

LANGGRAPH: DESARROLLO DE AGENTES DE IA AVANZADOS

SKU: N / A | Categorías: [ADMINISTRACIÓN Y GESTIÓN](#), [INFORMÁTICA Y COMUNICACIONES](#), [Inteligencia artificial](#)

INFORMACIÓN DEL CURSO

Horas 35

Cod. 18

CURSO DE ESPECIALIZACIÓN

LangGraph: desarrollo de agentes de IA avanzados

Objetivos

Diseñar e implementar agentes avanzados de IA utilizando LangGraph y LLMs para desarrollar sistemas autónomos y escalables que integren razonamiento, memoria y colaboración multi-agente.

Contenidos

1 - Introducción a LangGraph e instalación

Introducción a LangGraph e instalación
Niveles de autonomía en aplicaciones de IA
¿Qué es LangGraph y qué nos ofrece?
Estructura fundamental: nodos, transiciones y estado
Integración de herramientas y extensibilidad
Gestión de memoria y máquina de estados
Ventajas que aporta LangGraph al desarrollo de agentes de IA
Instalación librerías y entorno LangGraph
Instalación de Python
Selección de entorno de desarrollo integrado (IDE)
Instalación de librerías y configuración del entorno de trabajo
Resumen

2 - Agente reflexivo en LangGraph

Agente reflexivo en LangGraph
Presentación y configuración inicial del agente reflexivo
Arquitectura base del agente reflexivo en LangGraph



Estructura de archivos del proyecto
Consideraciones sobre el flujo de desarrollo
Uso de LLM en la creación de agentes
Creación de las cadenas de generación y reflexión
Definición de prompts estructurados para generación y reflexión
Construcción práctica de las cadenas
Ejemplo básico de configuración
Construcción del grafo con LangGraph
Definición del esquema de estado
Creación y registro de nodos
Definición del flujo y las transiciones
Compilación y visualización del grafo
Ejemplo de ejecución (flujo típico)
Ventajas del enfoque con LangGraph
Análisis de trazas del agente reflexivo con LangSmith
¿Por qué es relevante el análisis de trazas en agentes reflexivos?
Integración básica de LangSmith en el agente reflexivo
Visualización e interpretación de las trazas
Consideraciones éticas y de privacidad
Resumen

3 - Agente reflexión con herramientas en LangGraph

Agente reflexión con herramientas en LangGraph
Presentación y configuración inicial del agente reflexión
Propósito y ventajas principales del agente reflexión
Aspectos clave en la configuración inicial
Flujo operativo básico
Creación de esquemas de salida Pydantic
Modelado de salidas de herramientas y flujos de reflexión
Definición de esquemas Pydantic clave
Aplicación práctica en el agente de reflexión
Configuración de las herramientas con ToolNode
Integración de herramientas en agentes mediante ToolNode
Importación y preparación del entorno
Creación de herramientas estructuradas
Registro de herramientas en ToolNode
Ventajas del enfoque modular con ToolNode
Creación de cadenas para el agente de respuesta y agente revisor
Importación de dependencias y esquemas
Definición de plantillas de prompt para agentes estructurados
Prompt para el agente de respuesta inicial (first responder)
Prompt para el agente revisor (revisor)
Construcción práctica de las cadenas (chains)



Ventajas de este enfoque estructurado
Construcción del grafo con LangGraph
Definición y registro de nodos en el grafo
Definición del flujo y las transiciones
Compilación y visualización del grafo
Consideraciones prácticas y control de iteraciones
Análisis de trazas del agente reflexión con LangSmith
Resumen

