Retrieval-Augmented Generation (RAG)

El motor detrás de la nueva generación de chatbots

PhD. Oscar Alberto Rodriguez Melendez

Universidad de la Sabana

2025-II

Contenido

Requerimientos del Entorno Productivo

Para generar los códigos que trabajaremos en el curso, es necesario crear un entorno que nos permita desarrollar en **Python**, mantener las mejores prácticas de la industria y controlar versiones.

Herramientas principales

- **Conda** o **Poetry**: gestión de entornos virtuales y dependencias.
- **Python**: lenguaje de programación base.
- **GitHub**: control de versiones y trabajo colaborativo.
- **PyCharm** o **VS Code**: entornos de desarrollo integrados (IDE).

Instalación de Miniconda

Miniconda nos permite crear entornos virtuales ligeros para ejecutar y aislar nuestros proyectos.

1. Ingresa a: `https://www.anaconda.com/docs/getting-started/miniconda/install`
2. Descarga la versión correspondiente a tu sistema operativo (Windows, macOS o Linux).
3. Instala siguiendo las instrucciones por defecto.
4. Verifica la instalación con: **conda –version**

Nota para Windows

Ejecuta los comandos de instalación desde la consola **Anaconda Prompt**, no desde CMD o PowerShell.

Creación de cuenta en GitHub

1. Ingresa a: `https://github.com`
2. Haz clic en **Sign up** y crea una cuenta gratuita.
3. Verifica tu correo electrónico.
4. Personaliza tu perfil con nombre, foto y descripción opcional.

Propósito

GitHub será nuestro repositorio remoto para almacenar y versionar el código del curso.

Creación del entorno con Conda

Una vez instalado Miniconda y creada la cuenta de GitHub:

1. Descarga el archivo `Chatbots.yml` desde:
   `https://github.com/orodriguezm1/Chatbots/tree/dev`
2. Guarda el archivo en tu carpeta de trabajo.
3. Abre una terminal (o Anaconda Prompt en Windows).
4. Ejecuta el comando: **conda env create -f Chatbots.yml**
5. Activa el entorno: **conda activate Chatbots**

Instalación de PyCharm

1. Ingresa a: `https://www.jetbrains.com/pycharm/download/`
2. Descarga la versión disponible.
3. Instala el IDE con las opciones por defecto.
4. Al abrir PyCharm por primera vez:
   - Selecciona el entorno de Conda (`Chatbots`).
   - Configura el tema visual (claro u oscuro).

Ventaja

PyCharm permite ejecutar notebooks, scripts y controlar GitHub desde la misma interfaz.

Conexión de PyCharm con GitHub

Objetivo: conectar el entorno local con tu cuenta de GitHub mediante un token seguro.

1. **Generar token en GitHub:**
   - Entra a tu cuenta y ve a **Settings → Developer settings → Personal access tokens**.
   - Selecciona **Tokens → Generate new token**.
   - Copia el token generado.
2. **Conectar en PyCharm:**
   - Ve a **File → Settings → Version Control → GitHub**.
   - Haz clic en "+" → **Log in via Token**.
   - Pega el token y presiona **OK**.

Clonar el repositorio desde tu cuenta (Fork)

1. Ingresa al repositorio original del curso:
   `https://github.com/orodriguezm1/Chatbots/tree/dev`
2. Haz clic en el botón **Fork** (parte superior derecha). Esto creará una copia del repositorio en tu cuenta personal de GitHub.
3. Abre tu cuenta y verifica que el nuevo repositorio aparezca como:
   `https://github.com/tu_usuario/Chatbots`
4. En PyCharm, ve al menú: **Git** → **Clone Repository**
5. Pega la URL de tu fork (tu copia personal), no la del repositorio original.
6. Haz clic en **Clone** y PyCharm descargará el proyecto localmente.

Importante

Trabaja siempre sobre tu fork personal. Los cambios que realices (commits, push, ramas) afectarán solo tu repositorio.

Actualizar el Fork desde GitHub

Puedes mantener tu fork sincronizado con el repositorio original directamente desde la página de GitHub:

1. Ingresa a tu cuenta de GitHub y abre tu repositorio personal `Chatbots`.
2. Si el repositorio original tiene cambios, verás un mensaje en la parte superior: **"This branch is behind orodriguezm1:dev by X commits"**.
3. Haz clic en el botón **"Sync fork"** o **"Fetch upstream"**.
4. Luego selecciona la opción **"Update branch"**.
5. GitHub traerá los nuevos commits del repositorio original a tu fork automáticamente.

