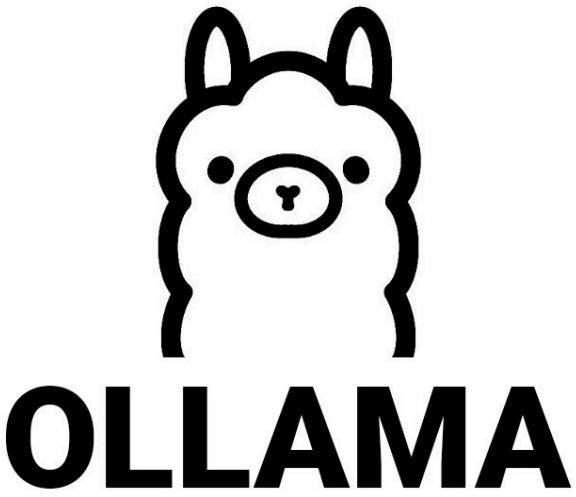


Ollama es una herramienta de código abierto, escrita en GO, que nos permite ejecutar **Grandes Modelos de Lenguajes (LLMs)** de forma muy sencilla y **local**.
Al ejecutar los modelos localmente, mantenemos la propiedad plena de los datos y evitamos posibles riesgos de filtraciones de información sensible. Las herramientas de IA offline como Ollama, también ayudan a reducir la **latencia y la dependencia de servidores externos**, haciéndolos más rápidos y fiables.

¿Cómo funciona?

Ollama crea un **entorno aislado** para ejecutar los LLMs localmente en nuestros sistemas, lo que evita conflictos con otro software instalado.
Este entorno, ya incluye todos los componentes necesarios para desplegar modelos de IA, tales como:

- **Pesos del modelo:** Los datos pre entrenados que utiliza el modelo para funcionar.

- **Archivos de configuración:** ajustes que definen como se comporta el modelo.
- **Dependencias necesarias:** bibliotecas y herramientas que apoyan la ejecución del modelo. S

En pocas palabras: primero, extrae los modelos de la biblioteca Ollama. Luego, ejecutamos estos modelos tal cual o ajustamos los parámetros para personalizarlos para tareas específicas. Tras la configuración, podemos interactuar con los modelos introduciendo preguntas y ellos generaran las respuestas.

Esta avanzada herramientas de IA, funciona mejor en **sistemas de unidades de procesamiento gráfico (GPUs).**
Aunque, es posible ejecutarlas en GPUs integradas en la CPU, o directamente utilizando la CPU. Ollama por defecto, al iniciarse, buscará la opción más viable dependiendo de las especificaciones actuales de tu equipo.

Características principales

Ollama ofrece varias funciones clave que facilitan la gestión de modelos fuera de línea y mejoran el rendimiento, como por ejemplo:

- **Gestión local del modelo de IA:** Ollama te concede el control total para descargar, actualizar y eliminar modelos fácilmente en nuestros sistemas. Esta función es muy valiosa para devs e investigadores que dan prioridad a la seguridad estricta en los datos. Además de la gestión básica, ollama te permite rastrear y controlar diferentes versiones del modelo. Esto es muy importante en entornos de investigación y producción, donde puede que necesitemos revertir o probar varias versiones del

modelo para ver cual genera los resultados que nosotros esperamos.

- **Opciones de línea de comandos y de la GUI:** Ollama funciona a través de una interfaz de línea de comandos (CLI), que te proporciona un control preciso de los modelos. La CLI, permite comandos rápidos para extraer, ejecutar y gestionar modelos, lo que es ideal para los fanáticos de Linux o a los que les gusta trabajar en terminal, como es en el caso de su profe.
  Ollama tambien, admite herramientas de interfaz gráfica de usuario (GUI) de terceros, como Open WebUI, para quienes prefieren un enfoque mas visual.

- **Soporte Multiplataforma:** Si bien, los modelos funcionan mejor en sistemas tipo UNIX como Linux y MacOs, es posible descargar, instalarlo y usarlo en sistemas windows, con un instalador típico de “siguiente, siguiente, instalar”. La diferencia principal es que, en distribuciones de Linux y MacOS, funciona de forma **nativa,** a la hora de integrarse con **CUDA** o con **METAL**(MAC).

Casos de uso de Ollama

Crear chatbots locales

Con Ollama, los desarrolladores pueden crear chatbots basados en IA con gran capacidad de respuesta que se ejecutan integramente en servidores locales, garantizando que las interacciones con los clientes sigan siendo totalmente privadas.
Ejecutar chatbots localmente permite a las empresas evitar la latencia asociada a soluciones basadas en la nube, mejorando los tiempos de respuesta para los usuarios finales.

