

PRÉSENTATION SUR LANGGRAPH

Présenté par :

• BOUABDALLAH Aymane

• LAMKADDEM Ayoub

Encadré par :

✓ Mme. BENTALEB Asmae

INTRODUCTION

LangGraph est un **cadre open-source** développé pour faciliter la conception, l'orchestration et la coordination d'agents d'intelligence artificielle interconnectés à travers un graphe.

Il permet de créer des **flux de travail dynamiques et adaptatifs**, où chaque nœud du graphe représente un agent spécialisé, capable de raisonner, d'échanger des informations et de prendre des décisions basées sur des objectifs spécifiques.

LangGraph repose sur LangChain, mais ajoute la **capacité de gérer des architectures multi-agents complexes**, où l'on peut modéliser non seulement le cheminement logique des tâches, mais aussi les interactions intelligentes entre les agents.



PLAN

 Définition de LangGraph

 Architecture et composants

 Fonctionnalités principales

 Comparaison avec LangChain

 Cas d'utilisation

 Exemples d'implémentation

 Workflows multi-agents

 Performances et avantages

DÉFINITION DE LANGGRAPH

LangGraph est une extension du framework LangChain, conçue pour permettre la création d'agents IA organisés sous forme de graphes d'état. C'est un cadre d'orchestration **stateful** qui apporte un contrôle supplémentaire aux flux de travail des agents, facilitant la conception d'applications IA génératives complexes et interactives.

 Architecture orientée **graphe**, où chaque nœud représente une étape ou un comportement de l'agent

 Permet des **transitions conditionnelles** définissant des flux complexes, incluant boucles et bifurcations

 Gestion d'état partagé entre différents composants pour maintenir le contexte

 Facilite la création de systèmes **multi-agents** où différentes entités collaborent

ARCHITECTURE DE LANGGRAPH



Nœuds : Fonctions ou Agents

Les nœuds représentent les unités de traitement. Ce sont généralement des fonctions simples ou des agents IA capables d'exécuter une tâche précise, comme générer une réponse ou analyser une donnée.



Arêtes : Transitions Conditionnelles

Les arêtes définissent les transitions entre les nœuds. Elles peuvent être fixes ou conditionnelles, permettant des flux de travail dynamiques avec des boucles et des décisions.



État Partagé (Shared State)

Une mémoire accessible et modifiable par tous les nœuds du graphe. Contient les informations accumulées pour garantir la cohérence du raisonnement global.



Structure de Graphe

L'architecture orientée graphe offre une plus grande souplesse pour modéliser des flux complexes, incluant des boucles, des bifurcations ou des cycles de décision.

FONCTIONNALITÉS PRINCIPALES DE LANGGRAPH

Cycles et Branches

Permet d'implémenter des boucles et des conditions dans les applications, facilitant la création de flux de travail complexes et adaptatifs qui peuvent revenir sur des étapes précédentes.

Gestion d'État Automatique

Suivi et persistance automatiques des informations entre plusieurs interactions, assurant la cohérence du contexte et des données partagées entre les agents.

Human-in-the-Loop

Support natif pour les workflows avec intervention humaine, permettant une collaboration efficace entre agents IA et utilisateurs à des points de décision critiques.

Voyage dans le Temps

Capacité à revenir à des états précédents du système, facilitant le débogage, les tests et l'exploration de différentes voies de raisonnement.

Tolérance aux Pannes

Architecture robuste capable de gérer les erreurs et les exceptions sans compromettre l'intégrité du système, assurant la fiabilité des applications.

Systèmes Multi-Agents

Orchestration native de multiples agents spécialisés qui peuvent collaborer, communiquer et se coordonner pour résoudre des problèmes complexes.

LANGGRAPH VS LANGCHAIN

LangGraph

- ✓ Architecture orientée **graphe** pour workflows complexes et cycliques
- ✓ Gestion d'état avancée avec **mémoire persistante** entre les étapes
- ✓ Support natif pour **human-in-the-loop** et interventions humaines
- ✓ Optimisé pour la **coordination** de multiples agents spécialisés
- ✓ Facilite le **débogage** avec fonctionnalité de "voyage dans le temps"

LangChain

- ✓ Architecture **modulaire** pour chaînes de traitements linéaires
- ✓ Large **écosystème** de connecteurs et d'intégrations prêts à l'emploi
- ✓ Plus **simple** à mettre en œuvre pour des cas d'usage basiques
- ✓ Davantage de **documentation** et d'exemples communautaires
- ✓ Idéal pour des **prototypes rapides** et applications simples

QUAND UTILISER CHAQUE FRAMEWORK ?



Utilisez **LangGraph** pour des agents intelligents avec mémoire et logique récursive



Utilisez **LangChain** pour des pipelines de traitement simples et directs



LangGraph est une **extension** de LangChain, les deux peuvent être utilisés ensemble

CAS D'UTILISATION DE LANGGRAPH



Chatbots Avancés

Développement d'assistants conversationnels capables de gérer des conversations complexes avec mémoire contextuelle, de prendre des décisions adaptatives et d'intégrer des sources d'information externes tout en maintenant une cohérence dans les échanges.



