

APIs, Agentes y Extracción Estructurada con LlamalIndex y Pydantic

Fundación

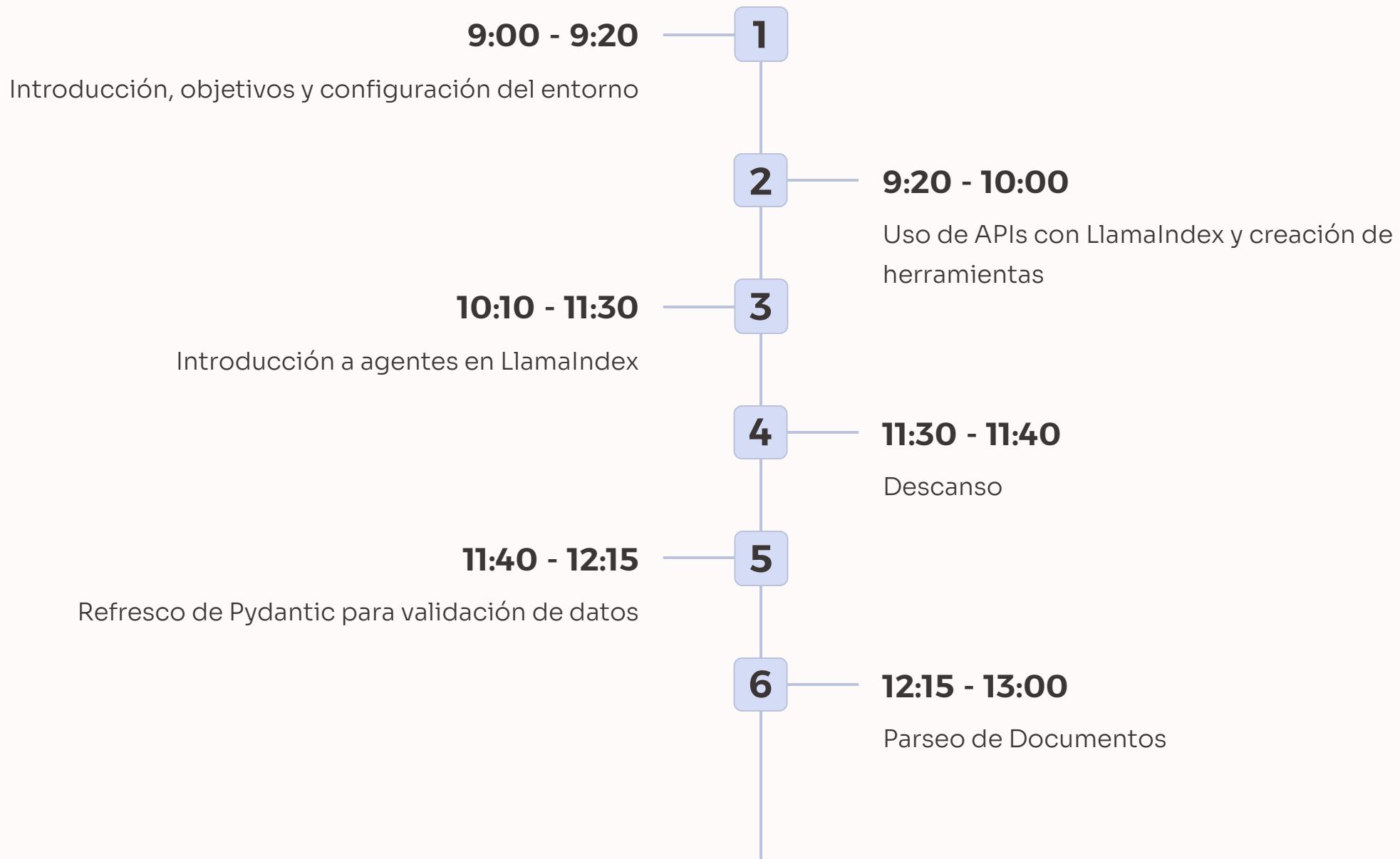
GOODJOB



Sesión práctica de 4 horas Fundación GoodJob

Made with **GAMMA**

Agenda de la Sesión



Resultados de Aprendizaje



Al finalizar esta sesión, serás capaz de:

Crear y utilizar herramientas (Tools) basadas en APIs externas con LlamalIndex

Implementarás FunctionTools que encapsulen llamadas a APIs como OpenWeather

Implementar agentes con LlamalIndex que combinen fuentes de datos

Diseñarás sistemas multi-agentes capaces de utilizar múltiples herramientas según el contexto

Validar datos con Pydantic aplicando validadores personalizados

Crearás modelos Pydantic robustos con validación avanzada para garantizar la calidad de los datos

