LangChain
Orchestration et automatisation des LLM

N° de la lecture individuelle :	3
Semestre	5
Étudiant	DAVID Guillaume, 805_1F
Sujet	LangChain – Plateforme d'orchestration, d'intégration et Orchestration et automatisation des LLM

Support théorique

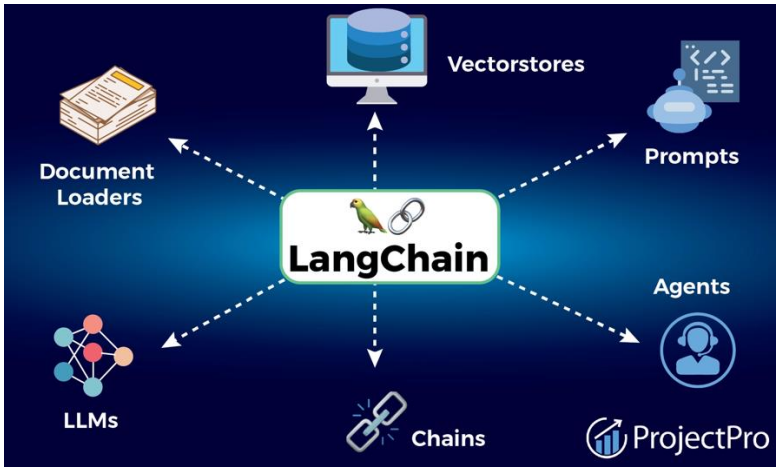
La recherche se base principalement sur la documentation officielle de LangChain, les exemples du dépôt GitHub travaillé avec ChatGPT.

Support théorique *1*
Introduction à LangChain *3*
Qu'est-ce que LangChain ? **3**
Exemple 3
Architecture de LangChain *4*
Composants principaux de LangChain **4**
Connecteurs et Intégrations *6*
Intégration avec des API (Hugging Face, OpenAI, Ollama, etc.) **6**
Agents autonomes : Détection des intentions et exécution des actions **7**
Utilisation des outils dans les Agents : Python, requêtes HTTP, et plus **7**
La mémoire dans LangChain *8*
Gestion du contexte et des conversations longues **8**
Types de mémoire **8**
Les limites de LangChain *9*
Problèmes de scalabilité et gestion des coûts **9**
Dépréciations et compatibilité des classes **9**
Bibliographie *10*

Introduction à LangChain

Qu’est-ce que LangChain ?

LangChain est une bibliothèque open-source conçue pour faciliter le développement d'applications basées sur des modèles de langage, en se concentrant sur l'orchestration et la gestion des interactions complexes. Son but principal est de simplifier l'intégration des grands modèles de langage (LLMs) dans des workflows complexes en permettant aux développeurs de créer des chatbots, des assistants virtuels, et des applications conversationnelles capables d'effectuer diverses tâches, comme la récupération d'informations, l'exécution de requêtes, ou l'automatisation de processus.



Son utilité dans l'intégration des pipelines utilisant les modèles de langage

L'intégration des modèles de langage dans des pipelines permet d'automatiser et d'enrichir des flux de travail complexes en exploitant la puissance des LLM (Large Language Models) pour comprendre, générer, et traiter des données textuelles.

Exemple

Le pipeline de traitement de contrats juridiques utilise un modèle de langage pour analyser automatiquement les contrats, en extrayant les clauses importantes et en générant des résumés pour les équipes légales. Le modèle peut également identifier les sections manquantes ou anormales grâce à des modèles préentraînés, accélérant ainsi le traitement des documents tout en réduisant les erreurs humaines. Cela permet aux juristes de se concentrer sur des tâches à plus forte valeur ajoutée.

Dans ce pipeline de traitement de contrats juridiques, LangChain intervient en orchestrant l'interaction entre le modèle de langage et d'autres composants du système. Grâce à son système de **Chains**, LangChain structure les différentes étapes du pipeline : extraction des clauses, génération des résumés et détection des anomalies. LangChain permet de définir un flux de travail qui enchaîne automatiquement ces tâches.

System Design :

1. **Modèle de langage (LLM)**
   LangChain gère l'appel au modèle de langage pour analyser et traiter le texte des contrats.
2. **Connecteurs aux bases de données/documentations**

LangChain peut se connecter à des bases de données juridiques ou de documents contractuels pour enrichir l'analyse avec des informations supplémentaires.

3. **Mémoire et prompts**
   Si la tâche nécessite un contexte de long terme (par exemple, des clauses récurrentes entre plusieurs contrats), LangChain utilise des mécanismes de mémoire pour maintenir la continuité dans l'analyse.
4. **Actions spécifiques**
   LangChain peut automatiser des actions, comme l'envoi de rapports ou d'alertes si une section manquante ou anormale est détectée.

