scholar. Additionally, his LinkedIn profile can give insights into his professional experience source.

AI Agents Frameworks

```python
from phi.agent import Agent
from phi.model.openai import OpenAIChat
from phi.tools.duckduckgo import DuckDuckGo
from pydantic import BaseModel, Field
import json
```

```python
my_agent = Agent(
    name="Web Agent",
    model=OpenAIChat(id="gpt-4o"),
    tools=[DuckDuckGo(), get_company_infos],
    instructions=["Always include sources"],
    show_tool_calls=True,
    markdown=True,
    response_model=CompanyInfos,
    structured_outputs=True,
)
```

```python
class CompanyInfos(BaseModel):
    company: str = Field(description="The name of the campany")
    number_of_employees: int = Field(description="The number of employees")
    turnover: float = Field(description="The turnover of the company")
```

```python
def get_company_infos(company: str) -> str:
    """Get The number Information about the company : Name, Turnover, number of employees"""
    return json.dumps(
        {"name": company, "Turnover": 91277000000, "number_of_employees": 20000}
    )
```

```python
my_agent.print_response("What is the number of employees and the turnover of OCP", stream=True)
res = my_agent.run("What is the number of employees and the turnover of OCP", stream=True)
print(res.content)
```

Message

What is the number of employees and the turnover of OCP

Response (4.1s)


```
{
  "company": "OCP",
  "number_of_employees": 20000,
  "turnover": 91277000000.0
}
```


```
company='OCP' number_of_employees=20000 turnover=91277000000.0
```

```python
industrial_tracker_agent = Agent(
    name = "Industrial Tracker Agent",
    instructions = """
    Vous êtes un assistant complet qui permet de surveiller le processus industriel parmi les dispositifs fournis dans le contexte
    L'objectif est d'indiquer quels sont les dispositifs qui nécessitent une maintenance en fonction de la différence entre
    - La durée de vie en jours du device
    - Le nombre de jours après la dernière maintenance du device
    """,
    model = "gpt-4o",
    tools = [get_list_of_devices, get_lifespan_of_device, get_last_maintenance_days],
    output_type = List[Device],
    model_settings = ModelSettings(temperature=0),
)
```

```python
query = "Donnez-moi les deux dispositifs mécaniques de la partie motrice de mon processus industriel qui nécessitent une maintenance immédiate?"
result = Runner.run_sync(industrial_tracker_agent, query)
devices = result.final_output
for device in devices:
    print("--" * 50)
    print(f"\nName : {device.name}")
    print(f"\nLifespan duration : {device. maintenance_days_left}")
    print("Recommandations : ")
    for i, recommandation in enumerate(device.recommandations, 1):
        print(f"{i}: {recommandation}")
```

```python
from agents import Agent, Runner, function_tool, ModelSettings
from dotenv import load_dotenv
import os
import asyncio
from pydantic import BaseModel, Field
from typing import List
load_dotenv()
```

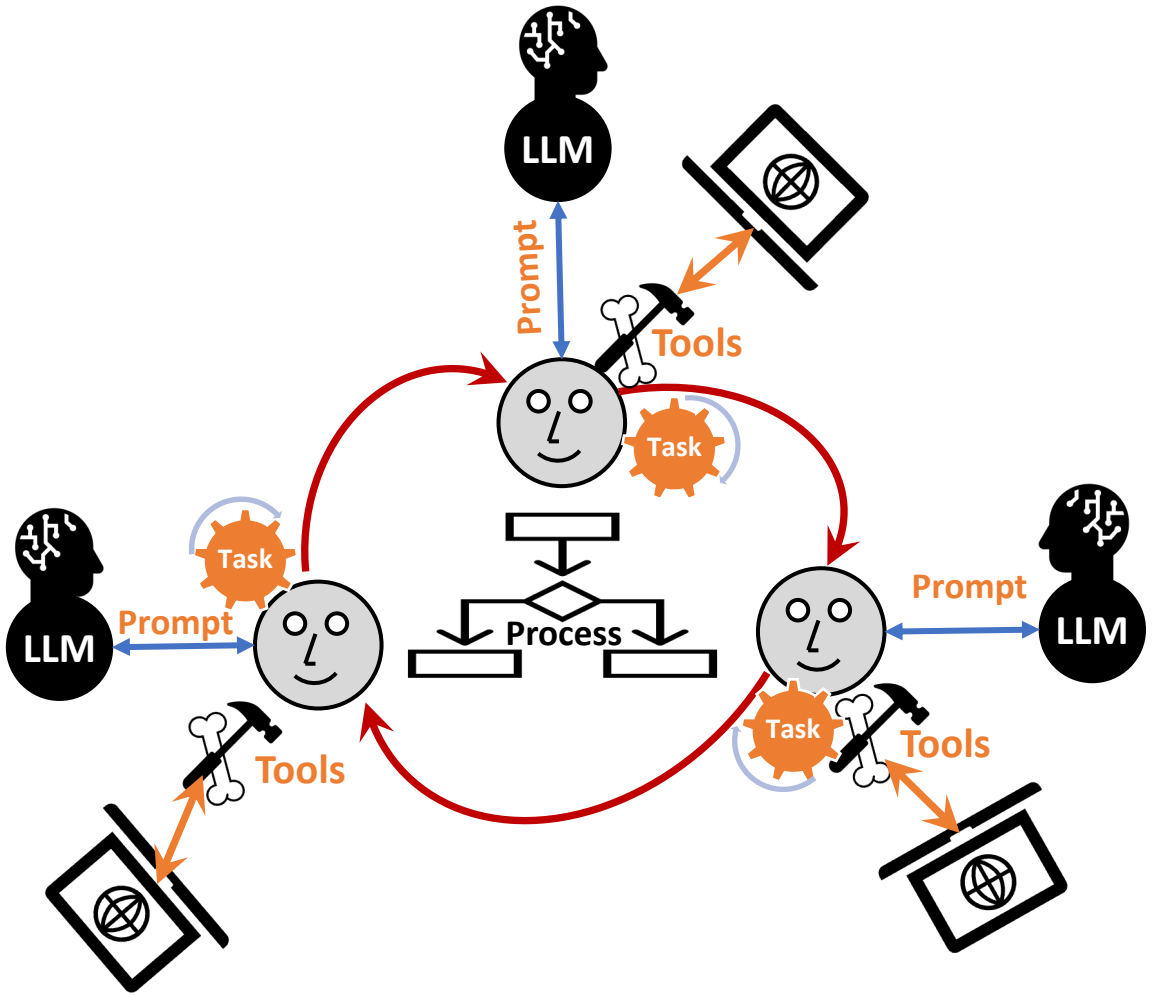
```python
@function_tool
def get_list_of_devices() -> List:
    """Get The list of devices"""
    return ["Timing chain", "Alternator", "Teperature sensor", "Humidity sensor"]
```

```python
@function_tool
def get_lifespan_of_device(device: str) -> dict:
    """
    get the lifespan of given device
    """
    devices = {
        "Timing chain": {"device_name": "Timing chain", "life_chain": 50},
        "Alternator": {"device_name": "Alternator", "life_chain": 60},
        "Teperature sensor": {"device_name": "Teperature sensor", "life_chain": 120},
        "Humidity sensor": {"device_name": "Humidity sensor", "life_chain": 133},
    }
    return devices[device]
```

```python
@function_tool
def get_last_maintenance_days(device: str) -> int:
    """
    get the the number of days spend after the last maintenance of the device
    """
    devices_maintenance_days = {
        "Timing chain": {
            "device_name": "Timing chain",
            "days_from_last_maintenance": 46,
        },
        "Alternator": {"device_name": "Alternator", "days_from_last_maintenance": 58},
        "Teperature sensor": {
            "device_name": "Teperature sensor",
            "days_from_last_maintenance": 12,
        },
        "Humidity sensor": {
            "device_name": "Humidity sensor",
            "days_from_last_maintenance": 15,
        },
    }
    return devices_maintenance_days[device]
```

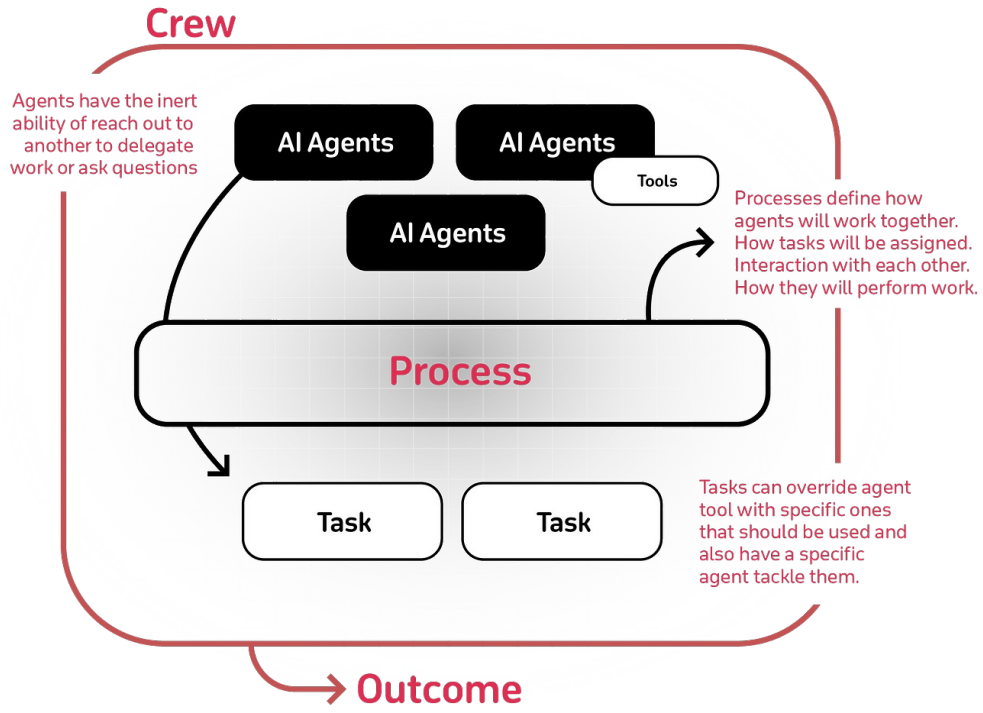
```python
class Device(BaseModel):
   name: str
   maintenance_days_left: int
   recommandations: List[str] = Field(description="List of recommandations")
```

AI Agents



https://docs.crewai.com/core-concepts/Agents/

Crew AI Framework



Agent

An agent is an **autonomous unit** programmed to:

- Perform tasks
- Make decisions
- Communicate with other agents

Attribute	Description
Role	Defines the agent's function within the crew. It determines the kind of tasks the agent is best suited for.
Goal	The individual objective that the agent aims to achieve. It guides the agent's decision-making process.
Backstory	Provides context to the agent's role and goal, enriching the interaction and collaboration dynamics.
LLM (optional)	Represents the language model that will run the agent. It dynamically fetches the model name from the OPENAI_MODEL_NAME environment variable, defaulting to "gpt-4" if not specified.
Tools (optional)	Set of capabilities or functions that the agent can use to perform tasks. Expected to be instances of custom classes compatible with the agent's execution environment. Tools are initialized with a default value of an empty list.
Function Calling LLM (optional)	Specifies the language model that will handle the tool calling for this agent, overriding the crew function calling LLM if passed. Default is None.
Max Iter (optional)	The maximum number of iterations the agent can perform before being forced to give its best answer. Default is 25.
Max RPM (optional)	The maximum number of requests per minute the agent can perform to avoid rate limits. It's optional and can be left unspecified, with a default value of None.
max_execution_time (optional)	Maximum execution time for an agent to execute a task It's optional and can be left unspecified, with a default value of None, menaning no max execution time
Verbose (optional)	Setting this to True configures the internal logger to provide detailed execution logs, aiding in debugging and monitoring. Default is False.
Allow Delegation (optional)	Agents can delegate tasks or questions to one another, ensuring that each task is handled by the most suitable agent. Default is True.
Step Callback (optional)	A function that is called after each step of the agent. This can be used to log the agent's actions or to perform other operations. It will overwrite the crew step_callback.
Cache (optional)	Indicates if the agent should use a cache for tool usage. Default is True.