Systèmes Multi-Agents

Conception de systèmes où plusieurs agents spécialisés collaborent pour résoudre des problèmes complexes, comme la génération de contenu avec des équipes d'agents rédacteurs, chercheurs et vérificateurs qui interagissent pour produire un résultat optimal.



Automatisation de Flux de Travail

Création d'outils qui automatisent des processus métier complets avec des étapes conditionnelles et cycliques, comme la planification de voyages ou la gestion de projets, où le système peut revenir sur des décisions précédentes selon les contraintes.



Support Client Intelligent

Développement d'agents de support client qui peuvent diagnostiquer des problèmes complexes, guider les utilisateurs à travers des solutions multi-étapes, et transférer intelligemment à un humain lorsque nécessaire avec tout le contexte pertinent.



Recherche et Synthèse

Création d'agents capables de mener des recherches approfondies sur un sujet, d'analyser diverses sources, et de synthétiser l'information dans un format cohérent, avec possibilité de vérification factuelle et d'exploration itérative.



Assistant de Développement

Conception d'assistants pour les développeurs qui peuvent analyser du code, suggérer des améliorations, générer des tests et expliquer des parties complexes, tout en maintenant la compréhension du contexte global du projet et des interactions entre composants.

IMPLEMENTATION DE LANGGRAPH

1 Installation

Installer LangGraph via pip et configurer l'environnement avec les dépendances nécessaires.

2 Définition des Nœuds

Créer les fonctions qui représenteront les noeuds du graphe, avec leurs logiques de traitement.

3 Construction du Graphe

Définir la structure du graphe et les transitions entre les nœuds selon les conditions spécifiées.

4 Exécution

Lancer le graphe avec un état initial et gérer les résultats à travers les différentes étapes.

```
# streamlit : pour créer l'interface web.
import streamlit as st
# pandas : pour manipuler ton fichier CSV.
import pandas as pd
# seaborn et matplotlib : pour faire les graphes.
import seaborn as sns
import matplotlib.pyplot as plt
# PCA, StandardScaler, KMeans : pour réduire les dimensions (PCA), normaliser les données (Scaler) et créer des groupes (clustering).
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
# LangGraph : pour organiser les étapes de traitement comme un pipeline intelligent.
from langgraph.graph import StateGraph
# TypedDict : pour définir un état structuré (le format des données partagées entre les étapes).
from typing import TypedDict
```

1. Définir la structure d'état

```
● ● ●  
# ----- 1. Shared State -----  
class DataState(TypedDict):  
    df: pd.DataFrame  
    processed_df: pd.DataFrame
```

2. Créer les fonctions de nœuds

```
● ● ●  
  
def load_data(state: DataState) -> DataState:  
    df = state["df"]  
    if "Unnamed: 0" in df.columns:  
        df.drop(columns=["Unnamed: 0"], inplace=True)  
    df["Date"] = pd.to_datetime(df["Date"], dayfirst=True,  
                               errors='coerce')  
    df.dropna(subset=["Date"], inplace=True)  
    state["df"] = df  
    return state
```

3. Construire le graphe

```
● ● ●  
builder = StateGraph(DataState)  
builder.add_node("Load Data", load_data)  
builder.add_node("Inspect", inspect_data)  
builder.add_node("Detect", detect_patterns)  
builder.add_node("Visualize", visualize)  
builder.add_node("Recommend", recommend)
```

4. Définir les transitions

```
● ● ●  
builder.set_entry_point("Load Data")  
builder.add_edge("Load Data", "Inspect")  
builder.add_edge("Inspect", "Detect")  
builder.add_edge("Detect", "Visualize")  
builder.add_edge("Visualize", "Recommend")
```

5. Compiler le graphe et Exécuter avec une entrée

```
graph = builder.compile()
# ----- 4. Streamlit App -----
st.title("📊 Analyse des Performances Académiques")
uploaded_file =
st.file_uploader("📁 Charger un fichier CSV", type="csv")
if uploaded_file is not None:
    df = pd.read_csv(uploaded_file)
    if st.button("🚀 Lancer l'analyse"):
        initial_state = {"df": df}
        graph.invoke(initial_state)
```

WORKFLOWS MULTI-AGENTS AVEC LANGGRAPH



Coordination Centralisée

Un agent coordinateur peut distribuer les tâches aux agents spécialisés, suivre leur progression et synthétiser leurs résultats, agissant comme un chef d'orchestre.



Communication Inter-Agents

Les agents peuvent partager des informations via l'état partagé, permettant une collaboration fluide et la transmission de connaissances entre les différentes étapes du processus.



Cycles de Rétroaction

LangGraph permet de créer des boucles de feedback entre les agents, comme la relation rédacteur-critique, où le contenu peut être itérativement amélioré jusqu'à atteindre la qualité désirée.