LangChain simplifie la création du pipeline en orchestrant les différentes étapes et en gérant les interactions complexes entre les modèles et les données.

Architecture de LangChain

Composants principaux de LangChain

LangChain repose sur une architecture modulaire qui permet de construire des flux de travail complexes en intégrant des modèles de langage et d'autres outils de manière flexible.

- **Chains**
  Les *Chains* sont au cœur de LangChain. Elles permettent de lier plusieurs étapes dans un pipeline. Chaque étape est définie comme une opération spécifique, comme la génération de texte, la récupération d'informations dans une base de données, ou encore la manipulation de données. Les Chains orchestrent le flux en passant les résultats d'une étape à l'autre. Par exemple, dans un pipeline de traitement de contrat, une chaîne pourrait extraire du texte, le résumer, et identifier les clauses anormales en une seule séquence fluide.

  Un des avantages indéniables réside dans la compréhension des chains comme des objets structurés lors du codage des pipelines.

- **Agents**
  Les *Agents* sont des composants plus dynamiques, capables de prendre des décisions en fonction de l'entrée de l'utilisateur ou des résultats intermédiaires. Contrairement

aux Chains, qui suivent un chemin préétabli, les Agents utilisent des outils (comme des API, des bases de données ou du code Python) pour répondre à des intentions particulières. Ils décident de la meilleure action à exécuter pour accomplir une tâche. Par exemple, si un utilisateur pose une question juridique spécifique, un Agent peut décider de consulter une base de données légale, d'extraire une information pertinente, puis de générer une réponse contextualisée.

- **Prompts**
  Les *Prompts* sont des instructions fournies au modèle de langage pour lui donner des indications sur la tâche à accomplir. LangChain facilite la création de prompts dynamiques, en les adaptant en fonction du contexte ou des données intermédiaires. Par exemple, dans un pipeline de révision de contrat, un prompt pourrait demander au modèle de langage de "Résumer les principales clauses de ce contrat" ou "Identifier les risques potentiels dans ce texte." La flexibilité des prompts permet d'ajuster la tâche à l'étape actuelle du flux de travail.

- **Memory**
  La *Memory* permet à LangChain de stocker et de rappeler des informations d'une session à l'autre ou au sein d'une même session. Elle est essentielle pour les applications conversationnelles où il est important de maintenir le contexte, comme dans un chatbot. Par exemple, si un utilisateur discute de plusieurs contrats avec le chatbot sur une longue période, LangChain peut se souvenir des contrats précédemment analysés et faire référence à eux lorsque c'est pertinent. Il existe plusieurs types de mémoire, comme la *Buffer Memory* (qui stocke l'intégralité de la conversation) et la *Summary Memory* (qui résume les échanges).

Orchestration des flux de travail

LangChain orchestre les flux de travail en combinant ces différents composants dans une séquence logique et efficace, permettant des interactions complexes entre les étapes.

- **Séquençage avec Chains**
  LangChain structure le flux de travail en plusieurs étapes reliées, chaque étape étant gérée par une chaîne spécifique. Par exemple, un pipeline de traitement de contrat peut commencer par une chaîne d'extraction de texte, suivie d'une chaîne d'analyse des clauses importantes, puis d'une chaîne de génération de résumé. Les résultats d'une étape sont automatiquement transmis à l'étape suivante.

- **Agents pour la prise de décision dynamique (logique)**
  Contrairement aux Chains qui suivent un chemin prédéfini, les Agents apportent de la flexibilité en s’adaptant aux besoins en temps réel. Si une tâche requiert une action spécifique (comme l'extraction de données d'une API ou l'exécution de code Python), un Agent intervient et sélectionne l'outil adéquat pour accomplir cette tâche. Cela permet à LangChain de gérer des cas d'utilisation complexes, comme la réponse à des questions contextuelles ou l'exécution de calculs.

- **Mémoire pour le maintien du contexte :**
  Dans des flux de travail longs ou interactifs, LangChain utilise la mémoire pour maintenir le contexte de la session. Cela permet d'adapter les réponses et les actions en fonction de l'historique de l'utilisateur. Par exemple, un agent conversationnel peut adapter ses réponses en fonction des questions ou documents précédemment analysés, offrant une expérience plus cohérente et personnalisée.