Agent

```python
# Example: Creating an agent with all attributes
from crewai import Agent

agent = Agent(
  role='Data Analyst',
  goal='Extract actionable insights',
  backstory="""You're a data analyst at a large company.
  You're responsible for analyzing data and providing insights
  to the business.
  You're currently working on a project to analyze the
  performance of our marketing campaigns.""",
  tools=[my_tool1, my_tool2],  # Optional, defaults to an empty list
  llm=my_llm,  # Optional
  function_calling_llm=my_llm,  # Optional
  max_iter=15,  # Optional
  max_rpm=None, # Optional
  verbose=True,  # Optional
  allow_delegation=True,  # Optional
  step_callback=my_intermediate_step_callback,  # Optional
  cache=True  # Optional
)
```

Task

Tasks are specific assignments completed by agents.

Attribute	Description
Description	A clear, concise statement of what the task entails.
Agent	The agent responsible for the task, assigned either directly or by the crew's process.
Expected Output	A detailed description of what the task's completion looks like.
Tools (optional)	The functions or capabilities the agent can utilize to perform the task.
Async Execution (optional)	If set, the task executes asynchronously, allowing progression without waiting for completion.
Context (optional)	Specifies tasks whose outputs are used as context for this task.
Config (optional)	Additional configuration details for the agent executing the task, allowing further customization.
Output JSON (optional)	Outputs a JSON object, requiring an OpenAI client. Only one output format can be set.
Output Pydantic (optional)	Outputs a Pydantic model object, requiring an OpenAI client. Only one output format can be set.
Output File (optional)	Saves the task output to a file. If used with Output JSON or Output Pydantic, specifies how the output is saved.
Callback (optional)	A Python callable that is executed with the task's output upon completion.
Human Input (optional)	Indicates if the task requires human feedback at the end, useful for tasks needing human oversight.

Task

```
from crewai import Task

task = Task(
    description='Find
      and summarize the
      latest and most.
      relevant news on AI',
    agent=sales_agent
)
```

```
import os
os.environ["OPENAI_API_KEY"] = "Your Key"
os.environ["SERPER_API_KEY"] = "Your Key" # serper.dev API key
from crewai import Agent, Task, Crew
from crewai_tools import SerperDevTool

research_agent = Agent(
  role='Researcher',
  goal='Find and summarize the latest AI news',
  backstory="""You're a researcher at a large company.
  You're responsible for analyzing data and providing insights
  to the business.""",
  verbose=True
)
search_tool = SerperDevTool()
task = Task(
  description='Find and summarize the latest AI news',
  expected_output='A bullet list summary of the top 5 most important AI news',
  agent=research_agent,
  tools=[search_tool]
)
crew = Crew(
    agents=[research_agent],
    tasks=[task],
    verbose=2
)
result = crew.kickoff()
print(result)
```

Tools

- A Tool is a skill or function that agents can utilize to perform various actions.
- This includes tools from the crewAI Toolkit and LangChain Tools, enabling everything from simple searches to complex interactions and effective teamwork among agents.

```
pip install 'crewai[tools]'
```

```
import os
from crewai import Agent, Task, Crew

# Set up API keys
os.environ["SERPER_API_KEY"] = "Your Key"
os.environ["OPENAI_API_KEY"] = "Your Key"
```

Crew AI Tools

```
# Importing crewAI tools
from crewai_tools import (
    DirectoryReadTool, FileReadTool, SerperDevTool,
    WebsiteSearchTool
)
# Instantiate tools
docs_tool = DirectoryReadTool(directory='./blog-posts')
file_tool = FileReadTool()
search_tool = SerperDevTool()
web_rag_tool = WebsiteSearchTool()
```

Subclassing BaseTool¶

```
from crewai_tools import BaseTool
class MyCustomTool(BaseTool):
    name: str = "Name of my tool"
    description: str = "Clear description for what this tool is useful
for, you agent will need this information to use it."

    def _run(self, argument: str) -> str:
        # Implementation goes here
        return "Result from custom tool"
```

Utilizing the tool Decorator¶

```
from crewai_tools import tool
@tool("Name of my tool")
def my_tool(question: str) -> str:
    """Clear description for what this tool is useful for, you
agent will need this information to use it."""
    # Function logic here
    return "Result from your custom tool"
```

Tools

Agents

```
# Create agents
researcher = Agent(
    role='Market Research Analyst',
    goal='Provide up-to-date market
analysis of the AI industry',
    backstory='An expert analyst with a
keen eye for market trends.',
    tools=[search_tool, web_rag_tool],
    verbose=True
)
writer = Agent(
    role='Content Writer',
    goal='Craft engaging blog posts about
the AI industry',
    backstory='A skilled writer with a
passion for technology.',
    tools=[docs_tool, file_tool],
    verbose=True
)
```

Tasks

```
# Define tasks
research = Task(
    description='Research the latest trends in
the AI industry and provide a summary.',
    expected_output='A summary of the top 3
trending developments in the AI industry with a
unique perspective on their significance.',
    agent=researcher
)

write = Task(
    description='Write an engaging blog post
about the AI industry, based on the research
analyst's summary. Draw inspiration from the
latest blog posts in the directory.',
    expected_output='A 4-paragraph blog post
formatted in markdown with engaging,
informative, and accessible content, avoiding
complex jargon.',
    agent=writer,
    output_file='blog-posts/new_post.md'  # The
final blog post will be saved here
)
```

Crew

```
# Assemble a crew
crew = Crew(
    agents=[researcher, writer],
    tasks=[research, write],
    verbose=2
)

# Execute tasks
crew.kickoff()
```

Processes

- Processes orchestrate the execution of tasks by agents, akin to project management in human teams.
- These processes ensure tasks are distributed and executed efficiently, in alignment with a predefined strategy.
- Process Implementations :
  - **Sequential**: Executes tasks sequentially, ensuring tasks are completed in an orderly progression.
  - **Hierarchical**: Organizes tasks in a managerial hierarchy, where tasks are delegated and executed based on a structured chain of command. A manager language model (manager_llm) must be specified in the crew to enable the hierarchical process, facilitating the creation and management of tasks by the manager.
  - **Consensual Process (Planned)**: Aiming for collaborative decision-making among agents on task execution, this process type introduces a democratic approach to task management within CrewAI. It is planned for future development and is not currently implemented in the codebase.

```python
from crewai import Crew
from crewai.process import Process
from langchain_openai import ChatOpenAI

# Example: Creating a crew with a sequential process
crew = Crew(
    agents=my_agents,
    tasks=my_tasks,
    process=Process.sequential
)

# Example: Creating a crew with a hierarchical process
# Ensure to provide a manager_llm
crew = Crew(
    agents=my_agents,
    tasks=my_tasks,
    process=Process.hierarchical,
    manager_llm=ChatOpenAI(model="gpt-4")
)
```

Crew

- A crew in crewAI represents a collaborative group of agents working together to achieve a set of tasks.
- Each crew defines the strategy for task execution, agent collaboration, and the overall workflow.

Attribute	Description
Tasks	A list of tasks assigned to the crew.
Agents	A list of agents that are part of the crew.
Process (optional)	The process flow (e.g., sequential, hierarchical) the crew follows.
Verbose (optional)	The verbosity level for logging during execution.
Manager LLM (optional)	The language model used by the manager agent in a hierarchical process. Required when using a hierarchical process.
Function Calling LLM (optional)	If passed, the crew will use this LLM to do function calling for tools for all agents in the crew. Each agent can have its own LLM, which overrides the crew's LLM for function calling.
Config (optional)	Optional configuration settings for the crew, in Json or Dict[str, Any] format.
Max RPM (optional)	Maximum requests per minute the crew adheres to during execution.
Language (optional)	Language used for the crew, defaults to English.
Language File (optional)	Path to the language file to be used for the crew.
Memory (optional)	Utilized for storing execution memories (short-term, long-term, entity memory).
Cache (optional)	Specifies whether to use a cache for storing the results of tools' execution.
Embedder (optional)	Configuration for the embedder to be used by the crew. mostly used by memory for now
Full Output (optional)	Whether the crew should return the full output with all tasks outputs or just the final output.
Step Callback (optional)	A function that is called after each step of every agent. This can be used to log the agent's actions or to perform other operations; it won't override the agent-specific step_callback.
Task Callback (optional)	A function that is called after the completion of each task. Useful for monitoring or additional operations post-task execution.
Share Crew (optional)	Whether you want to share the complete crew information and execution with the crewAI team to make the library better, and allow us to train models.
Output Log File (optional)	Whether you want to have a file with the complete crew output and execution. You can set it using True and it will default to the folder you are currently and it will be called logs.txt or passing a string with the full path and name of the file.

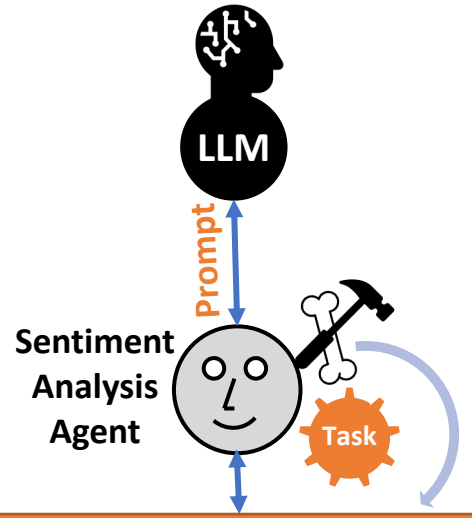
Crew

```
# Assemble the crew with a sequential process
my_crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, write_article_task],
    process=Process.sequential,
    full_output=True,
    verbose=True,
)
```

```
# Start the crew's task execution
result = my_crew.kickoff()
print(result)
print(crew.usage_metrics)
```

Crew AI Example App : Aspect based sentiment analysis

```
from crewai import Agent, Process, Task, Crew
from crewai.project import agent, task, CrewBase, crew
from langchain_groq import ChatGroq
from langchain_openai import ChatOpenAI
import json

OPEN_AI_KEY = "sk-proj-bkkDQOUCtyf7Ctj2afRXT3BlbkFJtxzf8IbMmA6uqAd1qt9j"
GROQ_API_KEY = "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"
```



Input

```
reviews = [
  "The screen is good. The keyboard is bad and the mousepad is quite",
  "The screen is good. The keyboard is bad and the mousepad is good",
  "The screen good. The keyboard is quite and the mousepad is good",
]
```

