

# Challenge 1

## Crée ton assistant IA de lecture de document

### Objectif

Tu vas construire un **assistant intelligent** capable de **répondre à des questions sur un document PDF** (ou texte).

Ce projet te permettra de découvrir comment utiliser LangChain pour connecter un LLM à des données **personnelles et non connues du modèle**.

---

### Contexte

Un grand nombre de documents importants sont trop longs pour être lus en entier : rapports annuels, documents juridiques, articles de recherche, etc.

L'idée est de créer une **interface où l'on peut poser une question libre** (ex. "Que dit la conclusion ?") et recevoir une réponse **pertinente et sourcée**, comme si une IA avait lu le document pour toi.

---

### Ce que tu dois construire

- Un **chargeur** qui extrait le texte d'un document (PDF ou `.txt`)
  - Un système qui **divise le texte en morceaux utilisables** par un LLM
  - Un module qui transforme ces morceaux en **vecteurs sémantiques** et les stocke dans une base vectorielle
  - Une **interface minimale** (ligne de commande ou web) pour poser une question
  - Une réponse générée par un LLM, qui utilise **uniquement les passages pertinents du document** pour répondre
-

## ✓ Critères de validation

- Le système **répond de façon cohérente à plusieurs questions** sur le document
  - Il est capable de **citer les passages** ou indiquer les pages concernées
  - Le code est propre, bien structuré, reproductible
  - Bonus : l'utilisateur peut poser **plusieurs questions de suite**, et l'IA garde le contexte (mémoire)
- 

## 🧠 Concepts que tu vas explorer

- Indexation vectorielle (RAG : Retrieval-Augmented Generation)
  - Chaînes dans LangChain
  - Gestion du contexte
  - Prompt engineering
  - Interaction entre un LLM et des données externes
- 

## 🔧 Contraintes techniques

- Tu peux utiliser OpenAI ou tout autre fournisseur d'embeddings/LLM
  - La base vectorielle sera en local (ex. FAISS ou Chroma)
  - Interface : terminal ou Streamlit selon ton niveau
- 

## 🚀 Pistes pour aller plus loin (facultatif)

- Ajouter une **mémoire conversationnelle**

- Permettre d'**indexer plusieurs documents** et filtrer les réponses
- Ajouter une **recherche plein texte classique** (ex : keyword match)
- Déployer ton assistant avec une API (FastAPI)

## Challenge 2

### Challenge LangChain – Crée un chatbot IA avec mémoire conversationnelle

#### Objectif

Construis un **chatbot IA capable de tenir une conversation fluide**, en se souvenant de ce que l'utilisateur dit, même après plusieurs messages.

Tu apprendras à utiliser la **mémoire de LangChain** pour donner de la continuité à ton agent.

---

#### Contexte

Les LLM comme GPT-4 sont puissants, mais sans mémoire, ils **oublient tout** entre deux messages.

Dans ce challenge, tu vas créer un **assistant conversationnel** capable de :

- se souvenir du prénom de l'utilisateur,
- faire des références à des éléments mentionnés plus tôt,
- répondre comme un vrai assistant personnel, **cohérent et contextuel**.

---

## Ce que tu dois construire

- Un **chatbot** accessible en terminal ou dans une interface simple
- Un LLM relié via LangChain
- Une **mémoire conversationnelle** (type `buffer`, `summary`, ou `combined`)
- Une boucle d'interaction multi-tours où :
  - L'utilisateur écrit un message
  - L'IA répond en tenant compte de la conversation précédente

---

## Critères de validation

- L'IA **se souvient d'informations données plus tôt**
  - ex : "Je m'appelle Thomas" → "Enchanté Thomas"
  - ex : "Je suis prof de maths" → "As-tu des projets pédagogiques ?"
- Le **dialogue est cohérent sur 4 à 5 tours**
- Le code est bien structuré et commenté
- Bonus : tu choisis **un style de personnalité pour le bot** (ex : formel, amical, expert, etc.)

---

## Concepts que tu vas explorer

- Chaîne conversationnelle (`ConversationChain`)
- Mémoires (`ConversationBufferMemory`, `ConversationSummaryMemory`, etc.)