Resultado

Tu fork se actualiza con el repositorio original sin escribir comandos. Al abrir PyCharm y ejecutar un **Git Pull**, los cambios aparecerán también en tu entorno local.

Consejo: Usa este método si solo necesitas sincronizar tu código. Si estás trabajando con ramas personalizadas o colaborando en grupo, es preferible hacerlo con los comandos de Git o desde PyCharm (o VS).

Crear cuenta en OpenRouter (openrouter.ai)

¿Qué es OpenRouter?

Un *gateway* que unifica acceso a múltiples modelos (Mistral, Llama, etc.) usando una API compatible con OpenAI.

1. Entra a `https://openrouter.ai` y pulsa **Sign in**.
2. Inicia sesión con **Google** o **GitHub**, etc.
3. Verifica tu correo si te lo solicita.

Generar tu Token de API

1. En **openrouter.ai** ve a **Dashboard** → **Keys** o **Settings** → **API Keys**.
2. Pulsa **Create new key**.
3. Asigna un nombre (p.ej., `Chatbots`) y crea el token.
4. Copia el token (formato típico `sk-or-v1-...`). **Guárdalo en lugar seguro**.

Importante

No compartas tu token en repositorios públicos. Trátalo como una contraseña.

Guardar el Token en `.env`

1. En la carpeta raíz de tu proyecto (donde está tu `.py`), existe un archivo `.env` con:

Contenido de `.env`

```
OPENAI_API_KEY = sk-or-v1-tu_token_de_openrouter
```

Reemplaza el valor que se encuentra en el archivo por el key que generaste.

Notas

- Asegúrate de que el token empiece por `sk-or-v1-`.
- Si usas **VS Code** o **PyCharm**, ejecuta desde la carpeta que contiene el `.env`.

Errores comunes (y cómo resolverlos)

- **Invalid authentication**: revisa que el token sea el de OpenRouter (`sk-or-v1-...`) y esté en `OPENAI_API_KEY`.
- **No se encuentra .env**: ejecuta el script desde la carpeta del proyecto o pasa ruta a `load_dotenv(ruta/.env")`.
- **Modelo no disponible**: verifica el `model_name` en **openrouter.ai/models**. Ej.: `mistralai/mistral-7b-instruct`.
- **403/429 (cuotas)**: revisa **Billing** en OpenRouter o baja `temperature` y frecuencia de llamadas.

¿Qué es RAG?

RAG

La **información aumentada por recuperación** o RAG es el proceso mediante el cual la salida de un modelo de lenguaje de gran tamaño (LLM) es influenciada para hacer referencia a una base de conocimientos autorizada externa a los orígenes de entrenamiento del modelo previo a la generación de la respuesta.

Esquema ilustrativo

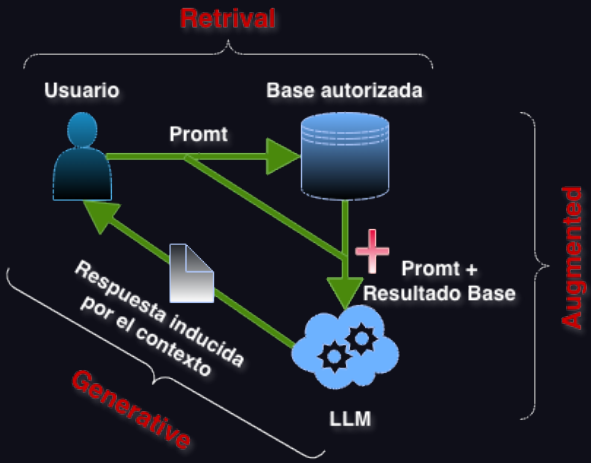


Figura: Flujo general de un sistema RAG: recuperación, aumento y generación.

¿Porqué es importante?

El enfoque RAG (Retrieval-Augmented Generation) representa una evolución clave en el uso de modelos de lenguaje grandes (LLMs), porque permite actualizar su conocimiento y adaptarlos a dominios específicos sin necesidad de reentrenamiento.
En un modelo tradicional, la única forma de incorporar nueva información o ajustar el comportamiento del modelo es mediante fine-tuning: volver a entrenar el modelo con datos adicionales.