Construir aplicaciones de IA centradas en la privacidad

Ollama ofrece una solución ideal para desarrollar aplicaciones de IA centradas en la privacidad, ideales para empresas que manejan información sensible.
Por ejemplo, los abogados podrían contratar un dev que les cree un sistema potenciado por IA, para el análisis de contratos o la investigación jurídica, sin comprometer la información de los clientes. Ejecutar la IA localmente garantiza que todos los cálculos se produzcan dentro de la infraestructura de la empresa, lo que ayuda a las empresas a cumplir los requisitos normativos de protección de datos personales ISO-27001**.**

INSTALACIÓN DE OLLAMA

Sistemas tipo UNIX (DISTRIBUCIONES DE LINUX y MACOS)

En la terminal, ingresamos el siguiente comando:

- curl -fsSL https://ollama.com/install.sh | sh

Una vez finalizado, podemos ingresar el comando

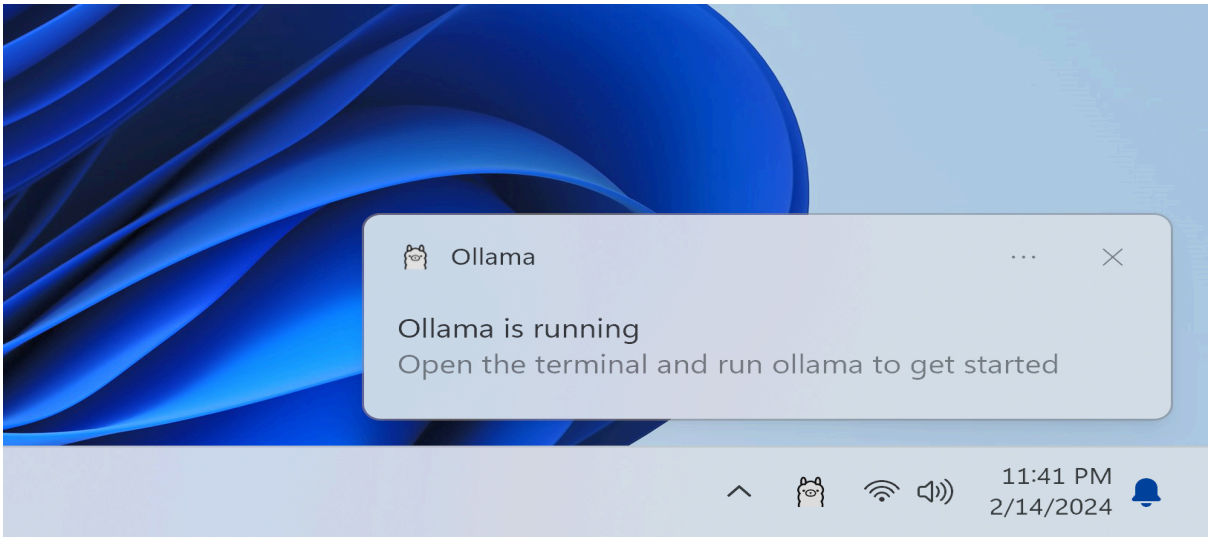
- ollama –version

Obtendremos lo siguiente:

Windows:

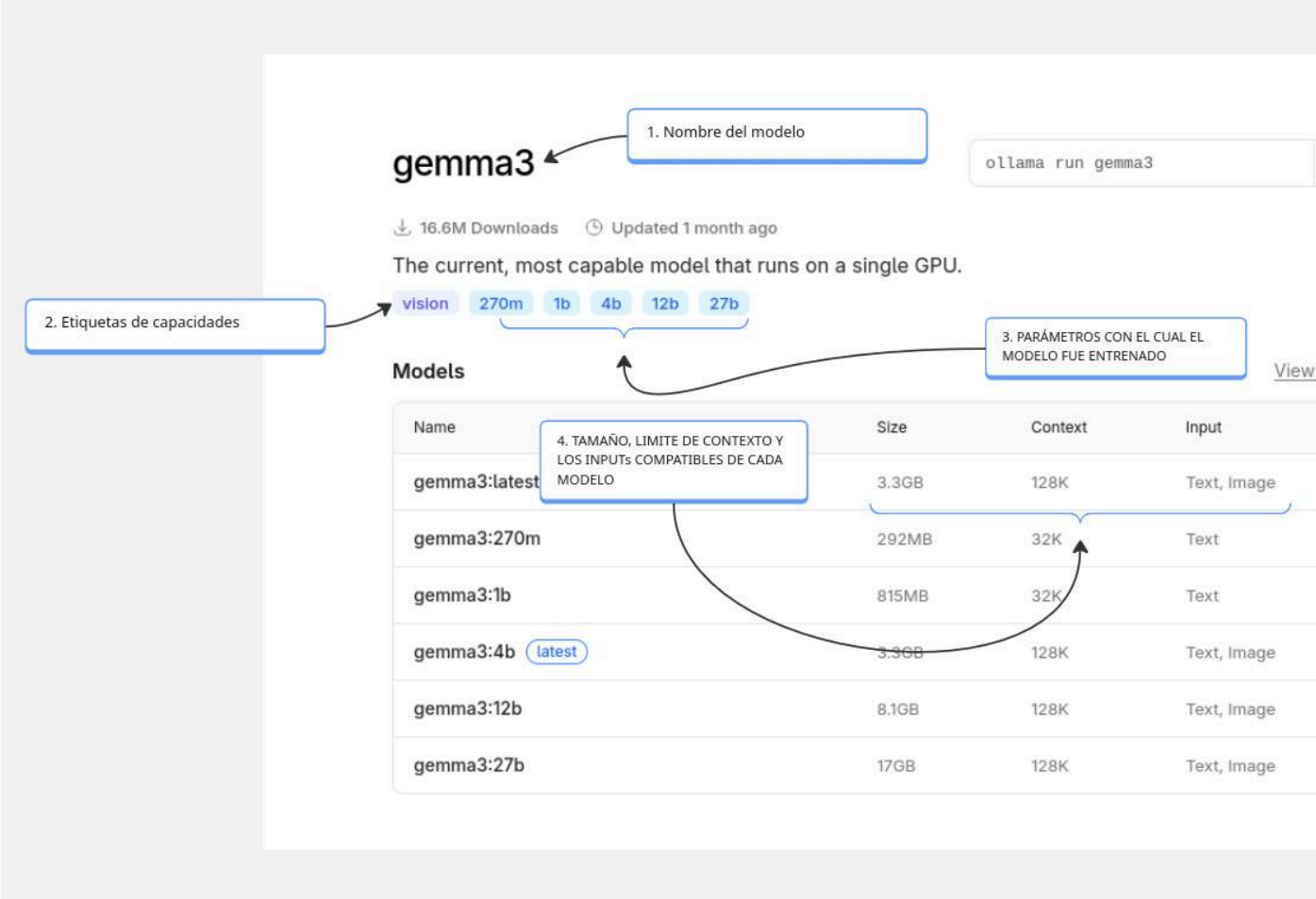
Ingresan a la siguiente URL: https://ollama.com/download/windows