Desarrollar un sistema completo de extracción y enriquecimiento de información

Construirás un agente capaz de extraer datos estructurados de documentos, enriquecerlos y validarlos

Requisitos Previos y Configuración



Conocimientos

- Python básico (funciones, clases)
- Manejo de entornos virtuales (conda/venv)
- Conceptos básicos de APIs y HTTP
- Nociones de JSON y diccionarios en Python

Herramientas

- Python 3.12 o superior
- VSCode o editor de preferencia
- Terminal/Consola para comandos pip

Preparación

Ejecuta estos comandos para instalar las dependencias:

```
# Crear entorno virtual  
python -m venv venv  
source venv/bin/activate # En Windows:  
venv\Scripts\activate
```

```
# Instalar dependencias  
pip install -r requirements.txt
```

Crea un fichero .env con tus claves de API:

```
OPENWEATHER_API_KEY=tu_clave_aquí  
NEWS_API_KEY=tu_clave_aquí  
HF_API_KEY=tu_clave_aquí
```

Bloque 1 : Llama-index



Uso de APIs con Llama-index (9:00 - 10:00)

- **Objetivo:** comprender cómo consumir APIs externas y cómo integrarlas con *Llama-index*.
- **Contenido:**
 - a. Breve repaso: ¿Qué es *Llama-index*?
 - b. Componentes clave para trabajar con APIs (Tools, Query Engines, Connectors).
 - c. Ejemplo simple: consumir una API REST y usar la respuesta como contexto para un LLM.
 - d. Ejercicio guiado: integración con una API pública (ej. OpenWeather y NewsAPI).

Agentes con Llama-index (10:00 - 11:30)

- **Objetivo:** aprender a construir agentes autónomos que combinen varias fuentes y herramientas.
- **Contenido:**
 - a. Concepto de agente y *tool* en *Llama-index*.
 - b. Arquitectura de un agente básico.
 - c. Ejemplo: agente que consulta varias APIs y responde de forma razonada.
 - d. Ejercicio guiado: construir un sistema multi-agente que llame a la API creada en el bloque anterior y nuestra propia base de datos vectorial.



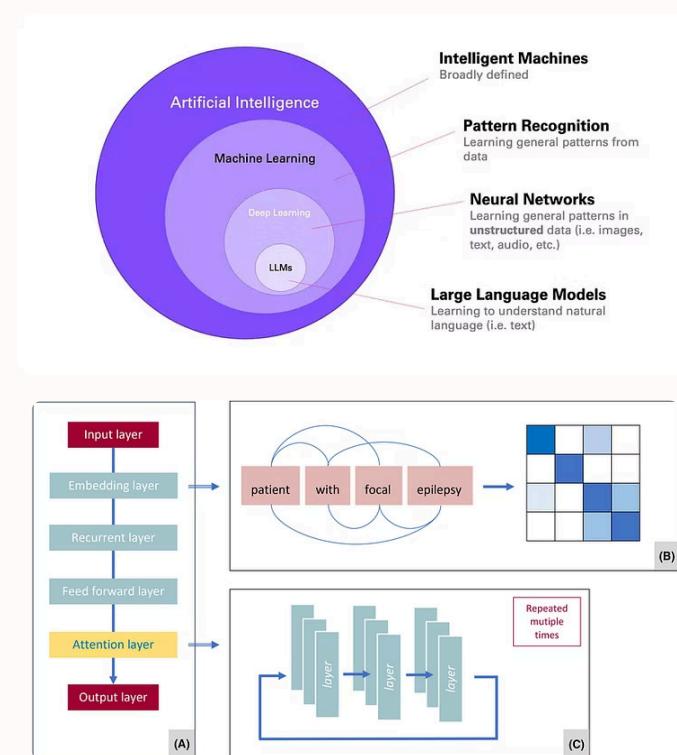
Uso de APIs con LlamalIndex



¿Qué son los LLMs (Large Language Models)?

Los LLMs son modelos de lenguaje avanzados entrenados con grandes volúmenes de datos. Aprenden patrones, gramática y conocimiento general a partir de este entrenamiento.

Es crucial entender que estos modelos se entran con **datos supervisados y finitos** hasta una fecha de corte específica. Esto significa que su "conocimiento" está limitado a la información presente en sus datos de entrenamiento.



Limitaciones y la Necesidad de Herramientas

Por sí solos, los LLMs:

- No tienen acceso a datos en tiempo real:** Su información no se actualiza automáticamente con eventos recientes o cambios constantes en el mundo.
- No pueden interactuar con sistemas externos:** No pueden consultar bases de datos, enviar emails, ni obtener información de servicios web.
- Su conocimiento es estático:** Las respuestas que generan se basan únicamente en la información "horneada" en sus pesos durante el entrenamiento.