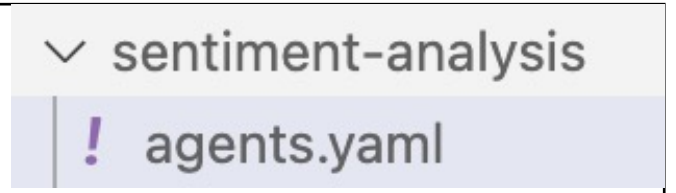
Crew AI : Sequential Process

Output

```
[ {
  'category': ['screen', 'keyboard', 'mousepad'],
  'polarity': ['positive', 'negative', 'neutral']
 },
 {
  'category': ['screen', 'keyboard', 'mousepad'],
  'polarity': ['positive', 'negative', 'positive']
 },
 {
   'category': ['screen', 'keyboard', 'mousepad'],
   'polarity': ['positive', 'positive', 'positive']
}
]
```

Crew AI Example App : Aspect based Sentiment Analysis

agents.yaml




```yaml
agents:
  systiment_analysis_agent:
    role: Aspect based sentiment analysis agent
    goal: Perform aspect based sentiment analysis of laptop reviews
    backstory: An expert of sentiment analysis of reviews
    allow_delegation: false
    verbose: true
tasks:
  sentiment_analysis_task:
    description: >
      Take a laptop review provided in the REVIW section and perform its aspect based sentiment analysis.
      In the review there might be one or more of the following : aspects; screen, keyboard, and mousepad.
      For the review presented as input,
      - Identify if there are any of the 3 aspects (screen, keyboard, mousepad) present in the review.
      - Assign a sentiment polarity (positive, negative or neutral) for each aspect.
      Arrange your response a JSON object with the following headers;
      - category:[list of aspects]
      - polarity:[list of corresponding polarities for each aspect]
      REVIEW_SECTION :
      {review}
    expected_output: >
      A consise aspect based sentiment analysis of the provided review
```

Crew AI Example App : Aspect based Sentiment Analysis

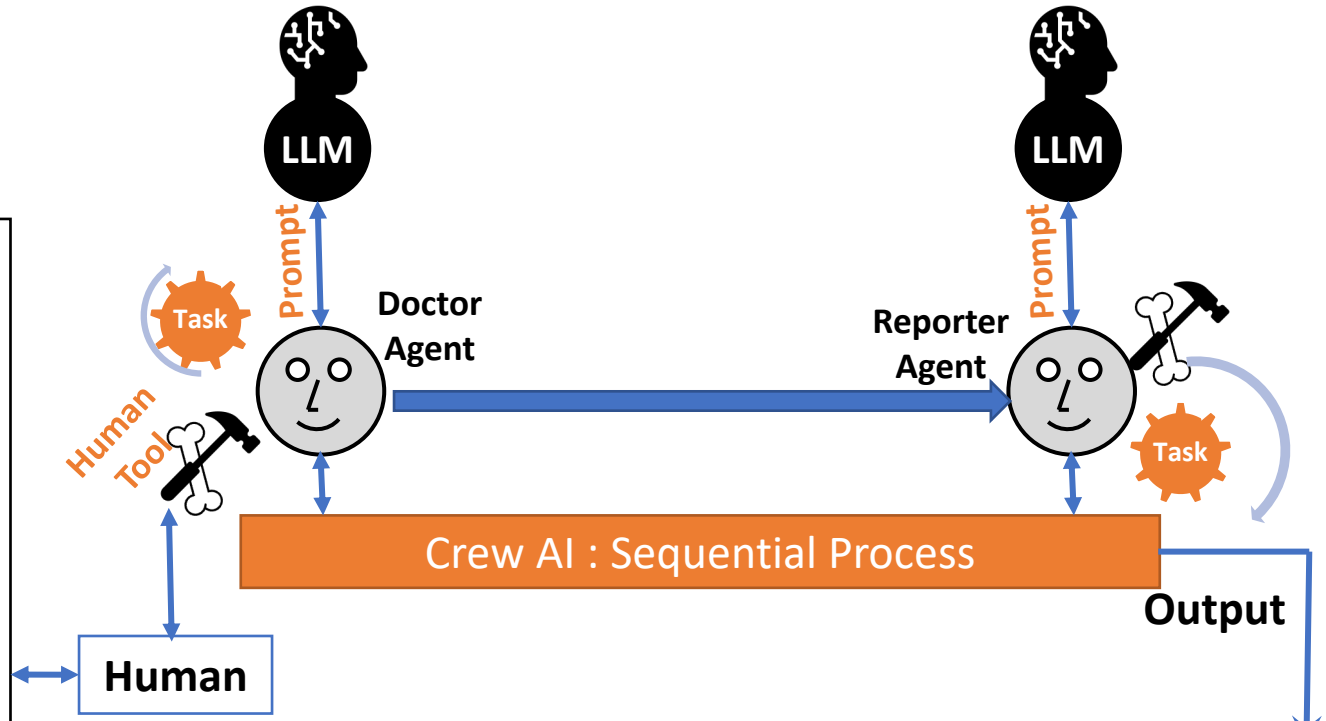
```
@CrewBase
class SentimentAnalysisCrew:
    """Aspect based sentiment analysis crew"" »
    agents_config = "config/sentiment-analysis/agents.yaml »

    def __init__(self) -> None:
        self.groq_llm = ChatGroq(
            model_name="mixtral-8x7b-32768", temperature=0, api_key=GROQ_API_KEY
        )
        self.gpt4_llm = ChatOpenAI(model="gpt-4", temperature=0, api_key=OPEN_AI_KEY)
    @agent
    def sentiment_analysis_agent(self) -> Agent:
        return Agent(
            llm=self.gpt4_llm,
            config=self.agents_config["agents"]["systiment_analysis_agent"],
        )

    @task
    def setiment_analysis_task(self) -> Task:
        return Task(
            config=self.agents_config["tasks"]["sentiment_analysis_task"],
            agent=self.sentiment_analysis_agent(),
        )
    @crew
    def sentiment_analysis_crew(self) -> Crew:
        return Crew(
            agents=self.agents, tasks=self.tasks, verbose=2, process=Process.sequential
        )
```

Crew AI Example App

```
reviews = [
  "The screen is good. The keyboard is bad and the mousepad is quite",
  "The screen is good. The keyboard is bad and the mousepad is good",
  "The screen good. The keyboard is quite and the mousepad is good",
]

crew = SentimentAnalysisCrew().sentiment_analysis_crew()
results = []
for review in reviews:
    result = crew.kickoff(inputs={"review": review})
    print("####################")
    print(result)
    results.append(json.loads(result))

print("========================")
print(results)
```

Medical Assistant Bot

- **What is your preferred language for this conversation?**
  - Français
- **Quel est votre nom?**
  - Mohamed
- **Quel est votre âge, Mohamed?**
  - 46
- **Quel est votre poids, Mohamed?**
  - 64
- **Quel est votre sexe, Mohamed?**
  - Masculin
- **Mohamed, avez-vous des difficultés à respirer ou essoufflement au repos ou à l'effort?**
  - oui
- **Mohamed, avez-vous un gonflement au niveau des jambes, des chevilles ou des pieds?**
  - oui
- **Mohamed, ressentez-vous de la fatigue ou de la faiblesse?**
  - oui
- **Mohamed, ressentez-vous un rythme cardiaque rapide ou irrégulier?**
  - oui
- **Mohamed, ressentez-vous des douleurs ou inconforts thoraciques?**
  - oui
-



Rapport médical.
Patient : Mohamed. Date : [Insérer la date]

Sur la base de l'entretien mené avec Mohamed, les symptômes suivants ont été identifiés :

- Essoufflement
- Fatigue
- Gonflement des jambes et des chevilles
- Battement de cœur rapide ou irrégulier

Ces symptômes sont associés à une insuffisance cardiaque, une maladie dans laquelle le cœur est incapable de pomper le sang efficacement pour répondre aux besoins de l'organisme. Il est important de noter que seul un professionnel de la santé peut diagnostiquer une insuffisance cardiaque au moyen d'un examen approfondi et de tests appropriés.
Recommandations :
-
-
-

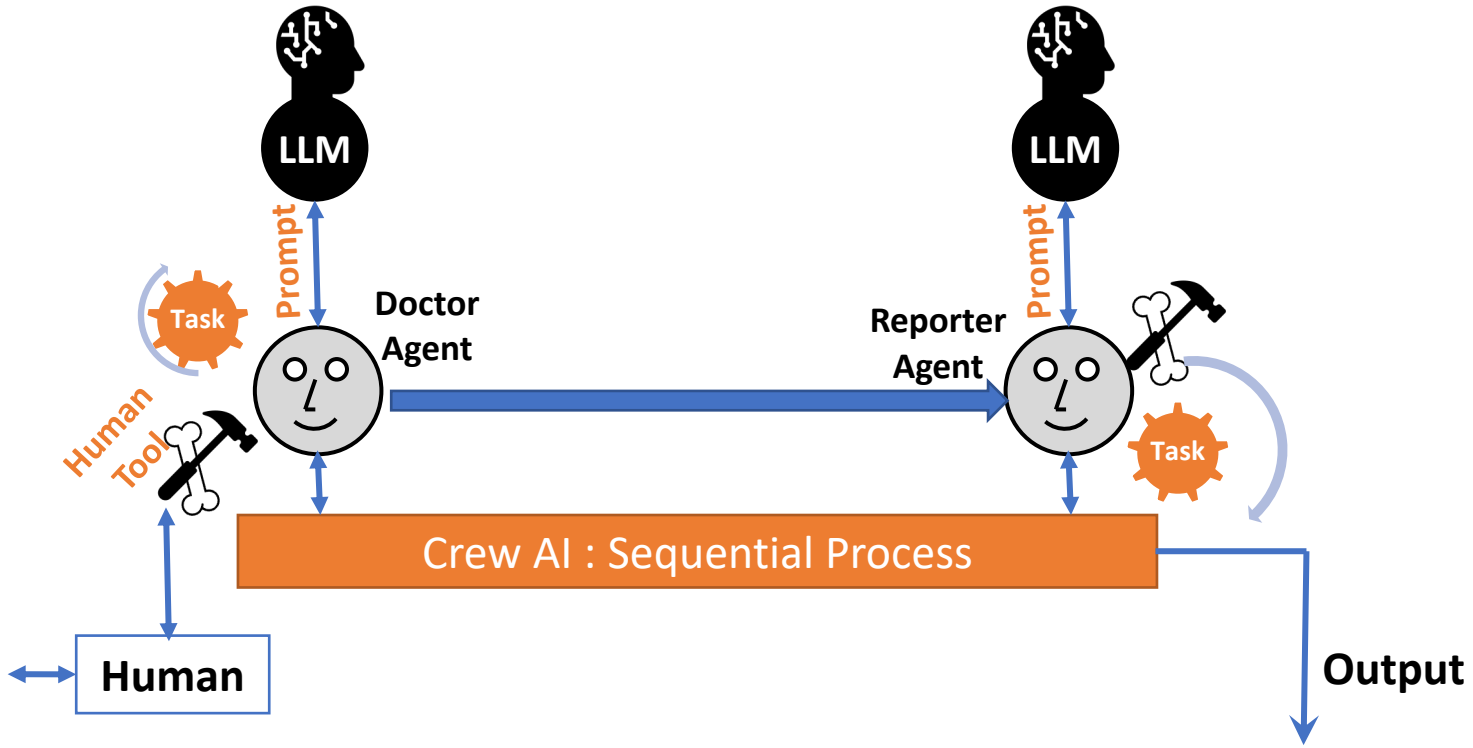
Crew AI Example App : Medical Assistant Agent


```yaml
agents:
  doctor_agent:
    role: Docteur
    goal: Diagnostiquer la maladie du patient
    backstory: >
      Un expert dans le domaine médical carliologue pour
diagnostiquer des éventuelles insuffisances cardiaques
    allow_delegation: false
    verbose: true

  reporter_agent:
    role: Rapporteur
    goal: >
Rédider un rapport médical qui résume l'interview du
docteur. utilise un outil medical pour proposer un
traitement au patient
    backstory: >
      Un expert qui a l'habitude de ridiger des rapports
médicaux sur la base d'un interveiw médical
    allow_delegation: false
    verbose: true
```

```yaml
tasks:
  interview_task:
    description: >
      Cette tâche consiste à inteviewer un paient qui présente des
signes d'insuffisance cardiaques.
      IL faudrait poser les questions suivantes :
      - demander au patient la langue qu'il souhaire pour l'entretien
      - En utilisant la langue du patient
      - demander au patient son nom
        - demander au patient son age
        - demander au patient son poids
        - demander au patient sont sex
        - Poser au patient 6 questions pour collecter les éléments d'un
diagnostique préliminaire
    expected_output: Des uqestion bien précises et ciblées
    max_iteration: 6

  reporter_task:
    description: >
      Résumer le contenu du rapport forni par le docteur
    expected_output: Rapport médical final
```