Damos click al botón “Download for Windows” y damos click al archivo descargado.
Una vez descargado e instalado, nos aparecerá lo siguiente:



Ya podemos hacer uso de Ollama!

Elementos a tener en cuenta cuando queremos descargar un modelo de forma local.



1. Nombre: definido del modelo (el que usaremos para descargarlo).

2. Etiquetas de capacidades: En estas etiquetas veremos por ejemplo:

Tools: La etiqueta **tools** en los modelos de Ollama significa que el modelo es capaz de utilizar herramientas externas. Esto implica que, además de generar texto, el modelo puede interactuar con otras funciones o APIs para realizar tareas específicas.

Por ejemplo, un modelo con la etiqueta tools podría ser capaz de:

- **Buscar información en la web:** Si el modelo necesita datos actualizados o específicos que no están en su entrenamiento, podría usar una herramienta de búsqueda para encontrarlos.
- **Realizar cálculos:** Podría invocar una herramienta de calculadora para resolver problemas matemáticos complejos.
- **Acceder a bases de datos:** Podría conectarse a una base de datos para recuperar o almacenar información.
- **Controlar otros sistemas:** En escenarios más avanzados, podría interactuar con software o hardware externos para automatizar acciones.

Vision: significa que el modelo es capaz de procesar y comprender información visual, además de texto. Esto implica que el modelo puede:

- **Analizar imágenes:** Identificar objetos, escenas, colores y otros elementos visuales dentro de una imagen.
- **Responder preguntas sobre imágenes:** Por ejemplo, si se le proporciona una imagen de un perro, podría responder preguntas como "¿Qué animal hay en la imagen?" o "¿De qué color es el perro?".
- **Generar descripciones de imágenes:** Crear texto que describa el contenido de una imagen.
- **Realizar tareas de reconocimiento óptico de caracteres (OCR):** Extraer texto de imágenes.

Thinking: implica que el modelo tiene la capacidad de realizar un razonamiento interno o una planificación antes de generar una respuesta. Esto no significa que el modelo "piensa" como un humano, sino que está diseñado para simular un proceso de pensamiento más complejo al abordar una consulta.

Un modelo con la etiqueta **thinking** podría:

- **Descomponer problemas complejos:** Dividir una pregunta grande en subpreguntas más pequeñas para abordarlas de forma secuencial.
- **Evaluar múltiples enfoques:** Considerar varias estrategias o caminos para llegar a una respuesta y seleccionar el más adecuado.

- **Generar planes de acción:** Crear una secuencia lógica de pasos para resolver una tarea o responder una pregunta.
- **Refinar su propia respuesta:** Analizar su salida preliminar y hacer ajustes para mejorar la calidad, coherencia o precisión.

Cloud: implica que podemos utilizar el modelo, sin la necesidad de descargarlo en nuestras computadoras, directamente de los servidores de meta. (Requiere registro previo).

3. Cantidad de Parámetros: cuando se especifica que un modelo es de por ejemplo: 3B, esto quiere decir 3 mil millones de parámetros.