- **Gestion flexible des Prompts**
  Les prompts jouent un rôle crucial dans la manière dont le modèle de langage interagit avec les données. LangChain génère des prompts dynamiques qui s'adaptent au contexte actuel. Cette flexibilité permet de guider le modèle de langage vers des tâches spécifiques tout en maintenant une performance optimale. Les prompts peuvent être ajustés pour différentes étapes du pipeline, qu'il s'agisse d'extraire des informations, de poser des questions précises, ou de générer des rapports.

Connecteurs et Intégrations

LangChain offre une grande flexibilité grâce à ses connecteurs, permettant une intégration fluide avec divers outils externes et bases de données.

Intégration avec des API (Hugging Face, OpenAI, Ollama, etc.)

LangChain peut se connecter à des modèles de langage via des API populaires comme Hugging Face, OpenAI ou Ollama. Cela permet d'accéder à des LLM de pointe pour générer, analyser ou comprendre du texte. L'intégration se fait facilement en définissant des prompts adaptés et en orchestrant les réponses au sein d'un flux de travail.

Intégration avec des bases de données vectorielles (Milvus, Pinecone, etc.)

LangChain permet également l'intégration avec des bases de données vectorielles telles que Milvus ou Pinecone. Ces bases stockent des embeddings, c'est-à-dire des représentations vectorielles du texte, ce qui permet de réaliser des recherches sémantiques avancées. Cela est particulièrement utile pour retrouver des informations pertinentes dans de grandes quantités de données textuelles.

Un exemple concret serait l'intégration de LangChain avec Milvus pour la recherche sémantique. Dans ce cas, un utilisateur pourrait poser une question complexe, et LangChain interrogerait la base de données vectorielle pour retrouver les documents ou informations les plus pertinentes en fonction du sens (plutôt que des correspondances exactes de mots-clés). Cela permet une recherche plus intelligente et contextuelle dans des systèmes d'information.

Agents et Automatisation

Les *Agents* dans LangChain jouent un rôle clé dans la gestion de flux de travail dynamiques et l'exécution de tâches automatisées. Contrairement aux *Chains* qui suivent un processus prédéfini, les agents sont capables de prendre des décisions en temps réel, de détecter des intentions spécifiques dans les requêtes utilisateur et d'exécuter des actions appropriées en fonction du contexte. Leur souplesse en fait un élément central pour des systèmes plus intelligents et autonomes.

Agents autonomes : Détection des intentions et exécution des actions

Les agents autonomes de LangChain sont capables d'analyser l'entrée de l'utilisateur pour détecter des intentions particulières et décider quelle action doit être effectuée en conséquence. Ces agents ne se contentent pas de suivre un chemin fixe comme dans les *Chains* ; ils évaluent continuellement les informations reçues pour choisir le meilleur chemin d'action.

- **Détection d'intention**
  Les agents peuvent comprendre l'intention derrière une question ou une commande utilisateur grâce à l'analyse contextuelle offerte par les modèles de langage. Par exemple, un utilisateur pourrait poser une question juridique ou demander une action particulière (ex. : "Quelles sont les prochaines étapes dans ce contrat ?" ou "Envoie-moi un résumé de ce document"). L'agent identifie automatiquement ce que l'utilisateur veut et exécute la tâche correspondante.

- **Exécution d'actions**
  Une fois l'intention détectée, l'agent est capable de déclencher des actions spécifiques. Par exemple, si l'intention est de "récupérer des données", l'agent pourrait interroger une base de données pour extraire les informations pertinentes. S'il s'agit de "générer un rapport", l'agent pourrait demander à un modèle de langage de créer un résumé ou un document structuré en fonction des besoins de l'utilisateur.

Utilisation des outils dans les Agents : Python, requêtes HTTP, et plus

Les agents dans LangChain sont conçus pour être très flexibles. Ils peuvent utiliser une variété d'outils pour accomplir des tâches précises, en fonction des besoins du flux de travail.

- **Exécution de code Python**
  Les agents peuvent exécuter du code Python directement pour automatiser des actions complexes. Cela est particulièrement utile pour les tâches qui nécessitent des calculs ou des manipulations de données qui vont au-delà des capacités des modèles de langage.

Exemple : Un agent pourrait répondre à une question financière en exécutant un script Python qui calcule des projections ou des statistiques, puis renvoyer les résultats sous forme de texte ou de graphiques.

- **Envoi de requêtes http**
  Les agents peuvent interagir avec des API externes en envoyant des requêtes HTTP. Cela permet de récupérer des informations en temps réel ou d'exécuter des actions via des services externes. Par exemple, un agent pourrait envoyer une requête à une API de météo pour récupérer les prévisions à un endroit donné, ou à une API financière pour obtenir les dernières données boursières. Cette capacité à interagir avec des services externes permet aux agents de LangChain d'aller au-delà du texte et de s'intégrer dans des écosystèmes plus larges.