4 - Agentes ReAct en LangGraph

Agentes ReAct en LangGraph
Presentación y configuración inicial del agente ReAct
Principios y dinámica de un agente ReAct
Comparativa: Agente reflexión vs. Agente ReAct
Ventajas del enfoque ReAct en LangGraph
Elementos necesarios para la configuración inicial
Creación de herramientas del agente ReAct con ToolNode
Creación y registro de herramientas personalizadas
Incorporación de herramientas preexistentes
Lista y gestión unificada de herramientas
Vinculación del LLM con herramientas disponibles
Ejemplo sintético de definición y vinculación
Buenas prácticas en la configuración de herramientas
Implementación de nodos para agente ReAct
Nodo de razonamiento: interacción con el LLM
Nodo de herramientas: ejecución mediante ToolNode
Buenas prácticas en la definición de nodos
Construcción del grafo con LangGraph
Definición y ensamblaje del grafo ReAct
1. Importación de dependencias y configuración del entorno
2. Creación de nodos funcionales
3. Definición de la lógica de control de flujo
4. Ensamblado del grafo y definición del ciclo ReAct
5. Compilación y visualización del grafo
Análisis de trazas del agente ReAct con LangSmith
Integración de LangSmith en agentes ReAct
Resumen

5 - Agentes RAG avanzados con LangGraph

Agentes RAG avanzados con LangGraph
Presentación y configuración inicial del agente RAG
Características iniciales del agente RAG avanzado
Flujo operativo del agente RAG



Ingesta de datos en la base de datos vectorial

1. Preparación del entorno y carga de variables
2. Lectura del corpus documental
3. Segmentación semántica por encabezados
4. Chunking optimizado para embeddings
5. Creación y persistencia de la base de datos vectorial
6. Configuración del sistema de recuperación

Consideraciones clave

Creación del estado del grafo

Definiendo el estado del grafo RAG

Estructura propuesta para el estado

Atributos principales del estado

Ventajas del enfoque centralizado

Definición del nodo recuperador para obtener información relevante del RAG

Función y lógica del nodo recuperador

Conceptos clave y mejores prácticas

Construcción de filtro de relevancia para el RAG

Pipeline para la evaluación de relevancia basada en LLM

Componentes principales del filtro de relevancia

Implementación del nodo filtrador en LangGraph

Ventajas del enfoque y consideraciones prácticas

Implementación del nodo de búsqueda web con Tavily

Configuración de la herramienta de búsqueda web Tavily

Preparación inicial

Función del nodo de búsqueda web

Puntos clave de la implementación

Definición del nodo de generación

Composición de la cadena de generación RAG

Implementación del nodo de generación en el grafo RAG

Aspectos clave en el diseño del nodo de generación

Creación del grafo RAG inicial

Secuencia y conexiones del grafo

Compilación y visualización del grafo

Resumen

6 - Agentes RAG avanzados con LangGraph con autoevaluación de alucinaciones y enrutado

Agentes RAG avanzados con LangGraph con autoevaluación de alucinaciones y enrutado

Autoevaluación de alucinaciones

Modelo estructurado para la evaluación de alucinaciones

Construcción del prompt para instrucción evaluadora

Secuencia ejecutable de autoevaluación

Aspectos clave para una integración efectiva

Autoevaluación de respuesta correcta



Modelo estructurado para la evaluación de respuestas
Construcción del prompt evaluador
Secuencia ejecutable de la autoevaluación
Ventajas de la autoevaluación estructurada
Implementación de enrutado adaptativo en RAG
Principios del enrutado adaptativo en entornos RAG
Definición de funciones de decisión y nodos de enrutado
Diseño programático del grafo de flujo adaptativo
Ventajas y consideraciones prácticas
Resumen

7 - Persistencia e intervención humana en el flujo

Persistencia e Intervención humana en el flujo
Persistencia en LangGraph: memoria y checkpointing
Memoria de proceso vs. memoria conversacional
¿Qué es un checkpoint?
Ventajas de la persistencia en LangGraph
Persistencia: de chatbot a sistema
Intervención humana con Memory Saver e interrupciones (persistencia RAM)
Persistencia en RAM y el uso de MemorySaver
Interrupciones y puntos de intervención (interrupt_before)
Flujo típico de intervención humana: arquitectura y pasos clave
Ventajas estratégicas de la intervención humana mediante persistencia en RAM
Consideraciones prácticas
Persistencia con Sqlite Saver
Diferencias clave entre SqliteSaver y MemorySaver
Configuración e implementación de SqliteSaver
1. Preparación del entorno
2. Definición y registro del grafo
3. Creación y utilización de SqliteSaver
4. Gestión de las sesiones (thread_id) y reanudación
Ventajas estratégicas de SqliteSaver en LangGraph
Creación de agentes ReAct con persistencia y validación humana
Incorporación de persistencia con SqliteSaver
Validación humana condicionada: patrón de interrupción controlada
Flujo interno y lógica de enrutado
Actualización y persistencia del estado tras intervención
Ejemplo de control de flujo y persistencia
Resumen

8 - Ejecución asíncrona en LangGraph

Ejecución asíncrona en LangGraph
¿Qué es la ejecución asíncrona en LangGraph?