Aquí es donde entra la importancia de las **Tools (herramientas)** y plataformas como **LlamalIndex**. Al integrar APIs externas, podemos dotar a los LLMs de la capacidad de:

- Acceder a información actualizada y en tiempo real.
- Realizar acciones específicas fuera de su propio conocimiento (ej., consultar el clima, obtener noticias, interactuar con servicios).
- Enriquecer sus respuestas con datos dinámicos y contextuales, trascendiendo las limitaciones de su entrenamiento inicial.

Uso de APIs con LlamalIndex



¿Qué son las Tools en LlamalIndex?

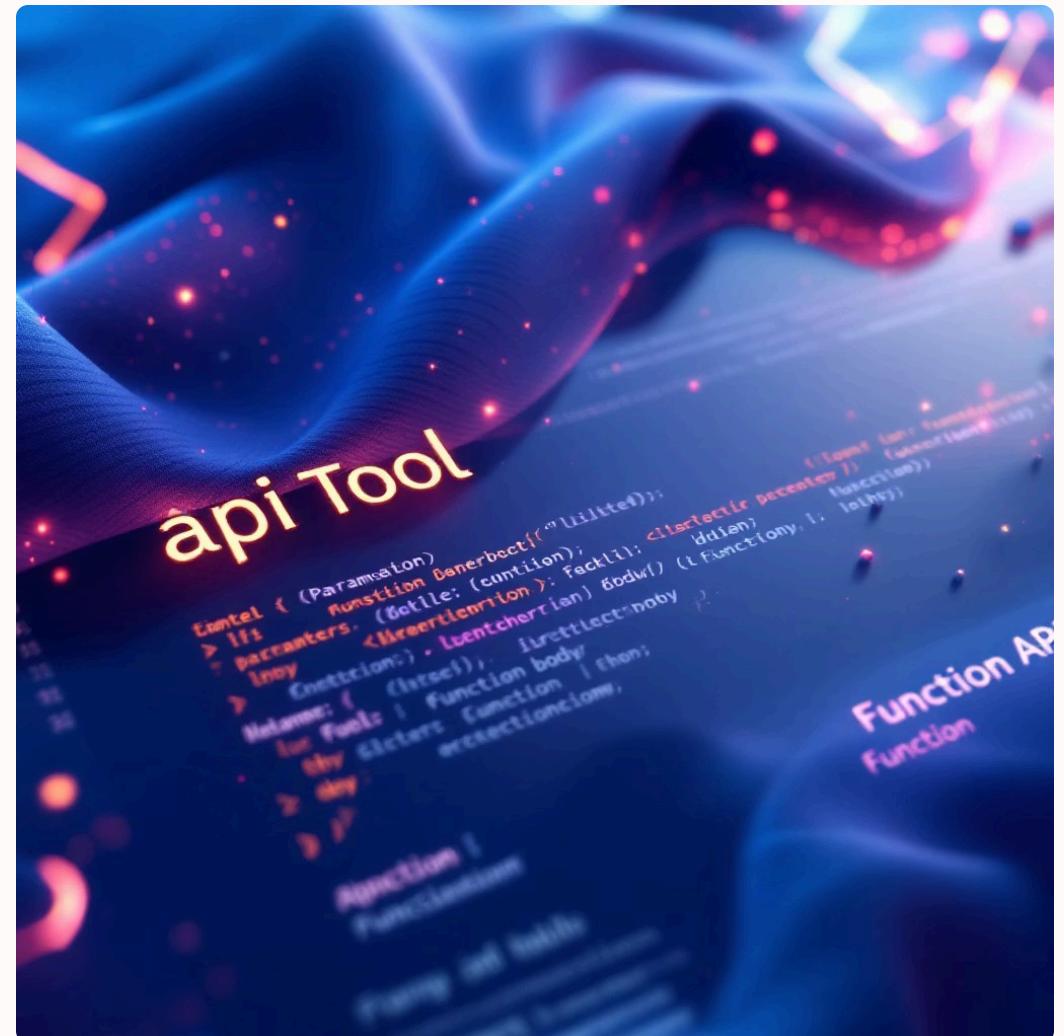
Las Tools son funciones encapsuladas que pueden ser utilizadas por agentes para interactuar con sistemas externos o realizar tareas específicas.

- Permiten ampliar las capacidades de los agentes
- Pueden invocar APIs externas, consultar bases de datos, o ejecutar funciones personalizadas

Son la **abstracción central** para construir sistemas agentivos: son interfaces de “APIs para agentes” (no para humanos). Puedes definir tanto una **Tool** como un **ToolSpec** (un paquete de funciones) según lo que necesites.