Crew AI Example App : Medical Assistant Agent

```
import os
from langchain_openai import ChatOpenAI
from langchain.agents import load_tools
from crewai import Agent, Task, Crew
from crewai.tasks.task_output import TaskOutput
from crewai.project import crew, agent, task, CrewBase
from langchain_groq import ChatGroq
```



Crew AI Example App : Medical Assistant Agent

```
@CrewBase
class MedicalCrew:
    """Equipage médical"""
    agents_config = "config/medical/medical_agents_config.yaml"

    def __init__(self) -> None:
        self.gpt4_llm = ChatOpenAI(model="gpt-4", temperature=0.4)
        self.groq_llm = ChatGroq(temperature=0.4, model_name="mixtral-8x7b-32768")
        self.human_tools = load_tools(["human"])

    @agent
    def doctor_agent(self) -> Agent:
        return Agent(
            config=self.agents_config["agents"]["doctor_agent"],
            llm=self.groq_llm,
        )

    @agent
    def reporter_agent(self) -> Agent:
        return Agent(
            config=self.agents_config["agents"]["reporter_agent"],
            llm=self.groq_llm,
        )
```

Crew AI Example App : Medical Assistant Agent

```
@task
    def interview_task(self) -> Task:
        return Task(
            config=self.agents_config["tasks"]["interview_task"],
            tools=self.human_tools,
            agent=self.doctor_agent(),
        )

    @task
    def reporter_task(self) -> Task:
        return Task(
            config=self.agents_config["tasks"]["reporter_task"],
            agent=self.reporter_agent(),
        )

    @crew
    def medical_crew(self) -> Crew:
        return Crew(agents=self.agents, tasks=self.tasks, verbose=2)
```

```
medical_crew = MedicalCrew()
crew = medical_crew.medical_crew()
result = crew.kickoff()
print(result)
```

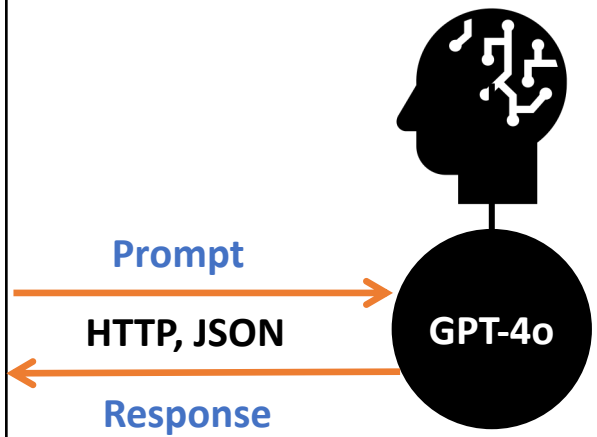
Interact with LLMs : Open AI

https://platform.openai.com/api-keys

```
from langchain_openai import ChatOpenAI
```

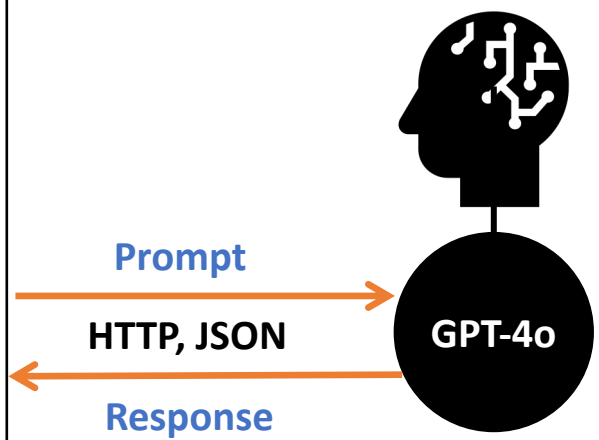
```
OPEN_AI_KEY = "sk-proj-bkkDQOUCtyf7Ctj2afRXT3BlbkFJtxzf8IbMmA6uqAd1qt9j"

gpt4_llm = ChatOpenAI(model="gpt-4o", temperature=0, api_key=OPEN_AI_KEY)

system_message ="""
  Classify the sentiment of the review presented in the input as 'positive' or 'negative'
  The review will be delimited by triple backticks that is ``` in the input.
  Answer only 'positive' or 'negative'
  Do not explain your answer.
"""

user_message_template ="```{review}```"

user_message ="I think that your services are very fine"

zero_shot_prompt = [
 {"role":"system","content":system_message},
 {"role":"user", "content":user_message_template.format(review=user_message) },
]
response = gpt4_llm.invoke(zero_shot_prompt )

print(response.content)
```

Text Classification



Interact with LLMs : Open AI - Direct Http Requests

```
import base64
import requests
# OpenAI API Key
api_key = "sk-proj-4f0aE2icZAzZ8qj1AvubT3BlbkFJqAsbp3Xx4qnvvIK5Jrj2"
headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {api_key}"
}
payload = {
     "model": "gpt-4o",
      "messages": [
        {"role":"system","content":system_message},
        {"role" : "user",
         "content" : user_message_template.format(review=user_message)
        }
    ],
    "max_tokens": 300,
    "temperature":0
}
url="https://api.openai.com/v1/chat/completions"
response = requests.post(url, headers=headers, json=payload)

print(response.json()['choices'][0]['message']['content'])
```

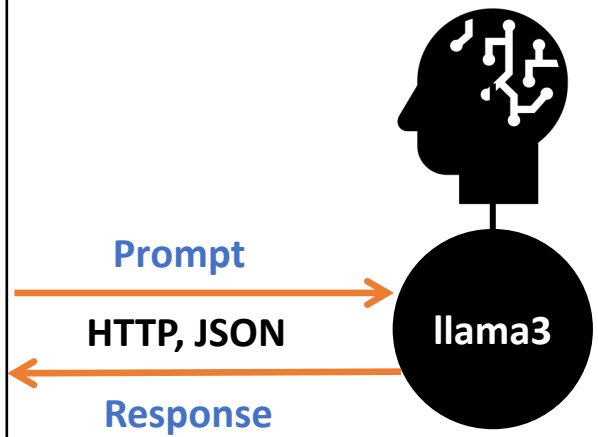


Interact with LLMs : Groq Ollama Mistral

https://console.groq.com/keys

```
from langchain_groq import ChatGroq
```

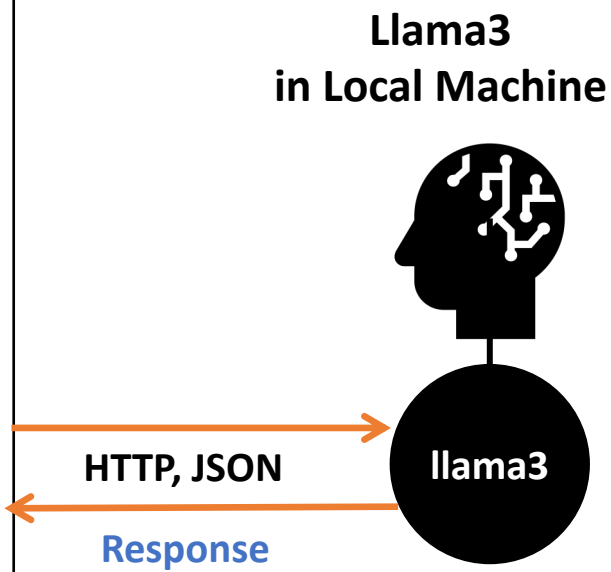
```
GROQ_API_KEY = "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"

groq_llm = ChatGroq(model_name="llama3-70b-8192", temperature=0, api_key=GROQ_API_KEY)

system_message ="""
   Classify the sentiment of the review presented in the input as 'positive' or 'negative'
   The review will be delimited by triple backticks that is ``` in the input.
   Answer only 'positive' or 'negative'
   Do not explain your answer.
"""

user_message_template ="```{review}```"


user_message ="I think that your services are very fine"

zero_shot_prompt = [
  {"role":"system","content":system_message},
  {"role":"user", "content":user_message_template.format(review=user_message2)},
 ]

response = groq_llm.invoke(zero_shot_prompt )

print(response.content.replace("</s>",""))
```



Interact with LLMs : Local Ollama LLM

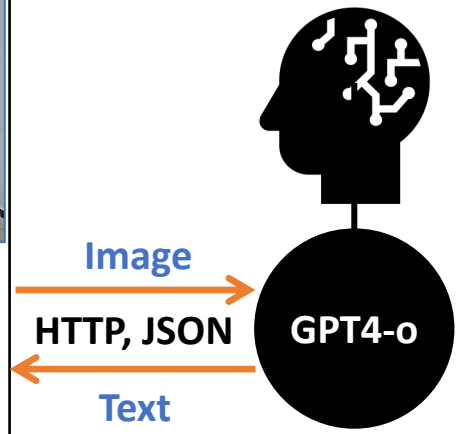
```
from from langchain_community.llms import Ollama
```

```
llama_llm = Ollama(model="llama3", temperature=0)

system_message ="""
   Classify the sentiment of the review presented in the input as 'positive' or 'negative'
   The review will be delimited by triple backticks that is ``` in the input.
   Answer only 'positive' or 'negative'
   Do not explain your answer.
   """

user_message_template ="```{review}```"
user_message ="The look is bad"

few_shot_prompt = [
  {"role":"system","content":system_message},
  {"role":"user", "content":user_message_template.format(review=user_message)},
]
response = llama_llm.invoke(few_shot_prompt )
print(response)
```