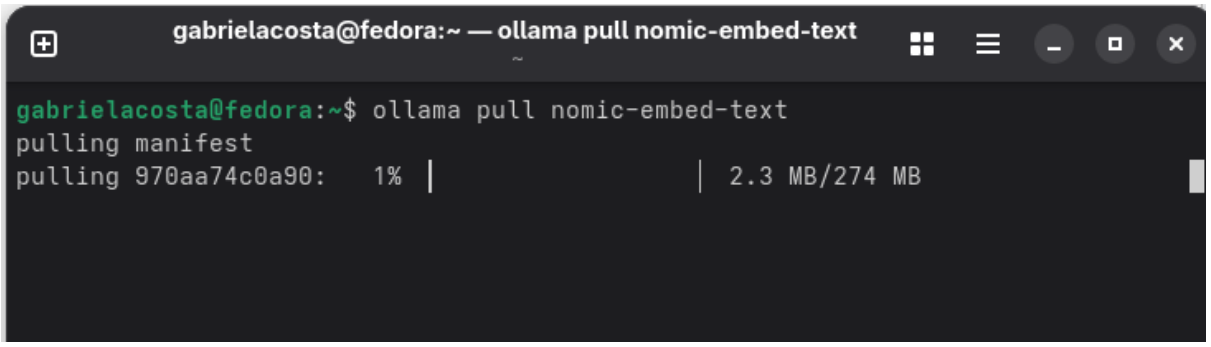
Para tener en cuenta, la cantidad de parámetros de un modelo es:

Cantidad de parámetros = pesos + sesgos de cada capa de la red neuronal, incluyendo los de las capas de atención.

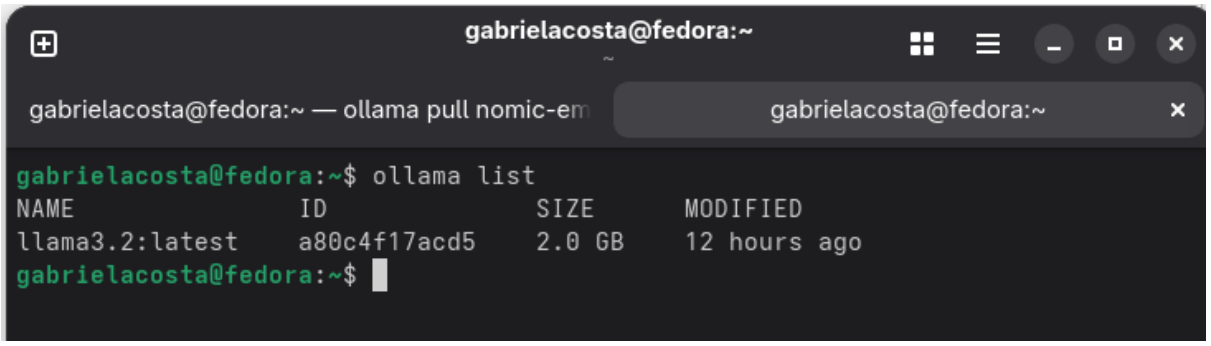
4. **Size**: TAMAÑO del modelo una vez descargado y alojado en el disco.
Contex: Max de token soportado por el modelo para contexto. Una vez superado el límite, el modelo empieza a ignorar los token más alejados.
Input: Entrada que el modelo soporta, por lo general es **texto** y con la nueva actualización de ollama, **imágenes**.

Comandos CLI

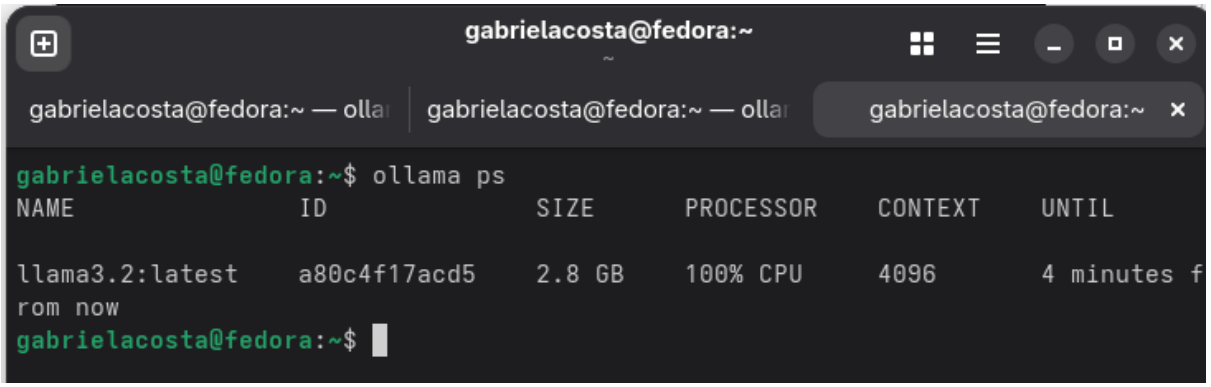
Descargar un modelo localmente: **ollama pull <nombre del modelo>**