El **nombre** de la tool, su **descripción** y el **esquema de argumentos** influyen directamente en cómo el LLM decide llamarla (vital para function calling). Ajustar estos metadatos cambia de forma notable el comportamiento

Elementos clave de una Tool



Una buena herramienta debe tener:

1. Nombre descriptivo y claro
2. Descripción detallada de su función
3. Parámetros bien documentados
4. Manejo adecuado de errores



Hugging Face

Hugging Face: Proveedor de Inferencia

Hugging Face se ha consolidado como la plataforma líder para modelos de IA de código abierto, ofreciendo una vasta colección de modelos pre-entrenados, datasets y herramientas.

Como proveedor de inferencia, Hugging Face permite ejecutar modelos de lenguaje grandes (LLMs) y otros modelos de IA de forma eficiente, sin necesidad de gestionar infraestructuras complejas. Esto es fundamental para:

1

Probar y desplegar modelos de código abierto.

2

Integrar capacidades de IA avanzadas en aplicaciones.

3

Realizar inferencia con modelos optimizados para diversas tareas.

En el contexto de LlamalIndex, [Hugging Face](#) puede ser utilizado como un **proveedor de LLM**, permitiendo a los agentes y herramientas interactuar con modelos alojados en su plataforma, ampliando así las opciones más allá de modelos propietarios como OpenAI.

¡Manos a la Obra!

Es momento de aplicar los conceptos que hemos cubierto sobre el uso de APIs con LlamaIndex.



Abre el siguiente Jupyter Notebook para comenzar con los ejercicios prácticos:

01_uso_apis_llamaindex.ipynb

¡Prepárate para programar y experimentar!

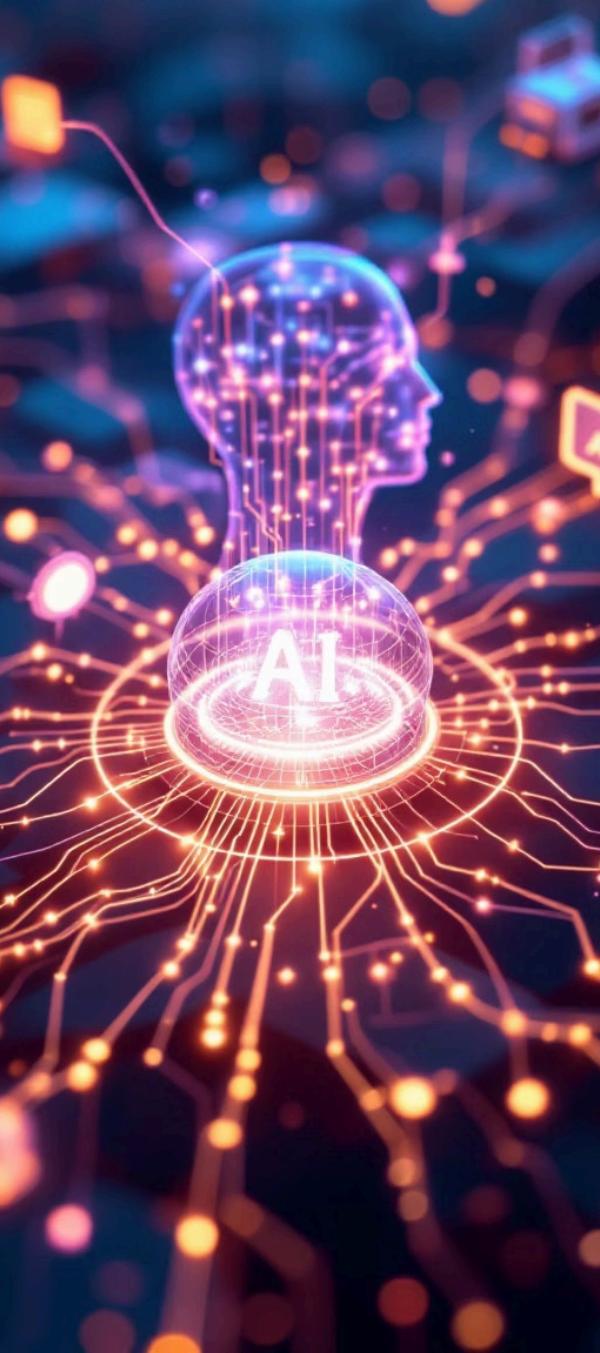
Micro-ejercicio: Agente de Noticias con LlamalIndex



Objetivo: Implementar un agente de IA que pueda buscar noticias en tiempo real y mantener conversaciones contextuales usando la API de NewsAPI.

Tiempo: 15 minutos.