Interact with LLMs : Multi Model – Text + Image

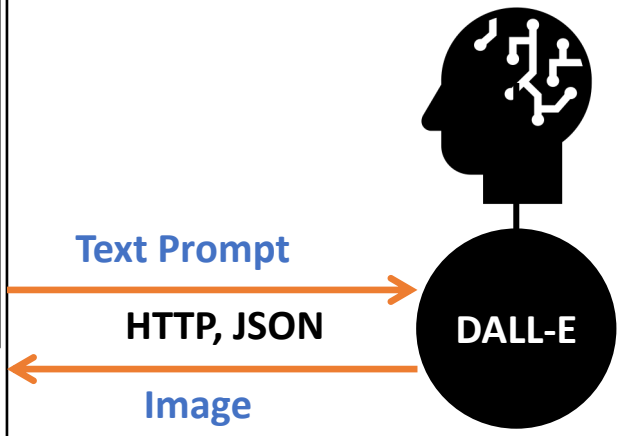
```python
import base64 import requests
api_key = "……"
# Function to encode the image
def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode('utf-8')
image_path = "images/IMG_4479.jpg"
base64_image = encode_image(image_path)
headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {api_key}"
}
prompt_text = """
Donne moi une description de cette image. Le résultat sera au format json avec les champs suivants :
 - Description : Une description de l'image
 - Nombre de personnes dans l'image
 - Nombre de filles
 - Nombre de garçon
"""
payload = {
    "model": "gpt-4o",
    "messages": [
      {"role": "user","content": [
          {"type": "text","text": prompt_text},{"type": "image_url","image_url": {
              "url": f"{base64_image}"}
          }]}],
    "max_tokens": 300
}
response = requests.post("https://api.openai.com/v1/chat/completions", headers=headers, json=payload)
print(response.json()['choices'][0]['message']['content'])
```




```
{
  "Description": "L'image montre une grande
salle de classe avec un total de 70
personnes. Elles sont assises à des bureaux
en rangées, tournées vers l'objectif,
souriantes et semblant participer à une
séance de cours. Les murs de la salle sont
blancs et le plafond est équipé de
luminaires modernes.",
  "Nombre de personnes dans l'image": 70,
  "Nombre de filles": 18,
  "Nombre de garçons": 52
}
```

Interact with LLMs : Image Generation – DALL-E

```python
import json
import requests
import io
import base64
from PIL import Image
from IPython.display import display

url = "https://api.openai.com/v1/images/generations"

headers = {
    "Content-Type": "application/json",
    "Authorization": f"Bearer {api_key}"
}

payload = {
    "model": "dall-e-3",
    "prompt": "un chat avec un costume dans une fête avec un café dans sa main",
    "response_format":"b64_json",
    "n": 1,
    "size":"1024x1024"
}

response = requests.post(url=f'{url}', headers=headers, json=payload)
image_data = response.json()["data"][0]["b64_json"]
image = Image.open(io.BytesIO(base64.b64decode(image_data)))
image.save('output.png')
display(image)
```



Interact with LLMs : Image Generation – Stable Diffusion

```python
import json
import requests
import io
import base64
from PIL import Image
from IPython.display import display

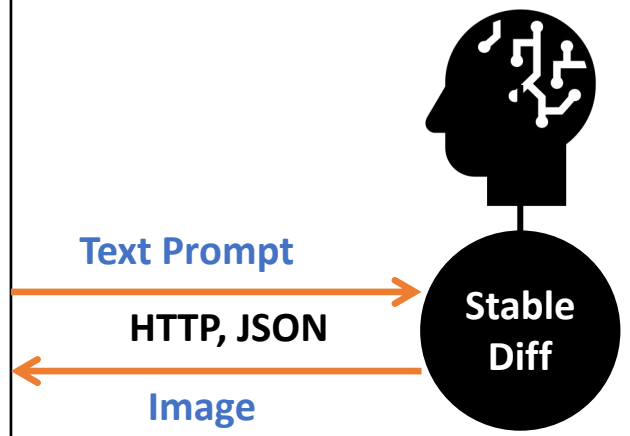
url = "http://127.0.0.1:7860"

payload = {
    "prompt": "Une peinture de PICASSO représentant un chat",
    "steps": 50,
    "width":600,
    "height":600
}

response = requests.post(url=f'{url}/sdapi/v1/txt2img', json=payload)

r = response.json()

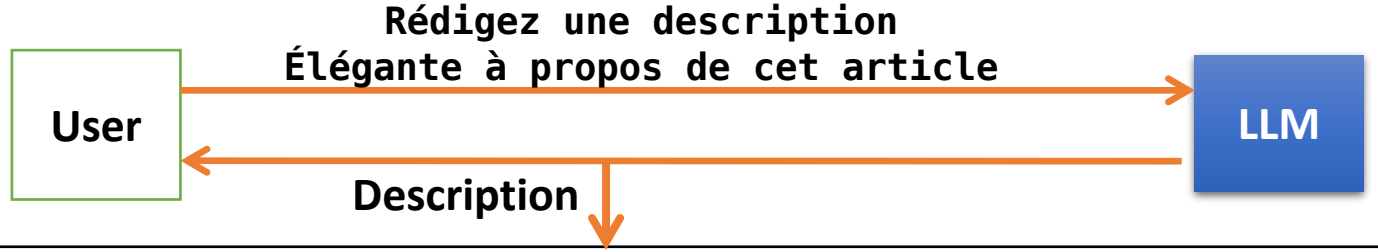
image = Image.open(io.BytesIO(base64.b64decode(r['images'][0])))
image.save('output2.png')
display(image)
```



```
$ ./webui.sh --api
```

Instructional prompt (Prompt pédagogique)

- Les invites pédagogiques sont un partenaire précieux dans les tâches créatives où les idées initiales sont difficiles à trouver.
- Voici une invite qui accomplit cette tâche ; varier la température générera une grande variété de descriptions de produits facilitant le processus créatif.




```
instuction_prompt = """

Vous êtes marketeur, Je voudrais mettre en vente ma voiture.
Vous trouverez ci-dessous quelques informations sur cette voiture :
- Marque : Toyota Corolla
- Modèle : 2017
- Couleur : Noir.
- Puissance fiscale : 8 cv
- Dernier prix : 120000
- Contact : 06442211 , infos@gmail.com
Avec ces informations, rédigez une description élégante « À propos de
cette voiture» qui sera utilisée sur un site de vente des voitures
d'occasion.

Utilisez des puces pour délimiter les principales fonctionnalités
mentionnées dans la description.
"""
```

Reasoning Prompt with ReAct Framework

- Une meilleure méthode pour utiliser les capacités de raisonnement de GPT consiste à utiliser le Framework **ReAct** (**Reasoning and Action**).
- Avec ce Framework, nous considérons GPT comme un agent intelligent et nous codifions explicitement les actions disponibles pour le modèle.
- Cela incite le modèle à peser sur les alternatives disponibles avant d'agir.

```
prompt = """
Dans notre université, les enseignants ont constaté un réel problème chez les apprenants comme le manque de concentration, baisse du niveau des langues, des matières scientifiques et manque de discipline. L'équipe pédagogique a identifier quelques pistes pour résoudre ce problème comme :
1. La révisions des pratiques pédagogiques.
2. Instauration d'une pratique disciplinaire.
3. Révision des méthodes d'évaluation
4. Autres actions à identifier

Recommander un plan d'action et créer un plan étape par étape que l'université devra suivre dans cette situation.
Présentez votre réponse sous le format suivant :
{
  "question": <la question d'entrée à laquelle il faut répondre>,
  "reflection": <réfléchissez à vos options et au plan d'action pour chaque option>,
  "action": <l'action à entreprendre parmi les trois options présentées">,
  "raison": <le raisonnement derrière votre réponse>
  "étapes" : <plan étape par étape mettant en œuvre l'action>
}
"""
```

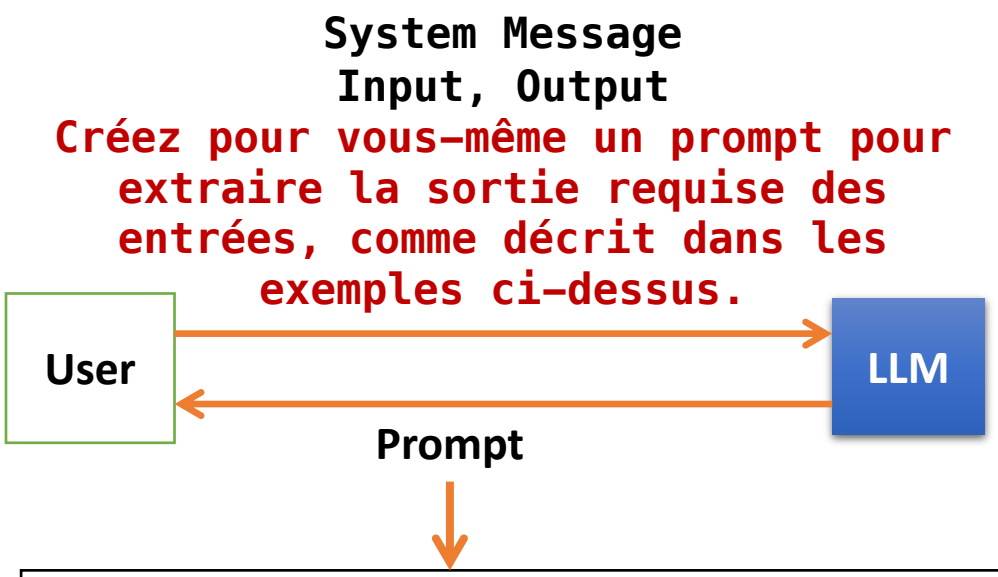
Induction Prompt

- Une approche pour générer des prompts pour un modèle de langage consiste à :
  - Présenter des paires d'entrées-sorties
  - et à inviter le modèle à générer une prompt qui obtient la sortie donnée à partir des entrées fournies.
- Cette technique peut servir de base d'idées d'incitation, contribuant ainsi à relancer le processus de génération de prompts.

```
Vous êtes un assistant qui aide l'équipe marketing d'une entreprise
d'électronique à mieux comprendre les avis de ses clients.
Les exemples d'entrées-sorties suivants ont été collectés à titre de
suggestions pour que vous puissiez en tirer des leçons.
- Entrée : Gris commandé qui annonce un éclairage vert, quand on opte pour
une esthétique cheap, c'est bouleversant. La -souris fonctionne bien.
- Sortie : Souris
- Entrée : j'en ai acheté un pour les jeux sur PC. J'ai adoré, puis j'en ai
acheté une autre pour le travail. Cette souris n'est pas à la hauteur des
souris haut de gamme comme la série Logitech MX Master, mais à 1/5-/8ème du
prix, je ne m'attendais pas à ce niveau de qualité. Il fonctionne bien, la
molette de la souris semble lourde, les boutons latéraux sont bien placés
avec différentes textures pour que vous puissiez les distinguer. La souris
semble plutôt plastique et bon marché, mais pour le prix, c'est à peu près ce
à quoi je m'attendais. J'aime une souris filaire pour éviter que le
pointeur/le jeu ne saute à cause de la latence. Fil long également, ce qui
minimise les problèmes d'accrochage. Très bon rapport qualité/prix dans
l'ensemble.
- Sortie : souris, Logitech MX Master, boutons DPI, molette de la souris, fil

Créez pour vous-même un prompt pour extraire la sortie requise des entrées,
comme décrit dans les exemples ci-dessus.

Créez l'invite pour qu'elle contienne des exemples généralisés à partir de
ceux présentés ci-dessus.
N'oubliez pas que l'invite doit contenir des instructions que vous pouvez
comprendre et générer le résultat attendu compte tenu de l'entrée.
"""
```



Prompt:
Lisez attentivement l'avis du client et identifiez le principal produit électronique en question. Notez également toutes les caractéristiques ou composants spécifiques du produit mentionnés, ainsi que tout autre produit électronique mentionné à des fins de comparaison. Le résultat doit être une liste de ces éléments.