¿Porqué es importante?

Tres limitaciones del fine-tuning

- **Costo y tiempo**: El reentrenamiento de un LLM requiere enormes recursos computacionales y tiempos prolongados.
- **Riesgo de degradar el modelo**: Cada ajuste puede alterar el conocimiento previo y generar errores o sesgos no deseados.
- **Desactualización rápida**: El modelo vuelve a quedar "congelado" en el conocimiento usado en el último fine-tuning.

Ventajas del enfoque RAG

Beneficios frente al fine-tuning

- **Actualización inmediata**: Basta con añadir nuevos documentos a la base externa.
- **Menor costo**: Evita el proceso de reentrenamiento completo.
- **Adaptabilidad**: Permite incorporar información específica sin modificar el modelo.
- **Trazabilidad**: Las fuentes utilizadas son identificables y auditables.

¿Qué son los Embeddings?

Idea general

Un **embedding** es una representación vectorial de una palabra, frase o entidad en un espacio continuo $\mathbb{R}^n$.

- El objetivo es que palabras que aparecen en contextos similares tengan vectores similares.
- Cada palabra se representa como un vector numérico que captura su significado semántico.

Palabra	**Embedding (ejemplo)**
perro	[0.45, -0.31, 0.22, ...]
gato	[0.44, -0.28, 0.20, ...]
coche	[-0.12, 0.35, -0.10, ...]

"perro" y "gato" están cerca en el espacio vectorial.

Inicialización de los vectores

Representación inicial

Antes de entrenar, a cada palabra del vocabulario se le asigna un vector aleatorio.

$$E \in \mathbb{R}^{V \times n}$$

donde:

- V: tamaño del vocabulario (por ejemplo, 10 000).
- n: dimensión del embedding (por ejemplo, 300).
- Cada fila E_i es el vector de la palabra w_i.

Palabra	Vector inicial (aleatorio)
gato	[0.01, -0.02, 0.03]
perro	[-0.04, 0.01, 0.02]
casa	[0.02, 0.01, -0.01]

Entrenamiento de Embeddings

Idea general

Durante el entrenamiento, el modelo ajusta los vectores para que palabras que aparecen juntas tengan embeddings similares.

Ejemplo: "El gato duerme en la cama."

pares de entrenamiento: $(gato, duerme), (duerme, gato), (gato, cama), (cama, duerme)$

- El modelo aprende relaciones de contexto.
- Los vectores se actualizan iterativamente para reducir la pérdida.

Modelos clásicos: Word2Vec

Dos variantes principales

1. **Skip-gram:** predice palabras de contexto dado el centro.

$$P(w_o|w_c) = \frac{\exp(\mathbf{v}'_{w_o}{}^\top \mathbf{v}_{w_c})}{\sum_{j=1}^{V} \exp(\mathbf{v}'_j{}^\top \mathbf{v}_{w_c})}$$

2. **CBOW (Continuous Bag of Words):** predice la palabra central dado el contexto.

$$P(w_c|\text{contexto}) = \frac{\exp(\mathbf{v}'_{w_c}{}^\top \sum_{w \in contexto} \mathbf{v}_w)}{\sum_{j=1}^{V} \exp(\mathbf{v}'_j{}^\top \sum_{w \in contexto} \mathbf{v}_w)}$$

Función de pérdida (Skip-gram con Negative Sampling)

Objetivo del entrenamiento

Maximizar la similitud entre pares reales y minimizarla para pares falsos.

$$L = -\log \sigma(\mathbf{v'}_{w_o}{}^{\top} \mathbf{v}_{w_c}) - \sum_{i=1}^{k} \log \sigma(-\mathbf{v'}_{w_i^-}{}^{\top} \mathbf{v}_{w_c})$$

donde:

- $\sigma(x) = \frac{1}{1+e^{-x}}$
- w_i^-: palabras negativas (aleatorias)
- k: número de muestras negativas

Actualización de los vectores

Gradiente y descenso

Los vectores se actualizan con descenso de gradiente:

$$\mathbf{v}_{new} = \mathbf{v}_{old} - \eta \frac{\partial L}{\partial \mathbf{v}}$$

donde η es la tasa de aprendizaje.