- ⓘ **Pista:** Considera cómo estructurar las herramientas (Tools) para que el agente pueda realizar búsquedas flexibles por temas o fuentes o lee el archivo `news_api.py`.



Agentes con LlamalIndex



Los agentes son sistemas que utilizan modelos de lenguaje para decidir qué acciones tomar para resolver tareas complejas.

Características clave

- Pueden **seleccionar herramientas** apropiadas según el contexto
- Siguen un **proceso iterativo** de razonamiento
- Son capaces de **descomponer problemas complejos** en tareas más sencillas
- Pueden **combinar información** de múltiples fuentes

En LlamalIndex existen principalmente dos tipos de agentes: `ReActAgent` y `OpenAIAGent`, con diferentes estrategias de razonamiento.

Componentes de un Agente en LlamalIndex

1

Modelo de Lenguaje (LLM)

El cerebro del agente que toma decisiones y genera respuestas. Puede ser OpenAI, Anthropic Claude, Llama2, etc.

2

Herramientas (Tools)

Funciones que el agente puede invocar para obtener información o realizar acciones. Ejemplos: FunctionTool, QueryEngineTool.

3

Memoria/Estado

Almacena el historial de interacciones y decisiones tomadas para mantener el contexto de la conversación.

4

Estrategia de Razonamiento

Define cómo el agente aborda los problemas. ReAct (Reasoning + Acting) es la estrategia más común.

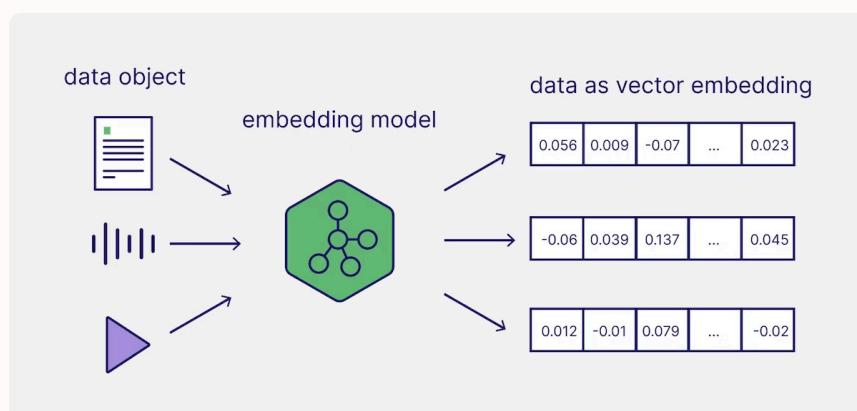
Bases de Datos Vectoriales

Una base de datos vectorial es un tipo especializado de base de datos diseñada para almacenar, gestionar y buscar vectores de alta dimensión, que son representaciones numéricas de datos (texto, imágenes, audio, etc.) en un espacio multidimensional. Son esenciales para aplicaciones de IA que requieren comprensión semántica y búsqueda de similitud.

El Problema: Búsqueda en Grandes Volúmenes de Datos

En la era de la información, el volumen de datos no estructurados (documentos, artículos, grabaciones) es inmenso. La búsqueda tradicional por palabras clave a menudo falla en capturar el significado o el contexto real de una consulta.

Buscar manualmente o con métodos de coincidencia exacta en miles o millones de documentos es ineficiente y no inteligente. Necesitamos una forma de encontrar información relevante incluso si las palabras exactas no coinciden, basándonos en el "significado" o la "intención".



La Solución: Embeddings y Búsqueda Semántica

Aquí es donde entran los **embeddings**. Los modelos de lenguaje transforman el texto (o cualquier tipo de dato) en vectores numéricos, donde la proximidad entre vectores indica similitud semántica. Las bases de datos vectoriales están optimizadas para almacenar estos embeddings y realizar búsquedas de "vecinos más cercanos" de forma rápida y eficiente.

Esto permite:

- Buscar por significado, no solo por palabras clave.
- Encontrar contenido relacionado, aunque no contenga la frase exacta.
- Potenciar funcionalidades como la recuperación de información para LLMs (RAG).

Ventajas Clave de las Bases de Datos Vectoriales

Velocidad

Realizan búsquedas de similitud en milisegundos, incluso con conjuntos de datos masivos.

Inteligencia Semántica

Comprenden el contexto y el significado, ofreciendo resultados más relevantes y enriquecidos.

Persistencia

Permiten almacenar los embeddings de forma duradera, para su recuperación y uso posterior en diversas aplicaciones.

Similitud del Coseno vs. Distancia Euclíadiana

En el mundo de los embeddings, necesitamos métricas para cuantificar la "similitud" o "proximidad" entre vectores. Aunque ambas son formas de medir la relación entre puntos en un espacio multidimensional, cada una tiene sus fortalezas y usos específicos.