Exemple : Si l'agent détecte une intention de "vérifier la météo", il pourrait envoyer une requête HTTP à un service météo comme OpenWeatherMap pour récupérer les prévisions, puis formater la réponse de manière à la présenter à l'utilisateur.

- **Utilisation d'autres outils personnalisés**
  LangChain permet aux développeurs de créer et d'intégrer des outils personnalisés que les agents peuvent utiliser. Cela pourrait inclure des systèmes internes spécifiques à l'entreprise, comme des bases de données propriétaires, des CRM, ou des ERP. L'agent peut ainsi déclencher des actions à l'intérieur de systèmes spécifiques, créant des workflows sur mesure adaptés aux besoins uniques d'une organisation.

Exemple : Dans un environnement de gestion de projet, un agent pourrait interagir avec un CRM pour récupérer les informations d'un client et générer automatiquement un rapport de suivi, ou encore créer une tâche dans un système de gestion de projet comme Jira.

La mémoire dans LangChain

La mémoire dans LangChain est un composant essentiel pour maintenir le contexte des interactions, notamment dans des applications nécessitant des conversations longues ou des suivis d'interaction. Elle permet aux agents de se souvenir des échanges précédents, offrant ainsi une expérience utilisateur plus fluide et cohérente.

Gestion du contexte et des conversations longues

La gestion du contexte est cruciale pour les applications interactives, telles que les chatbots ou les assistants virtuels. La mémoire permet aux agents de conserver des informations pertinentes au fil des interactions, évitant ainsi de recommencer à zéro à chaque nouvelle question. Cela améliore l'efficacité des échanges et la satisfaction de l'utilisateur. Par exemple, si un utilisateur discute d'un projet sur plusieurs sessions, la mémoire aide l'agent à rappeler les détails discutés précédemment, comme les échéances ou les points de blocage, ce qui facilite la continuité des échanges.

Types de mémoire

LangChain propose plusieurs types de mémoire, chacun adapté à des besoins spécifiques :

- **Buffer Memory**
  Cette mémoire stocke l'intégralité de la conversation actuelle, enregistrant chaque échange entre l'utilisateur et l'agent. Cela permet à l'agent d'avoir accès à toutes les informations de la session en cours, facilitant ainsi la compréhension du contexte immédiat. Cependant, cette méthode peut devenir inefficace pour des conversations très longues, car la taille de la mémoire peut rapidement croître.

- **Résumé Memory**
  Ce type de mémoire crée un résumé des interactions précédentes au lieu de conserver l'intégralité de la conversation. Cela réduit l'espace mémoire utilisé tout en préservant les informations essentielles. L'agent peut utiliser ces résumés pour se rappeler des points clés et des décisions prises dans des échanges antérieurs.

- **Conversation Memory**
  La mémoire conversationnelle est conçue pour maintenir un contexte tout en adaptant le niveau de détail stocké. Elle peut rappeler les informations importantes sans conserver chaque échange. Ce type de mémoire est particulièrement utile dans des

scénarios où certaines informations sont cruciales, tandis que d'autres peuvent être omises.

Les limites de LangChain

Malgré ses nombreuses capacités et avantages, LangChain présente également des défis et des limitations qui doivent être pris en compte lors de son utilisation dans des applications réelles.

Problèmes de scalabilité et gestion des coûts

L'un des principaux défis de LangChain réside dans sa scalabilité. À mesure que le volume d'interactions ou de données à traiter augmente, les exigences en matière de ressources (calcul, mémoire) peuvent croître de manière exponentielle. Cela peut entraîner des coûts élevés, notamment si des API tierces sont utilisées pour des modèles de langage. Les utilisateurs doivent donc planifier soigneusement l'architecture de leurs systèmes pour garantir qu'ils peuvent gérer la charge croissante tout en contrôlant les coûts associés.

Dépréciations et compatibilité des classes

LangChain est en constante évolution, ce qui peut entraîner des dépréciations de certaines classes ou fonctionnalités (comme celles liées à HuggingFaceHub). Ces changements peuvent poser des problèmes de compatibilité pour les développeurs qui ont construit leurs applications autour de versions spécifiques. Il est essentiel de rester informé des mises à jour et des changements de l'API pour garantir que le code reste fonctionnel et compatible avec les dernières versions de LangChain.

Bibliographie

https://raw.githubusercontent.com/davidmigloz/langchain_dart/main/docs/img/langchain.dart.png
https://daxg39y63pxwu.cloudfront.net/images/blog/langchain/LangChain.webp