- Modèle de langage en mode dialogue
  - Prompt contextuel évolutif
- 





### Scénario test conseillé

- Étudiant : Salut, je m'appelle Julie
  - Bot : Bonjour Julie ! Que puis-je faire pour toi ?
  - Étudiant : J'aimerais apprendre à coder
  - Bot : Très bon choix Julie ! Tu veux commencer par Python ?
- 

### Contraintes techniques

- Tu peux utiliser [OpenAI](#), [ChatOpenAI](#) ou tout autre backend
  - La mémoire doit être stockée **en RAM** (pas besoin de base externe)
  - L'interface peut être en terminal, ou un front React/Streamlit si tu veux
- 

### Extensions possibles

-  Ajoute une **commande /reset** pour effacer la mémoire
-  Enregistre l'historique dans un fichier JSON
-  Connecte ton chatbot à un outil (ex : moteur de recherche, calculatrice, base de documents)
-  Donne une personnalité au bot avec un prompt initial "système" stylé

# Challenge 3

## **Projet avancé LangChain – Crée un Agent IA Personnel Multitâche**

---






### **Objectif**

Construire un **agent IA personnel** capable d'accomplir **différentes tâches à la demande**, en analysant le besoin de l'utilisateur, choisissant les bons outils, et gardant un **fil conducteur conversationnel**.

---

### **Fonctionnalités attendues**

L'utilisateur peut demander :

1.  De **répondre à des questions sur un document** (type RAG sur PDF)
2.  De **rechercher une information sur le web**
3.  De **faire des calculs simples ou complexes**
4.  De **gérer des rappels ou une TODO liste**
5.  De **converser normalement**, comme un chatbot

L'IA doit :

- Comprendre l'intention
  - Choisir l'action appropriée (tool)
  - Revenir à la conversation avec **contexte, style et cohérence**
- 

### **Architecture à mettre en place**

Composant	Rôle
Agent	Le cœur : décide quelle action faire
Tools	Fonctions utilisables par l'agent (web, documents, calcul, todo...)
Memory	Garde le fil d'une session
Retrieve r	Pour la partie RAG (lecture documentaire)
Interfac e	Terminal ou web (Streamlit ou React)

---

### Stack recommandée

- LangChain
  - OpenAI ou HuggingFace
  - FAISS ou Chroma pour la base vectorielle
  - DuckDuckGo ou SerpAPI pour la recherche web
  - Python standard pour outils perso
  - `langchain.agents` pour orchestrer tout ça
- 

### Exemple de dialogue

bash

CopierModifier

Utilisateur : J'ai besoin d'un résumé du fichier 'climat2024.pdf'

→ L'agent lit le PDF, crée un index, retrouve les passages clés et résume.

Utilisateur : Quel est le PIB du Japon cette année ?

→ L'agent appelle l'outil de recherche web et synthétise la réponse.

Utilisateur : Ajoute "Appeler Clara jeudi 10h" à ma TODO.  
→ Ajout dans une liste de rappels.

Utilisateur : Rappelle-moi mes tâches à venir  
→ Récupère les éléments enregistrés et les lit.

---

### ✓ Défis techniques à relever

- Créer plusieurs **Tools** personnalisés
- Gérer le **contexte conversationnel dynamique**
- Fournir un **historique cohérent**
- Savoir dire "je ne peux pas" proprement si l'outil est inopérant
- Bonus : faire un **dashboard avec logs des actions**

---

### 🧠 Concepts avancés couverts

- Agents LangChain (**initialize\_agent**, **Tool**, **AgentExecutor**)
- Planification dynamique (zero-shot-react)
- Mémoire conversationnelle + accès aux outils externes
- Chaînes composées (ex : RAG combiné à une mémoire)
- Séparation backend / interface

---

### 🚀 Bonus (si le groupe va vite)

- Ajouter un **système de permissions** pour certains outils



- Intégrer une API externe (comme météo, calendriers, etc.)
- Déployer via FastAPI + frontend React/Next.js

---

## Organisation possible des livrables

bash

CopierModifier

 langchain-multitool-agent/

├─ agent.py

├─ tools/

| └─ todo\_tool.py

| └─ search\_tool.py

| └─ doc\_reader.py

├─ memory/

| └─ memory\_manager.py

├─ retriever/

| └─ index\_manager.py

├─ app.py # boucle d'interaction ou interface Streamlit

├─ README.md

└─ .env