Paraphrasing Prompt

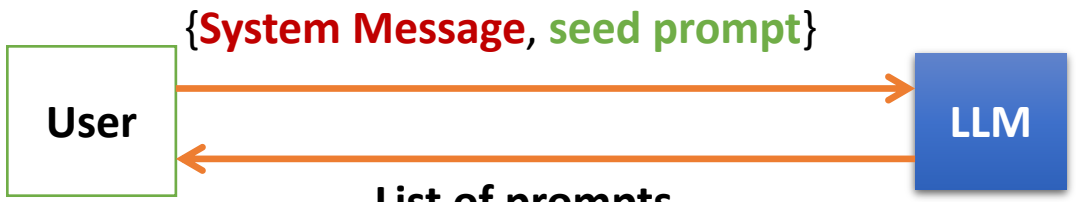
- Le Prompte de paraphrase est une technique qui consiste à générer un ensemble de promptes à l'aide d'un LLM à partir d'un Prompt de départ,
- Puis à évaluer les performances de chaque prompte sur un ensemble de test.
- Choisir le prompt le plus performant

```
paraphrase_prompt = """

Un promprt de départ vous sera présentée,
délimitée par trois backticks, c'est-à-dire
```.
Ce prompt de départ sera présentée à un grand
modèle de langage qui génère une sortie
requise spécifique.

Veuillez générer 5 variantes du prompt de
départ en gardant intacte l'intention de
l'invite de départ.

```
Veuillez analyser les avis clients suivants
et identifier les produits et fonctionnalités
qui sont mentionnés.
Fournissez une liste de produits et de
fonctionnalités mentionnés dans chaque avis.
```
"""
```




```
``` Analyzez les commentaires des clients fournis ci-dessous et identifiez les produits et
les fonctionnalités respectives qui sont discutés. Compilez une liste de ces produits et de
leurs fonctionnalités correspondantes mentionnées dans chaque avis. ```

``` Examinez les témoignages de clients fournis et déterminez les produits spécifiques ainsi
que leurs caractéristiques auxquels il est fait référence. Créez une liste détaillant le
produit et ses fonctionnalités associées mentionnées dans chaque avis. ```

``` Veuillez examiner les évaluations des clients ci-dessous et identifier les produits
explicites et leurs fonctionnalités qui sont en discussion. Créez une liste des produits et
fonctionnalités mentionnés dans chaque avis individuel. ```

``` Pouvez-vous analyser les retours clients suivants et reconnaître les produits et les
caractéristiques sur lesquels ils mettent l'accent ? Établissez une liste comprenant les
produits et les fonctionnalités mentionnés dans chacun des avis. ```

``` Inspectez les évaluations des clients qui s'ensuivent et discernez les produits et leurs
attributs qui sont mis en avant. Compilez une liste détaillant les produits et les
fonctionnalités respectives mentionnées dans chaque avis. ```
```

CoT Prompt : Chain-of-Thought Prompting :

Chain-of-Thought Prompting : CoT Prompt

- Pour l'invite CoT, nous ajoutons des instructions détaillées étape par étape au message système demandant au modèle de réfléchir attentivement avant de décider du résultat à générer
- En dehors de cet ajout, il n'y a aucun autre changement par rapport à prompt standard

```
cot_system_message = """
Classify the sentiment of movie reviews presented
in the input as 'positive' or 'negative'.
Movie reviews will be delimited by triple backticks
in the input.
Answer only 'positive' or 'negative'. Do not
explain your answer.

Instructions:
1. Carefully read the text of the review and think
through the options for sentiment provided
2. Consider the overall sentiment of the review and
estimate the probability of the review being
positive

To reiterate, your answer should strictly only
contain the label: positive or negative.
"""
```

Standard Prompt

- Q : Ahmed has 5 tennis balls, he buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls doas he have now?
- A : **The answer is 11**
- Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more. How many apples do they have?

The cafeteria now has 9 apples.

CoT Prompt

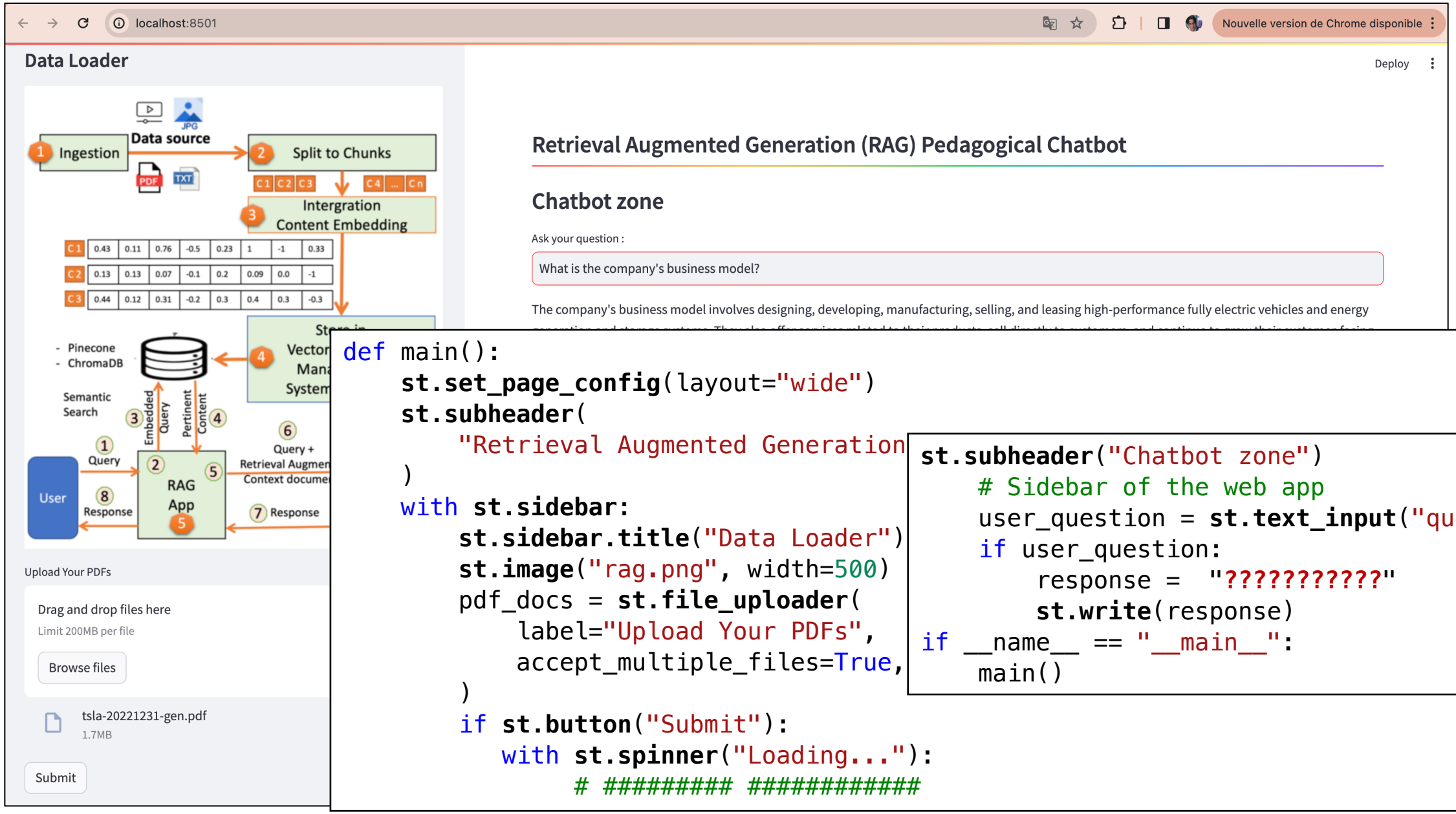
- Q : Ahmed has 5 tennis balls, he buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls doas he have now?
- A : **Ahmed started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5+6=11. The answer is 11**
- Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more. How many apples do they have?

The cafeteria started with 23 apples. They used 20, so they had 23–20=3 apples left. Then they bought 6 more, so they now have 3+6=9 apples.

The building blocks of RAG

```
# Importing the streamlit library to create the web app interface
import streamlit as st

# Importing the PyPDF2 library to read the PDF files and extract the text from the PDF files
from PyPDF2 import PdfReader

# Importing the CharacterTextSplitter class from the langchain library to split the text into chunks
from langchain.text_splitter import CharacterTextSplitter

# Importing the OpenAIEmbeddings class from the langchain library to create the vector store
from langchain.embeddings import OpenAIEmbeddings

# Importing the FAISS class from the langchain library to create the vector store
from langchain.vectorstores import FAISS

# Importing the ChatOpenAI class from the langchain library to create the language model
from langchain.chat_models import ChatOpenAI

# Importing the ChatPromptTemplate class from the langchain library to create the prompt
from langchain_core.prompts import ChatPromptTemplate

# Importing the create_stuff_documents_chain and create_retrieval_chain functions from the langchain library
from langchain.chains.combine_documents import create_stuff_documents_chain

# Importing the create_retrieval_chain function
from langchain.chains import create_retrieval_chain
from langchain_community.llms import Ollama
from langchain_community.embeddings.fastembed import FastEmbedEmbeddings
```

The building blocks of RAG




```python
def main():
    st.set_page_config(layout="wide")
    st.subheader(
        "Retrieval Augmented Generation
    )
    with st.sidebar:
        st.sidebar.title("Data Loader")
        st.image("rag.png", width=500)
        pdf_docs = st.file_uploader(
            label="Upload Your PDFs",
            accept_multiple_files=True,
        )
        if st.button("Submit"):
            with st.spinner("Loading..."):
                # ######### ############
```

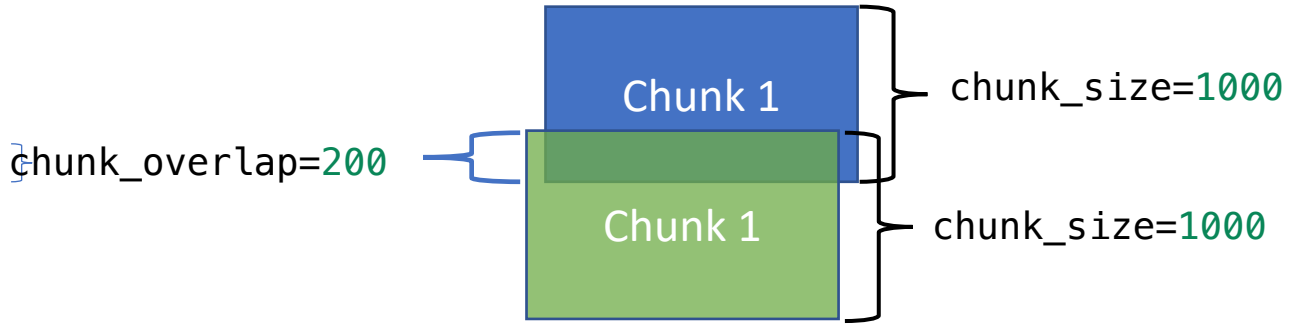
```python
st.subheader("Chatbot zone")
    # Sidebar of the web app
    user_question = st.text_input("question :")
    if user_question:
        response =  "???????????"
        st.write(response)
if __name__ == "__main__":
    main()
```