Similitud del Coseno

Mide el **coseno del ángulo** entre dos vectores. Un valor cercano a 1 indica alta similitud, mientras que 0 indica ortogonalidad (no relación) y -1, oposición. Se enfoca en la **dirección** de los vectores, ignorando su magnitud.

Es la métrica preferida para texto, ya que los documentos de longitudes variables (y por tanto, diferentes magnitudes vectoriales) pueden tratar temas idénticos, y la dirección de su vector captura ese significado semántico.

Distancia Euclíadiana

Mide la **distancia en línea recta** entre dos puntos (vectores) en un espacio euclidiano. Es la familiar fórmula de distancia que usamos en geometría. Considera tanto la **dirección como la magnitud** de los vectores.

Puede ser engañosa en espacios de alta dimensión o con datos de texto, ya que un documento más largo podría tener una mayor distancia euclíadiana de uno corto, incluso si son semánticamente similares, simplemente por su magnitud.

La elección de la métrica de similitud es crucial y debe basarse en la naturaleza de los datos y el problema a resolver.

¡Manos a la Obra!

Es momento de aplicar los conceptos que hemos cubierto sobre agentes con Llamaindex.



Abre el siguiente Jupyter Notebook para comenzar con los ejercicios prácticos:

02_agentes_llamaindex.ipynb

¡Prepárate para programar y experimentar!

Micro-ejercicio: Sistemas Multi-agentes de noticias con LlamaIndex



Objetivo: Implementar un sistema multi-agentes híbrido capaz de buscar noticias en tiempo real , lanzar consultas a nuestra base de datos vectorial , consultar el tiempo y calcular índices personalizados.

Tiempo: 15 minutos.

- ⓘ **Pista:** Puedes reutilizar descripciones de agentes y código del micro-ejercicio anterior dónde usamos el módulo news_api.py.



Bloque 2 : Pydantic



Refresco Pydantic (11:40 – 12:00)

- **Objetivo:** comprender cómo usar **Pydantic v2** para controlar y validar salidas de LLM en proyectos con *llama-index*.
- **Contenido:**
 - a. Breve repaso: ¿Qué es *Pydantic*?
 - b. Fundamentos de Pydantic (BaseModel, Field, @model_validator, @field_validator).
 - c. Validación y normalización avanzada. Serialización y conversiones.
 - d. Ejercicio guiado: integración en un generador sintético de datos mediante un LLM.

Parseo de documentos (12:00 – 12:30)

- **Objetivo:** aprender a construir agentes autónomos que combinen varias fuentes y herramientas.
- **Contenido:**
 - a. Concepto de agente y *tool* en *llama-index*.
 - b. Arquitectura de un agente básico.
 - c. Ejemplo: agente que consulta varias APIs y responde de forma razonada.
 - d. Ejercicio guiado: construir un sistema multi-agente que llame a la API creada en el bloque anterior y nuestra propia base de datos vectorial.

Pydantic: Validación y Estructuración de Datos

Pydantic es una poderosa biblioteca de Python que aprovecha los "type hints" para proporcionar validación de datos en tiempo de ejecución, configuración de ajustes y serialización de datos de manera eficiente.

¿Qué es Pydantic?

Es una biblioteca de Python que facilita la declaración de cómo deben ser tus datos, garantizando que los datos recibidos (por ejemplo, de una API o un formulario web) cumplan con la estructura y el tipo esperados. Actúa como un guardián de la calidad de tus datos.

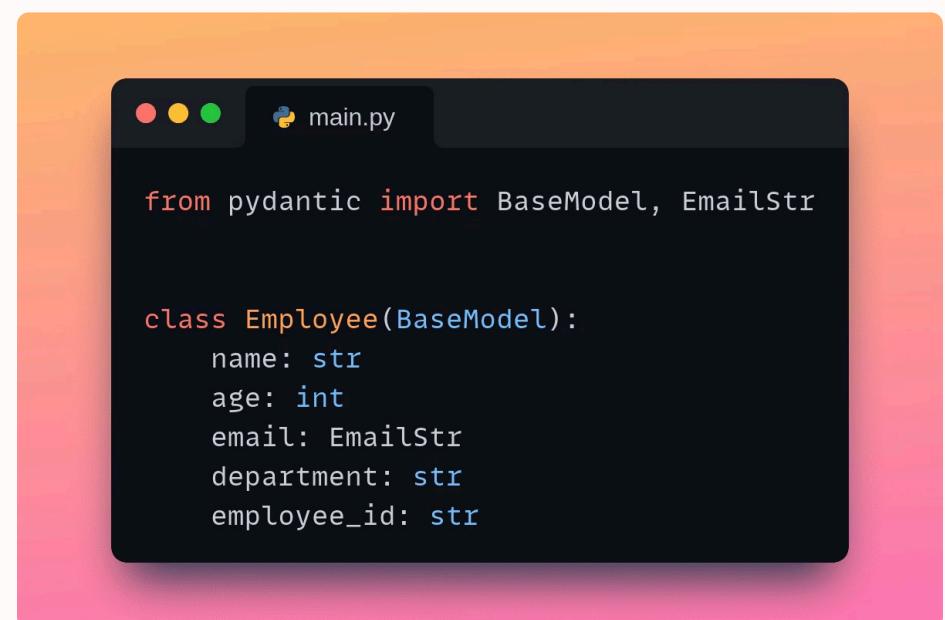
¿Para qué se utiliza?