Spécialisation des Agents

Chaque agent peut être optimisé pour une tâche spécifique avec des prompts, outils et modèles adaptés, augmentant l'efficacité globale du système par rapport à un agent unique généraliste.

GESTION D'ÉTAT DANS LANGGRAPH



Mémoire Partagée

Permet aux agents de construire sur les résultats précédents et de maintenir un contexte cohérent tout au long de l'exécution.

Persistance d'État

L'état peut être sauvegardé et restauré, permettant de reprendre l'exécution ou d'analyser le processus ultérieurement.

Flux Conditionnels

Le routage entre les nœuds peut être dynamique, basé sur l'état actuel, permettant des workflows adaptatifs.

INTÉGRATION D'OUTILS AVEC LANGGRAPH



Vectorisation et RAG

Intégration avec des bases de données vectorielles pour la récupération de documents pertinents et la génération augmentée par récupération.

APIs Externes

Connection à des services tiers comme météo, actualités, ou outils SaaS pour enrichir les capacités des agents avec des données en temps réel.

Exécution de Code

Capacité à écrire, analyser et exécuter du code dynamiquement pour effectuer des calculs ou automatiser des tâches complexes.

Bases de Données

Interaction avec différents types de bases de données pour stocker, récupérer et mettre à jour des informations persistantes.

Vision et Multimodalité

TraITEMENT d'images et de contenus multimodaux pour enrichir la compréhension contextuelle des agents.

Systèmes Locaux

Interaction avec le système d'exploitation ou les ressources locales pour exécuter des opérations sur l'infrastructure hôte.

MEILLEURES PRATIQUES AVEC LANGGRAPH

Conception Modulaire

Divisez votre graphe en nœuds simples et spécialisés plutôt qu'en quelques nœuds complexes. Chaque nœud devrait avoir une responsabilité unique, facilitant le test et la réutilisation.

Conseil : Commencez par identifier les étapes indépendantes de votre flux et transformez-les en nœuds distincts.

Gestion Efficace de l'État

Définissez clairement la structure de votre état avec `TypedDict` et limitez les modifications à ce qui est nécessaire. N'incluez pas de données volumineuses ou redondantes dans l'état partagé.

Conseil : Utilisez des clés descriptives et cohérentes dans votre dictionnaire d'état pour faciliter le suivi et le débogage.

Transitions Conditionnelles Claires

Concevez des conditions de routage précises qui évitent les ambiguïtés. Documentez la logique de transition pour chaque arête du graphe afin de faciliter la compréhension du flux.

Conseil : Visualisez votre graphe avant l'implémentation pour identifier les chemins possibles et les conditions de branchement.

Planification du Débogage

Intégrez des points de journalisation stratégiques dans vos nœuds pour suivre l'évolution de l'état. Utilisez les fonctionnalités de voyage dans le temps pour identifier les problèmes dans la logique.

Conseil : Ajoutez des métadonnées à chaque transition d'état pour expliquer les décisions prises par le système.

Coordination Multi-Agents

Pour les systèmes multi-agents, définissez clairement les protocoles de communication et la hiérarchie entre agents. Utilisez un agent coordinateur pour gérer les interactions complexes.

Conseil : Privilégiez l'état partagé pour les communications inter-agents plutôt que des mécanismes ad hoc pour maintenir la cohérence.

Optimisation des Performances

Minimisez les appels aux LLMs en mettant en cache les résultats fréquemment utilisés. Utilisez des modèles plus légers pour les tâches intermédiaires et réservez les modèles avancés pour les étapes critiques.

Conseil : Instrumentez votre graphe pour mesurer le temps passé dans chaque nœud et identifiez les goulots d'étranglement.

CHALLENGES ET PERSPECTIVES

⚠ Défis Actuels

- › Complexité d'orchestration pour les graphes très larges avec de nombreux agents
- › Difficulté à capturer et maintenir le contexte global à travers de longues séquences
- › Coûts computationnels élevés liés aux multiples appels de modèles

Opportunités Futures

- › Intégration avec des modèles multimodaux pour des agents capables de traiter texte, images et audio
- › Développement de patterns d'architecture standardisés pour les cas d'usage communs
- › Amélioration des outils de visualisation et monitoring des graphes complexes

Tendances d'Évolution

- › Vers des architectures hybrides combinant approches symboliques et neuronales
- › Intégration plus poussée avec des environnements de simulation pour le test des agents
- › Développement de formats collaboratifs entre agents humains et IA

Place dans l'Écosystème IA

- › LangGraph complète les frameworks LLM en ajoutant des capacités d'orchestration avancées
- › Bridge entre approches dirigées par les données et systèmes à base de règles
- › Facilite l'intégration des agents IA dans les systèmes d'entreprise existants

CONCLUSION

LangGraph représente une évolution majeure dans la conception d'applications IA, offrant un cadre structuré pour orchestrer des agents intelligents capables de résoudre des problèmes complexes grâce à son architecture orientée graphe et sa gestion d'état avancé.

MERCI DE VOTRE
ATTENTION