RAG : Extracting Text from PDFs

```
with st.sidebar:
    st.sidebar.title("Sidebar")
    pdf_docs = st.file_uploader("Upload PDF", accept_multiple_files=True)
    if st.button("Submit"):
        # Loading spinner to show the process is running
        with st.spinner("Loading..."):
            # Extract the content of the PDF
            pdf_content = ""
            # Loop through the PDF files
            for pdf in pdf_docs:
                # Read the PDF file
                pdf_reader = PdfReader(pdf)
                # Loop through the pages of the PDF file
                for page in pdf_reader.pages:
                    # Extract the text from the PDF page and add it to the pdf_content variable
                    pdf_content += page.extract_text()
                    # st.write(pdf_content)
```

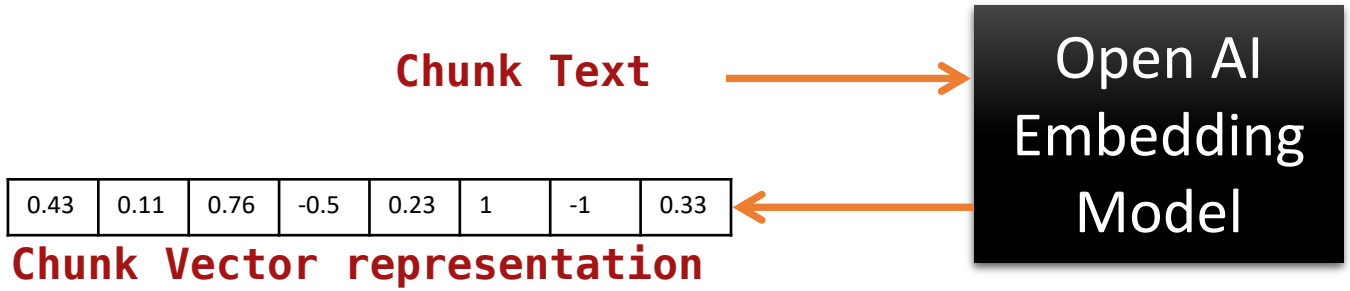
RAG / Split Text into Chunks

```
# Get chunks of the content
# Split the text into chunks of 1000 characters with an overlap of 200 characters
text_splitter = CharacterTextSplitter(
    separator="\n",
    chunk_size=1000,
    chunk_overlap=200,
    length_function=len,
)

# Split the text into chunks of 1000 characters with an overlap of 200 characters
chunks = text_splitter.split_text(pdf_content)

# Display the chunks of the text
st.write(chunks)
```



RAG : Create Vector Store using Open AI Embedding

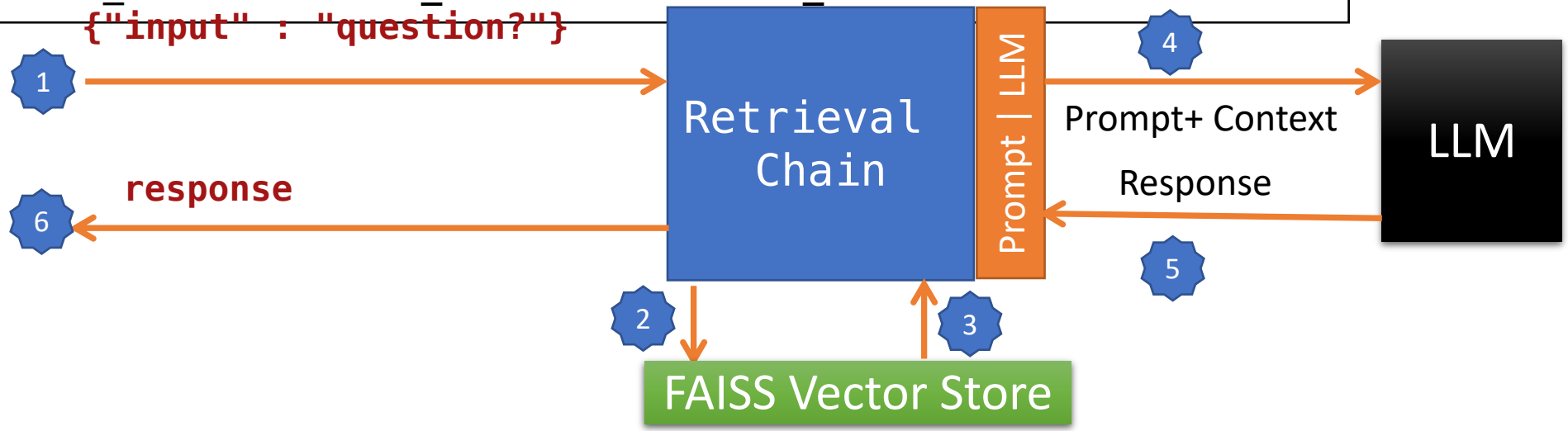
```
# # OpenAI API key
OPEN_API_KEY = "sk-……"

# Create the OpenAIEmbeddings object
embedding_model = OpenAIEmbeddings(api_key = OPEN_API_KEY)
# embedding_model = FastEmbedEmbeddings()

# Create the FAISS vector store from the text chunks and the OpenAIEmbeddings object
openai_vector_store = FAISS.from_texts(
    texts=chunks, embedding = embedding_model
)
```



RAG

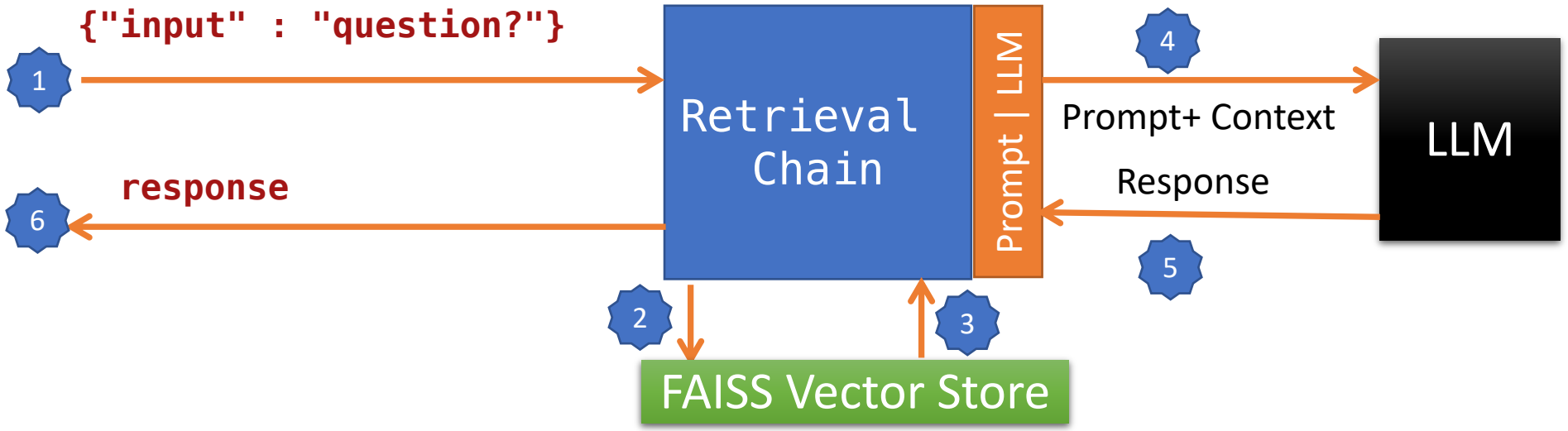
```
llm = ChatOpenAI(api_key=OPEN_API_KEY)
# llm = Ollama(model="llama3", temperature=0)
prompt = ChatPromptTemplate.from_template(
"""
Answer the following question based only on the provided context:
<context>
{context}
</context>
Question: {input}
"""
)
document_chain = create_stuff_documents_chain(llm, prompt)
retriever = openai_vector_store.as_retriever()
retrieval_chain = create_retrieval_chain(retriever, document_chain)
st.session_state.retrieve_chain = retrieval_chain
```

1. User Query
2. Similarity Vector Search
3. Relevant Documents
4. Prompt LLM
5. LLM Response
6. RAG Response

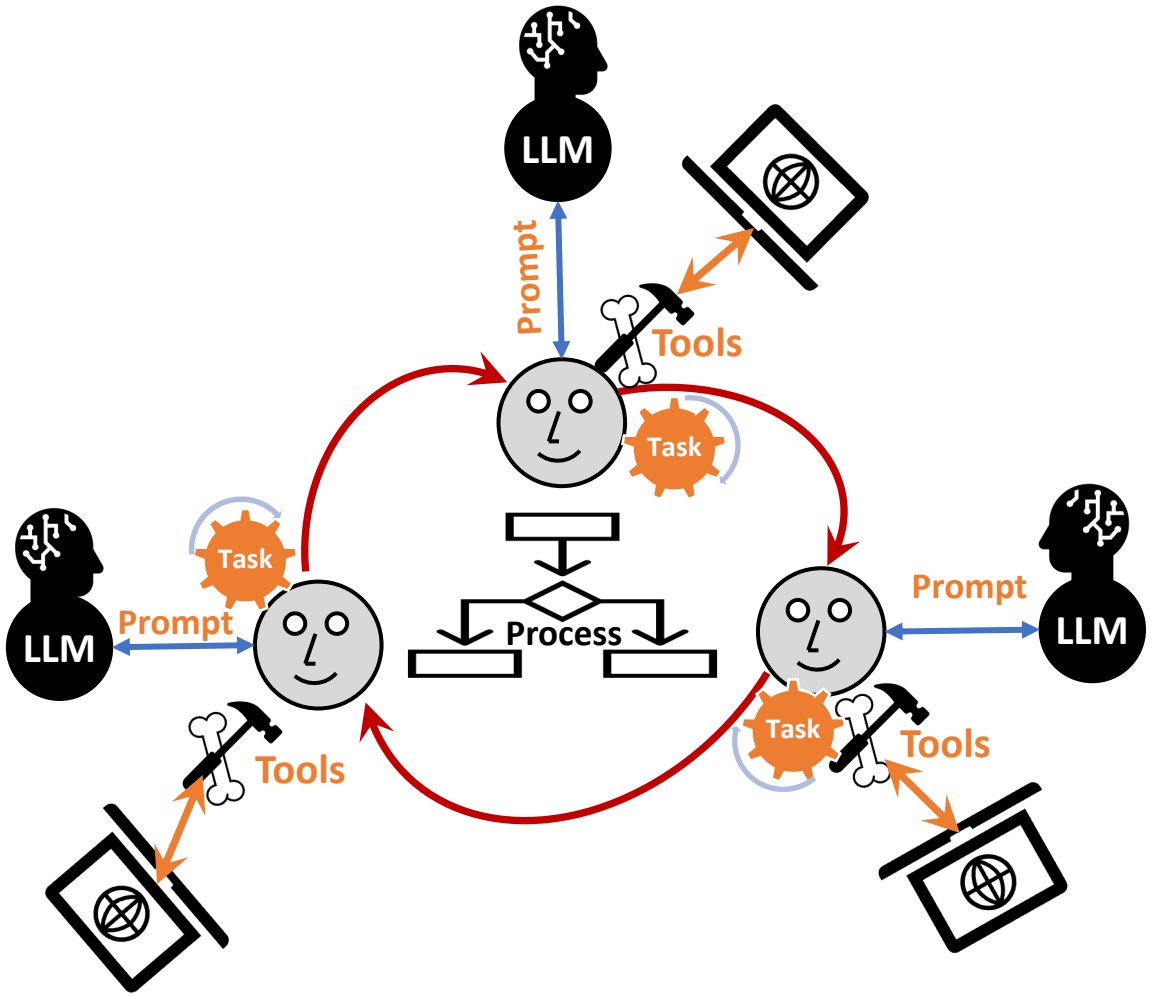


RAG

```
st.subheader("Chatbot zone")

user_question = st.text_input("Ask your question :")

if user_question:

   response = st.session_state.retrieve_chain.invoke({"input": user_question})

   st.write(response["answer"])
```