Pydantic es ideal para validar entradas de APIs, gestionar configuraciones de aplicaciones complejas, serializar y deserializar objetos de datos (por ejemplo, convertir un objeto Python a JSON y viceversa) y crear modelos de datos robustos que se auto-validan.

Ejemplo Claro: Registro de Usuario

Imagina que tienes un formulario de registro donde un usuario debe introducir su nombre, email y edad. Sin Pydantic, tendrías que escribir un montón de código manual para verificar que el nombre es una cadena, el email tiene formato de email y la edad es un número entero positivo.

Con Pydantic, simplemente defines un "modelo" de usuario con esos campos y sus tipos. Cuando recibes los datos del formulario, Pydantic se encarga automáticamente de validar cada campo. Si el email no es válido o la edad no es un número, Pydantic levanta un error claro, simplificando enormemente el manejo de datos y reduciendo errores.



```
from pydantic import BaseModel, EmailStr

class Employee(BaseModel):
    name: str
    age: int
    email: EmailStr
    department: str
    employee_id: str
```

¡Manos a la Obra!

Es momento de aplicar los conceptos que hemos cubierto sobre Pydantic.



Abre el siguiente Jupyter Notebook para comenzar con los ejercicios prácticos:

03_refresco_pydantic.ipynb

¡Prepárate para programar y experimentar!

Micro-ejercicio: Validación Avanzada de Datos Sintéticos con Pydantic



Objetivo: Implementar un sistema avanzado de **validación y generación de datos sintéticos** utilizando **Pydantic** con validaciones personalizadas, constraints avanzados y control estricto de salidas de LLMs para demostrar la importancia del tipado en sistemas de IA.

Tiempo: 15 minutos.

- ⓘ **Pista:** Al generar un dataset sintético, conviene aumentar la temperatura de los LLMs para lograr mayor diversidad en los datos.

Extracción Estructurada: LlamalIndex y Pydantic en Acción



→ El Desafío: Texto Libre de LLMs

Los Modelos de Lenguaje Grandes (LLMs) son herramientas potentes, pero su salida natural es texto no estructurado. Esto dificulta su integración directa en aplicaciones empresariales que requieren datos precisos y predecibles (por ejemplo, para bases de datos, APIs o paneles de control).

→ La Solución: Pydantic para Estructuración

Aquí es donde Pydantic brilla. Al definir modelos de validación claros, podemos transformar el texto libre de los LLMs en formatos estructurados como JSON. Esto asegura que los datos sean fiables, consistentes y fácilmente consumibles por otros sistemas.

LlamalIndex: El Motor de Extracción Documental

Antes de estructurar datos, necesitamos acceder a ellos. LlamalIndex actúa como un potente motor para cargar y abstraer diversas fuentes documentales, preparándolas para el procesamiento con LLMs.



Fuentes Diversas

LlamalIndex puede cargar documentos de múltiples formatos, incluyendo PDFs, DOCX, páginas web (HTML), y archivos de texto (.TXT).



Abstracciones Clave: Document & Node

Internamente, LlamalIndex organiza la información utilizando abstracciones como `Document` (un archivo completo) y `Node` (segmentos más pequeños de texto de un documento), facilitando la gestión y recuperación de información.



Lectores Integrados

Ofrece lectores pre-construidos como `SimpleDirectoryReader` para archivos locales o `BeautifulSoupWebReader` para contenido web, simplificando la ingesta de datos.

El Poder de LlamaIndex: Transformando Datos Desestructurados



El Problema: Datos Dispersos y Desestructurados

En el mundo real, la información vital se encuentra fragmentada en diversos formatos: PDFs, HTML, emails o informes financieros. Su naturaleza desestructurada dificulta su uso directo en modelos de lenguaje.