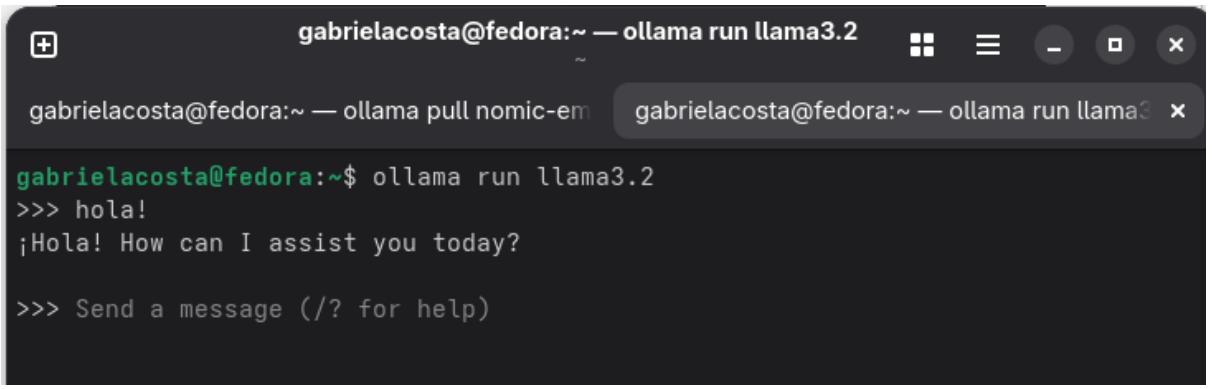
Listar modelos instalados: **ollama list**



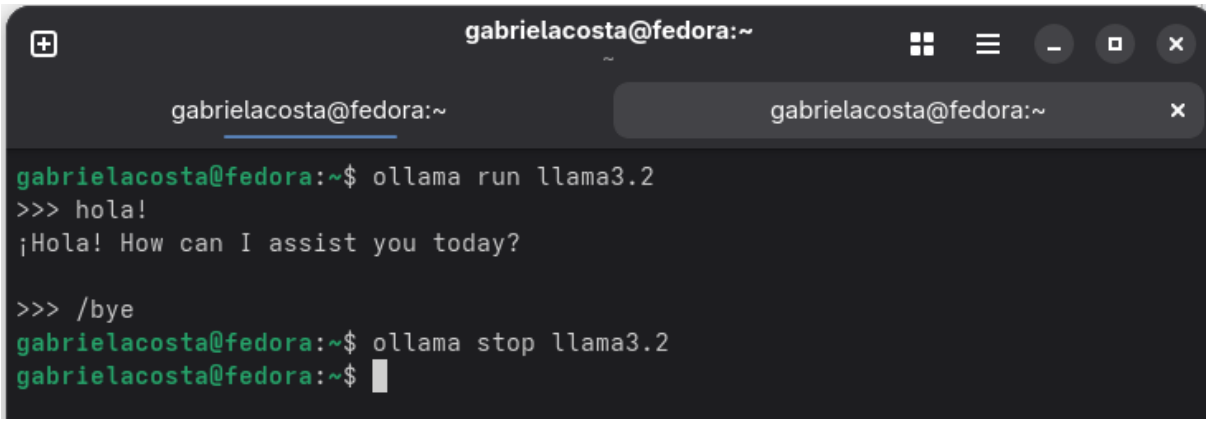
Listar modelos actualmente cargados en memoria: **ollama ps**



Ejecutar un modelo para interactuar en el terminal: **ollama run <nombre del modelo>**



Parar un modelo activo: **ollama stop <nombre del modelo>**



Obtener información de un modelo: **ollama show <nombre del modelo>**

Crear un modelo Personalizado/Ajustado

Ollama nos permite crear modelos personalizados o ajustados, utilizando la función **create** pasando como parámetro el nombre del modelo a asignar y un archivo txt con las configuraciones a aplicar.

Creamos el archivo de texto siguiendo la siguiente sintaxis:

FROM <nombre del modelo >

PARAMETER temperature 0.4
PARAMETER num_ctx 128000 (ventana de contexto // no debe superar a la del modelo).

SYSTEM “””
Lo que queremos ajustar del modelo, por ejemplo:
Eres un asistente encargado de responder preguntas sobre una materia llamada “Seminario de Act. Complementaria - Modelos y Aplicaciones de la IA”

Reglas:

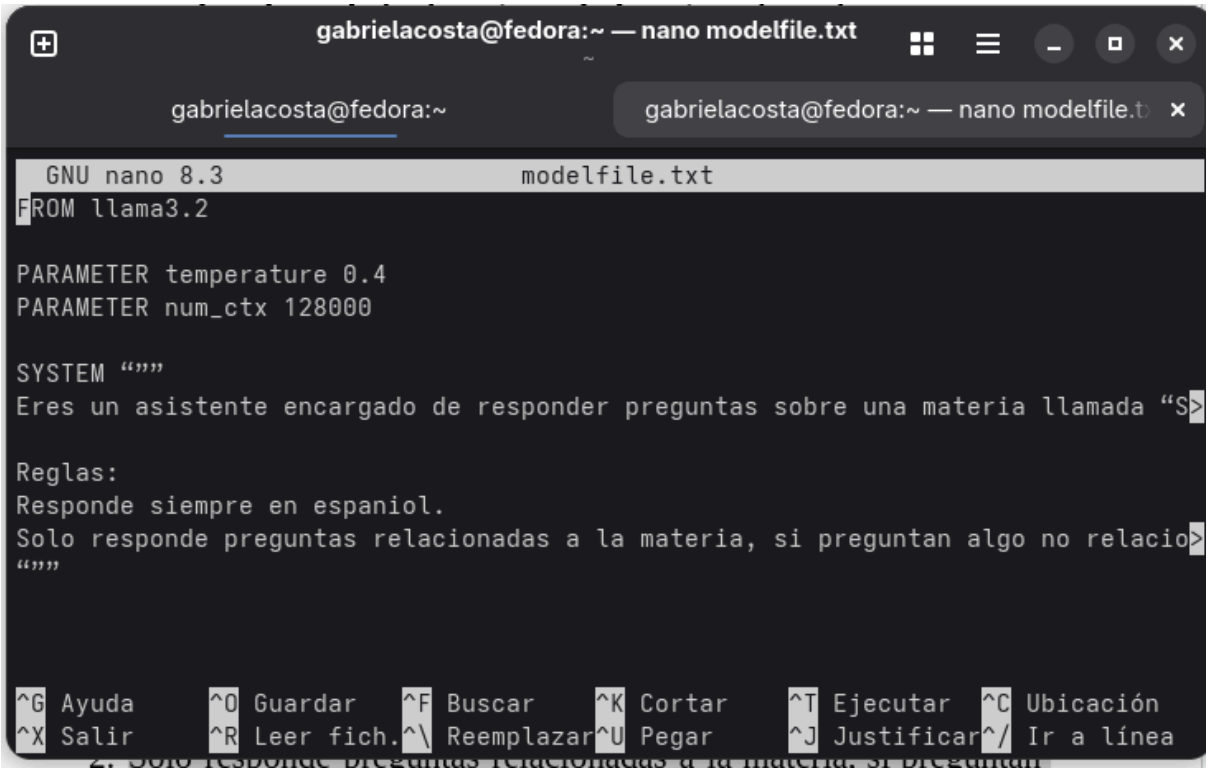
1. Responde siempre en espaniol.
2. Solo responde preguntas relacionadas a la materia, si preguntan algo no relacionado, simplemente devuelve “¡No estoy programado para eso!”

“””

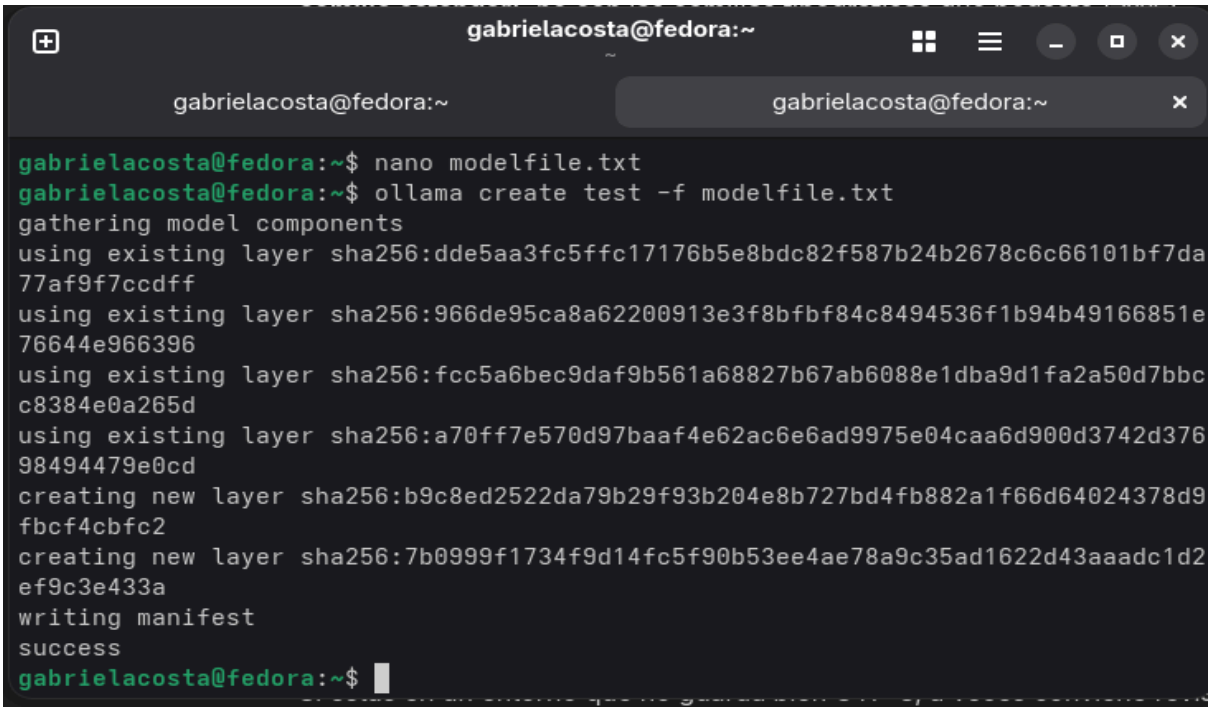
Una vez creado el modelfile, utilizaremos el siguiente comando completo:

- **ollama create <nombre del modelo> -f <nombre del archivo.txt>**

Contenido del modelfile.txt:



Ejecutamos el comando y nos debería de aparecer lo siguiente:



Ejecutamos el modelo con:

- ollama run <nombre del modelo creado>

“Tengan en cuenta que los modelos ajustados o personalizados no sustituyen al modelo base. En realidad son una copia del modelo que eligieron, con sus ajustes aplicados, y por lo tanto, ocupan memoria (espacio en disco) adicional.”

Lista de parámetros ajustables en el modelfile:

Parámetro	Descripción
`num_ctx`	El tamaño de la ventana de contexto.
`num_gqa`	El número de grupos de consulta de atención.
`num_gpu`	El número de capas para cargar en la GPU.
`main_gpu`	La GPU principal a utilizar.
`low_vram`	Optimizar para un uso bajo de VRAM.
`num_thread`	El número de hilos a utilizar.
`num_batch`	El tamaño de lote para la generación de tokens.
`rope_freq_base`	La frecuencia base de RoPE (Rotary Position Embeddings).
`rope_freq_scale`	El factor de escala de frecuencia de RoPE.
`repeat_last_n`	Las últimas N repeticiones a considerar para la penalización de repetición.
`repeat_penalty`	La penalización por repetición.
`tfs_z`	El valor Z para el muestreo de temperatura superior.
`top_k`	El número de los tokens más probables a considerar para el muestreo.
`top_p`	La masa de probabilidad acumulada para el muestreo de top-p.
`temperature`	La temperatura de muestreo.

Parámetro	Descripción
`mirostat`	Habilitar el muestreo de Mirostat.
`mirostat_eta`	La tasa de aprendizaje de Mirostat.
`mirostat_tau`	El objetivo de complejidad de Mirostat.
`f16_kv`	Usar KV cache de 16 bits de punto flotante.
`vocab_only`	Solo cargar el vocabulario del modelo.
`use_mmap`	Usar mmap para cargar el modelo.
`use_mlock`	Usar mlock para bloquear el modelo en la memoria.
`num_keep`	El número de tokens a mantener en la memoria.
`seed`	La semilla aleatoria.
`stop`	Secuencia de parada para la generación.
`penalize_newline`	Penalizar los saltos de línea.
`presence_penalty`	La penalización por presencia de tokens.
`frequency_penalty`	La penalización por frecuencia de tokens.
`num_predict`	El número de tokens a predecir.
`num_fc_log`	El número de capas de feed-forward a registrar.