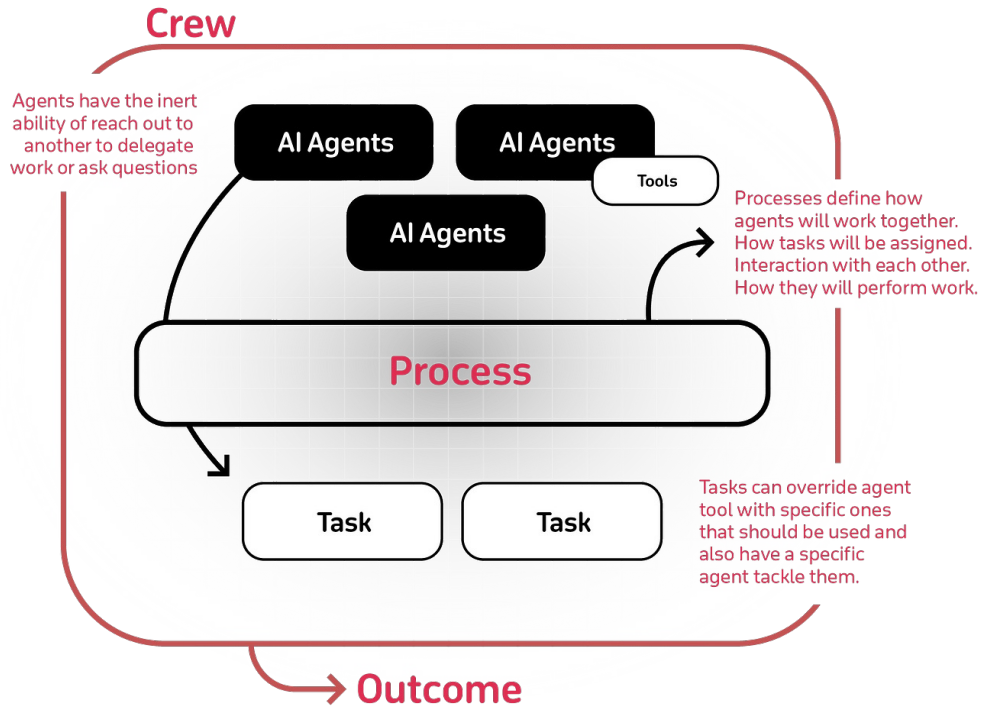


AI Agents



https://docs.crewai.com/core-concepts/Agents/

Crew AI Framework



Agent

An agent is an **autonomous unit** programmed to:

- Perform tasks
- Make decisions
- Communicate with other agents

Attribute	Description
Role	Defines the agent's function within the crew. It determines the kind of tasks the agent is best suited for.
Goal	The individual objective that the agent aims to achieve. It guides the agent's decision-making process.
Backstory	Provides context to the agent's role and goal, enriching the interaction and collaboration dynamics.
LLM (optional)	Represents the language model that will run the agent. It dynamically fetches the model name from the OPENAI_MODEL_NAME environment variable, defaulting to "gpt-4" if not specified.
Tools (optional)	Set of capabilities or functions that the agent can use to perform tasks. Expected to be instances of custom classes compatible with the agent's execution environment. Tools are initialized with a default value of an empty list.
Function Calling LLM (optional)	Specifies the language model that will handle the tool calling for this agent, overriding the crew function calling LLM if passed. Default is None.
Max Iter (optional)	The maximum number of iterations the agent can perform before being forced to give its best answer. Default is 25.
Max RPM (optional)	The maximum number of requests per minute the agent can perform to avoid rate limits. It's optional and can be left unspecified, with a default value of None.
max_execution_time (optional)	Maximum execution time for an agent to execute a task It's optional and can be left unspecified, with a default value of None, menaning no max execution time
Verbose (optional)	Setting this to True configures the internal logger to provide detailed execution logs, aiding in debugging and monitoring. Default is False.
Allow Delegation (optional)	Agents can delegate tasks or questions to one another, ensuring that each task is handled by the most suitable agent. Default is True.
Step Callback (optional)	A function that is called after each step of the agent. This can be used to log the agent's actions or to perform other operations. It will overwrite the crew step_callback.
Cache (optional)	Indicates if the agent should use a cache for tool usage. Default is True.

Agent

```python
# Example: Creating an agent with all attributes
from crewai import Agent

agent = Agent(
  role='Data Analyst',
  goal='Extract actionable insights',
  backstory="""You're a data analyst at a large company.
  You're responsible for analyzing data and providing insights
  to the business.
  You're currently working on a project to analyze the
  performance of our marketing campaigns.""",
  tools=[my_tool1, my_tool2],  # Optional, defaults to an empty list
  llm=my_llm,  # Optional
  function_calling_llm=my_llm,  # Optional
  max_iter=15,  # Optional
  max_rpm=None, # Optional
  verbose=True,  # Optional
  allow_delegation=True,  # Optional
  step_callback=my_intermediate_step_callback,  # Optional
  cache=True  # Optional
)
```

Task

Tasks are specific assignments completed by agents.

Attribute	Description
Description	A clear, concise statement of what the task entails.
Agent	The agent responsible for the task, assigned either directly or by the crew's process.
Expected Output	A detailed description of what the task's completion looks like.
Tools (optional)	The functions or capabilities the agent can utilize to perform the task.
Async Execution (optional)	If set, the task executes asynchronously, allowing progression without waiting for completion.
Context (optional)	Specifies tasks whose outputs are used as context for this task.
Config (optional)	Additional configuration details for the agent executing the task, allowing further customization.
Output JSON (optional)	Outputs a JSON object, requiring an OpenAI client. Only one output format can be set.
Output Pydantic (optional)	Outputs a Pydantic model object, requiring an OpenAI client. Only one output format can be set.
Output File (optional)	Saves the task output to a file. If used with Output JSON or Output Pydantic, specifies how the output is saved.
Callback (optional)	A Python callable that is executed with the task's output upon completion.
Human Input (optional)	Indicates if the task requires human feedback at the end, useful for tasks needing human oversight.

Task

```
from crewai import Task

task = Task(
    description='Find
      and summarize the
      latest and most.
      relevant news on AI',
    agent=sales_agent
)
```

```
import os
os.environ["OPENAI_API_KEY"] = "Your Key"
os.environ["SERPER_API_KEY"] = "Your Key" # serper.dev API key
from crewai import Agent, Task, Crew
from crewai_tools import SerperDevTool

research_agent = Agent(
  role='Researcher',
  goal='Find and summarize the latest AI news',
  backstory="""You're a researcher at a large company.
  You're responsible for analyzing data and providing insights
  to the business.""",
  verbose=True
)
search_tool = SerperDevTool()
task = Task(
  description='Find and summarize the latest AI news',
  expected_output='A bullet list summary of the top 5 most important AI news',
  agent=research_agent,
  tools=[search_tool]
)
crew = Crew(
    agents=[research_agent],
    tasks=[task],
    verbose=2
)
result = crew.kickoff()
print(result)
```

Tools

- A Tool is a skill or function that agents can utilize to perform various actions.
- This includes tools from the crewAI Toolkit and LangChain Tools, enabling everything from simple searches to complex interactions and effective teamwork among agents.

```
pip install 'crewai[tools]'
```

```
import os
from crewai import Agent, Task, Crew

# Set up API keys
os.environ["SERPER_API_KEY"] = "Your Key"
os.environ["OPENAI_API_KEY"] = "Your Key"
```

Crew AI Tools

```
# Importing crewAI tools
from crewai_tools import (
    DirectoryReadTool, FileReadTool, SerperDevTool,
    WebsiteSearchTool
)
# Instantiate tools
docs_tool = DirectoryReadTool(directory='./blog-posts')
file_tool = FileReadTool()
search_tool = SerperDevTool()
web_rag_tool = WebsiteSearchTool()
```

Subclassing BaseTool¶

```
from crewai_tools import BaseTool
class MyCustomTool(BaseTool):
    name: str = "Name of my tool"
    description: str = "Clear description for what this tool is useful
for, you agent will need this information to use it."

    def _run(self, argument: str) -> str:
        # Implementation goes here
        return "Result from custom tool"
```

Utilizing the tool Decorator¶

```
from crewai_tools import tool
@tool("Name of my tool")
def my_tool(question: str) -> str:
    """Clear description for what this tool is useful for, you
agent will need this information to use it."""
    # Function logic here
    return "Result from your custom tool"
```

Tools

Agents

```
# Create agents
researcher = Agent(
    role='Market Research Analyst',
    goal='Provide up-to-date market
analysis of the AI industry',
    backstory='An expert analyst with a
keen eye for market trends.',
    tools=[search_tool, web_rag_tool],
    verbose=True
)
writer = Agent(
    role='Content Writer',
    goal='Craft engaging blog posts about
the AI industry',
    backstory='A skilled writer with a
passion for technology.',
    tools=[docs_tool, file_tool],
    verbose=True
)
```

Tasks

```
# Define tasks
research = Task(
    description='Research the latest trends in
the AI industry and provide a summary.',
    expected_output='A summary of the top 3
trending developments in the AI industry with a
unique perspective on their significance.',
    agent=researcher
)

write = Task(
    description='Write an engaging blog post
about the AI industry, based on the research
analyst's summary. Draw inspiration from the
latest blog posts in the directory.',
    expected_output='A 4-paragraph blog post
formatted in markdown with engaging,
informative, and accessible content, avoiding
complex jargon.',
    agent=writer,
    output_file='blog-posts/new_post.md'  # The
final blog post will be saved here
)
```

Crew

```
# Assemble a crew
crew = Crew(
    agents=[researcher, writer],
    tasks=[research, write],
    verbose=2
)

# Execute tasks
crew.kickoff()
```

Processes

- Processes orchestrate the execution of tasks by agents, akin to project management in human teams.
- These processes ensure tasks are distributed and executed efficiently, in alignment with a predefined strategy.
- Process Implementations :
  - **Sequential**: Executes tasks sequentially, ensuring tasks are completed in an orderly progression.
  - **Hierarchical**: Organizes tasks in a managerial hierarchy, where tasks are delegated and executed based on a structured chain of command. A manager language model (manager_llm) must be specified in the crew to enable the hierarchical process, facilitating the creation and management of tasks by the manager.
  - **Consensual Process (Planned)**: Aiming for collaborative decision-making among agents on task execution, this process type introduces a democratic approach to task management within CrewAI. It is planned for future development and is not currently implemented in the codebase.

```python
from crewai import Crew
from crewai.process import Process
from langchain_openai import ChatOpenAI

# Example: Creating a crew with a sequential process
crew = Crew(
    agents=my_agents,
    tasks=my_tasks,
    process=Process.sequential
)

# Example: Creating a crew with a hierarchical process
# Ensure to provide a manager_llm
crew = Crew(
    agents=my_agents,
    tasks=my_tasks,
    process=Process.hierarchical,
    manager_llm=ChatOpenAI(model="gpt-4")
)
```