Por qué la asincronía es relevante en agentes basados en grafos

Concurrencia vs. paralelismo: matiz necesario

Beneficios operativos en escenarios típicos de agentes

Cuando la asincronía marca la diferencia

Implicaciones de diseño: estado, consistencia y coordinación

Implementación de asincronía en LangGraph

Componentes importados y rol en el grafo

Definición del estado: implicaciones para consistencia y coordinación

Nodos del grafo: contrato de entrada/salida y simulación de latencia

Construcción del grafo: topología y puntos de sincronización

Compilación, visualización y ejecución: artefactos operativos

Ejecución con estado inicial y configuración de hilo

Ejecución de ramas en paralelo con nodos adicionales

1. Estado compartido y estrategia de agregación
2. Nodo parametrizable: ReturnNodeValue
3. Definición de nodos
4. Transiciones: bifurcación, secuenciación interna y join
5. Qué se considera paralelo en este diseño
6. Evolución esperada del estado (agregación)
7. Visualización del grafo (Mermaid)
8. Ejecución con identificador de hilo (thread_id)

Pautas para extender este patrón sin introducir errores

Ejecución asíncrona con ramas condicionales

Estado del grafo y agregación de resultados

Nodos del grafo y comportamiento común

Enrutamiento condicional y validación de rutas

Convergencia de ramas y finalización

Compilación, visualización y ejecución

Resumen

9 - Ecosistema multiagente con LangGraph

Ecosistema multiagente con LangGraph

¿Qué ecosistemas de agentes podemos implementar y por qué son necesarios?

Por qué surge la necesidad de una arquitectura multiagente

Problema 1: demasiadas herramientas para un único agente

Problema 2: el contexto se vuelve demasiado complejo

Problema 3: se requieren distintas especializaciones

Beneficios principales de los sistemas multiagente

1. Modularidad
2. Especialización
3. Control

Presentación del proyecto multiagente

Agente supervisor



Agente clarificador
Responsabilidad del nodo
Construcción de mensajes e invocación del LLM
Actualización de estado y transición al supervisor
Agente buscador
Prompt de sistema y rol del agente
Creación e invocación del agente
Transición de estado y envío al nodo validador
Agente codificador
Prompt de rol y creación del agente
Ejecución y transición hacia validación
Agente validador
Lógica de evaluación y enrutado
Invocación del LLM y salida estructurada
Normalización de fin de flujo y actualización del estado
Creación del grafo y ejecución
Ejecución del grafo en modo streaming
Resumen

10 - Despliegue de soluciones con LangGraph

Despliegue de soluciones con LangGraph
¿Qué opciones de despliegue tenemos?
1. Despliegue en local mediante script
2. Despliegue en servidor local
3. Despliegue con Agent Server mediante imagen de Docker
4. Despliegue en la nube con LangSmith Deployments
Despliegue con LangSmith Deployments
Preparación del proyecto para un despliegue reproducible
Creación del deployment en LangSmith
Configuración de build y arranque
Validación del endpoint y pruebas operativas
Versionado, despliegues seguros y rollback
Control de acceso y consideraciones de seguridad
Integración con flujo de trabajo (CI/CD) orientado a despliegues
Resumen

Tal vez te interese este curso: [INTELIGENCIA ARTIFICIAL: FUNDAMENTOS DE APRENDIZAJE AUTOMÁTICO Y OPORTUNIDADES DE NEGOCIO](#)

O quizá este otro: [n8n desde cero: automatización de flujos y agentes IA sin código Nuevo](#)

Síguenos en: [Instagram](#)