La Solución: LlamaIndex Unifica y Estructura

LlamaIndex convierte eficientemente cualquier documento desestructurado en "nodos" que los LLMs pueden consumir. Proporciona un pipeline consistente, simplificando el procesamiento de información, sin importar la fuente original.

Conceptos Fundamentales y Flujo de Procesamiento

01

Document

Representa un documento completo (PDF, HTML, texto, etc.) como unidad de entrada.

02

Splitter

Divide el documento en fragmentos más pequeños y manejables, optimizados para los LLMs.

03

Nodes

Son los fragmentos de información (texto) junto con sus metadatos asociados, listos para ser indexados.

04

Vector Store

Almacena los nodos vectorizados, permitiendo búsquedas semánticas eficientes.

05

Query Engine

Utiliza los nodos almacenados para generar respuestas estructuradas a partir de consultas de lenguaje natural.

Conectores de Datos Integrados

LlamaIndex ofrece una amplia gama de conectores para facilitar la ingesta de datos desde diversas fuentes.

Conejor	Uso Típico	Ejemplo
SimpleDirectoryReader	Archivos locales	PDFs, TXT, DOCX
PDFReader	Documentos PDF específicos	Reportes financieros
WebPageReader	Contenido de páginas web	Artículos, blogs
WikipediaReader	Datos enciclopédicos	Información general
NotionPageReader	Documentación en Notion	Notas de proyectos

Metadatos: La Clave para un Contexto Enriquecido

Cada Node en LlamaIndex no solo contiene texto, sino también metadatos cruciales que enriquecen su valor y permiten a los modelos trabajar con información más precisa y relevante.



Contextualización

Ayudan al modelo a comprender el origen de la información y bajo qué condiciones fue generada, por ejemplo, la fecha de publicación de un informe financiero.



Filtrado Inteligente

Permiten restringir las consultas a criterios específicos, como buscar solo artículos posteriores a una fecha determinada o de una fuente particular.



Priorización y Relevancia

Facilitan la asignación de importancia a los nodos. Un nodo de un experto tendrá más peso que uno de un blog anónimo, aunque contengan texto similar.



Trazabilidad

Cada respuesta generada puede rastrearse hasta su nodo original y sus metadatos, esencial en entornos corporativos para validar el origen de la información.

¡Manos a la Obra!

Es momento de aplicar los conceptos que hemos cubierto sobre el parseo de documentos con `llamaindex` + `pydantic`.



Abre el siguiente Jupyter Notebook para comenzar con los ejercicios prácticos:

04_parseo_documentos.ipynb

¡Prepárate para programar y experimentar!



Micro-ejercicio: Parseo de lyrics de canciones de una web con Pydantic

Objetivo: Implementar un sistema de **extracción estructurada de letras de canciones (lyrics)** desde páginas web utilizando **Llamaindex** y **Pydantic**. Deberás tipar tanto la letra de la canción como las posibles visualizaciones o características extraíbles de las mismas (por ejemplo, recuento de palabras, análisis de sentimientos, etc.). Además, se deberá utilizar LLMs para **calificar y clasificar las canciones por estado de ánimo y género**.

Tiempo: 15 minutos.

- ⓘ **Pista:** Utiliza el JSON de ejemplo que se encuentra en la carpeta `/data` para guiar la estructuración de tus modelos Pydantic y el tipado de las salidas de los LLMs.

Recapitulando Nuestro Viaje: LlamalIndex y Pydantic

Hemos explorado el poder transformador de LlamalIndex y Pydantic, dos herramientas clave para construir aplicaciones robustas con Large Language Models.



LlamalIndex: El Orquestador de Datos

Aprendimos cómo LlamalIndex carga, indexa y organiza datos de diversas fuentes, transformándolos en "nodos" accesibles para los LLMs.

- Manejo de documentos, nodos y conectores de datos.
- Importancia de los metadatos para la contextualización.

Agentes y APIs con LlamalIndex

Vimos cómo construir agentes inteligentes capaces de interactuar con el mundo exterior a través de APIs y herramientas, ampliando las capacidades de los LLMs.

- Integración con servicios externos.
- Construcción de lógica de decisión y ejecución.



Pydantic: La Estructura es Clave

Comprendimos la importancia de Pydantic para validar, tipar y estructurar la salida de los LLMs, asegurando la fiabilidad y la consistencia de los datos.

- Modelos de datos para una salida predecible.
- Validación de datos para evitar errores.

Extracción Estructurada: Sinergia Perfecta

Finalmente, unimos LlamalIndex y Pydantic para realizar una extracción de información estructurada potente y fiable de datos desestructurados.

- Transformación de texto libre a formatos JSON.
- Casos de uso para análisis de sentimientos y clasificación.