Actualización de los vectores

Ejemplo práctico:

$$\mathbf{v}_{gato} = [0{,}2, 0{,}4]$$
$$\mathbf{v}'_{animal} = [0{,}3, 0{,}1]$$
$$\mathbf{v}'_{mesa} = [-0{,}2, 0{,}5]$$

$$\sigma({\mathbf{v}'_{animal}}^{\top} \mathbf{v}_{gato}) = \sigma(0{,}1) = 0{,}525$$
$$\sigma(-{\mathbf{v}'_{mesa}}^{\top} \mathbf{v}_{gato}) = \sigma(-0{,}16) = 0{,}46$$

Tras la actualización, $\mathbf{v}_{gato}$ se acerca a "animal" y se aleja de "mesa".

Resultado final del entrenamiento

Embeddings semánticos

Después del entrenamiento, los embeddings capturan relaciones semánticas:

$$\mathbf{v}_{rey} - \mathbf{v}_{hombre} + \mathbf{v}_{mujer} \approx \mathbf{v}_{reina}$$

- Las distancias reflejan similitudes semánticas.
- cos(“perro”, “gato”) $\approx$ alto.
- cos(“perro”, “mesa”) $\approx$ bajo.

¿Cómo comparamos la similitud entre palabras?

Objetivo

Una vez que cada palabra, frase o documento se representa como un vector, necesitamos una forma de medir qué tan **parecidos** son entre sí. Para esto, usamos métricas o medidas de similitud.

- En RAG, las medidas de similitud se usan para encontrar los fragmentos más relevantes respecto a una consulta.
- Permiten identificar qué embeddings son más cercanos al embedding del **prompt**.
- Cuanto más alta sea la similitud, más relevante se considera el texto.

Tipos comunes de medidas de similitud

Principales medidas

- **Similitud del coseno**: compara el ángulo entre los vectores.
- **Distancia euclidiana**: mide la distancia "en línea recta".
- **Distancia de Manhattan**: suma las diferencias absolutas entre coordenadas.

Medida	**Fórmula**
Coseno	$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$
Euclidiana	$d(A,B) = \sqrt{\sum_i (A_i - B_i)^2}$
Manhattan	$d(A,B) = \sum_i \|A_i - B_i\|$

Similitud del Coseno

Definición

Mide el **ángulo** entre dos vectores en el espacio. Si el ángulo es pequeño, los vectores apuntan en direcciones similares, lo que significa que los textos son semánticamente parecidos.

$$\text{sim}(A, B) = \frac{A \cdot B}{\|A\|\|B\|}$$

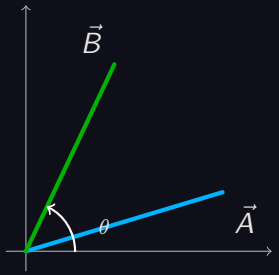
Interpretación:

- 1 $\rightarrow$ vectores idénticos.
- 0 $\rightarrow$ sin relación (perpendiculares).
- -1 $\rightarrow$ opuestos.

Visualización geométrica de la similitud

Ejemplo en 2D

Los embeddings se comportan como vectores en un espacio geométrico.



La similitud del coseno mide qué tan alineados están los vectores. Cuanto menor sea θ, mayor es la similitud.

Ejemplo numérico

Ejemplo

Sean:

$$A = [1, 2, 3], \quad B = [2, 3, 4]$$

$$\text{sim}(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{1 * 2 + 2 * 3 + 3 * 4}{\sqrt{1^2 + 2^2 + 3^2}\sqrt{2^2 + 3^2 + 4^2}} = \frac{20}{\sqrt{14}\sqrt{29}} = 0{,}97$$

Interpretación: Los vectores A y B son muy similares porque el ángulo entre ellos es pequeño.

Comparación entre medidas

Ejemplo conceptual

Consideremos tres palabras con sus embeddings:

Palabra	Embedding (simplificado)
perro	[0.45, -0.31, 0.22]
gato	[0.44, -0.28, 0.20]
mesa	[-0.10, 0.33, -0.12]

- $\cos(\text{perro}, \text{gato}) \approx 0{,}99 \rightarrow$ alta similitud.
- $\cos(\text{perro}, \text{mesa}) \approx 0{,}05 \rightarrow$ casi nula.
- La distancia euclidiana entre "perro" y "gato" también es menor que con "mesa".

Uso de las medidas en RAG

Cómo se usan

En un sistema **RAG**, la similitud del coseno se emplea para buscar los fragmentos de texto más cercanos al embedding del **prompt**.

¡Gracias!