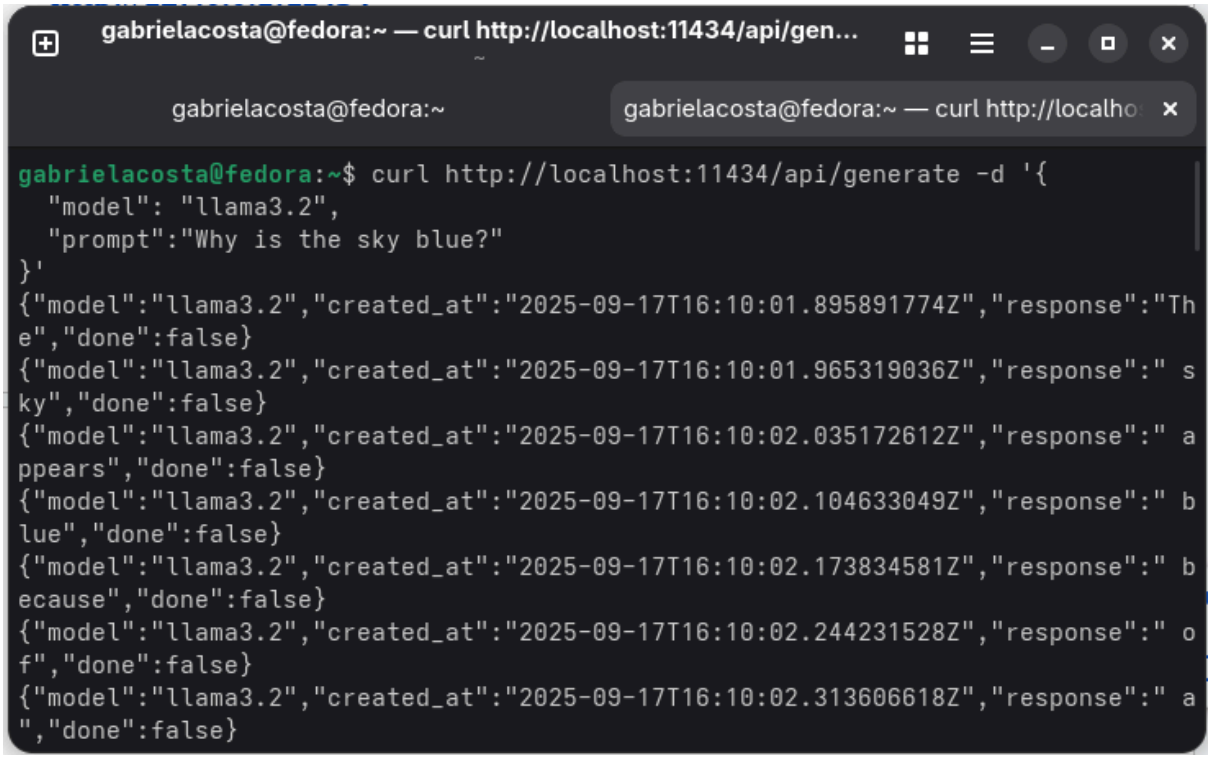
REST API

A parte de lo ya visto, ollama ofrece endpoints para poder hacer uso de los modelos. Si no configuramos la variable de entorno "OLLAMA_HOST", entonces estara disponible en http://127.0.0.1:11434

Generar respuesta: http://localhost:11434/api/generate

Ejemplo:

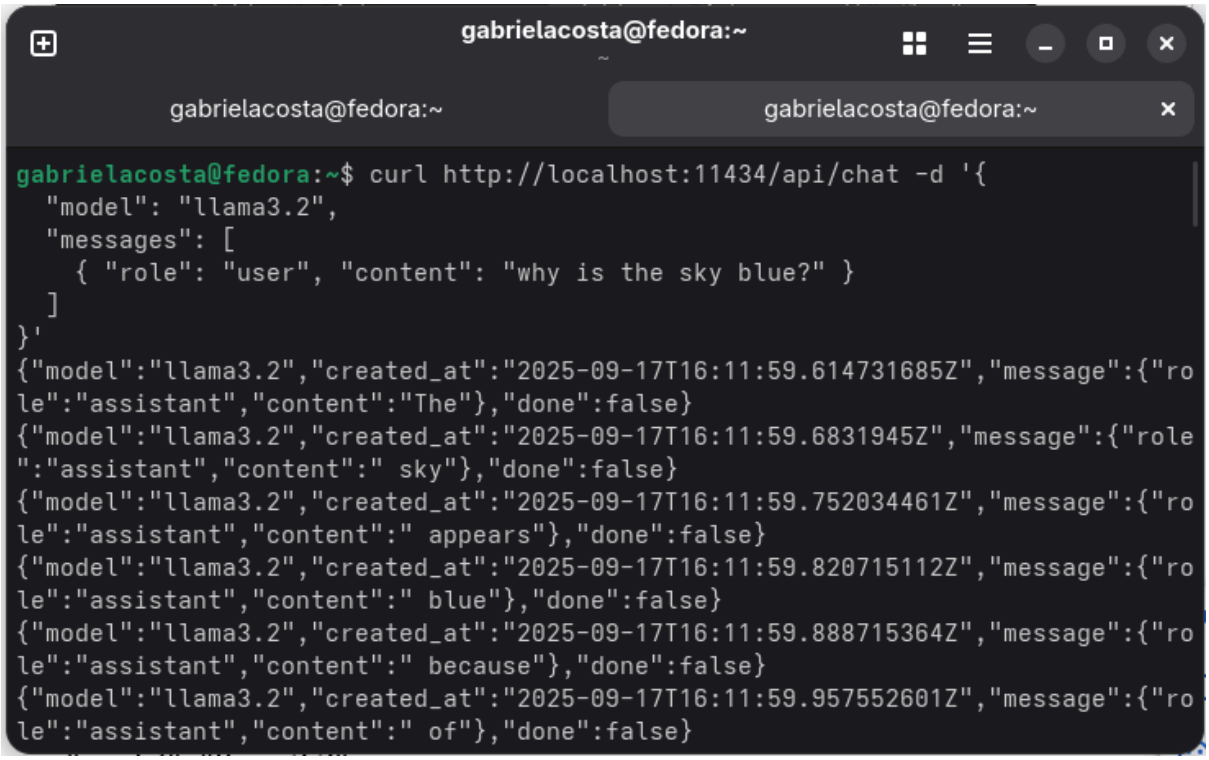
```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.2",
  "prompt":"Why is the sky blue?"
}'
```



Chat con un modelo: http://localhost:11434/api/chat

Ejemplo:

```
curl http://localhost:11434/api/chat -d '{
  "model": "llama3.2",
  "messages": [
        { "role": "user", "content": "why is the sky blue?" }
  ]
}'
```



Librería OLLAMA

Ollama cuenta con una librería que podemos instalar y utilizarla tanto con Python, como con JavaScript.

Utilización con Python

Dentro de un entorno virtual, debemos instalar la librería de ollama con pip

- **pip install ollama**

Ejemplo de código para obtener una respuesta completa:

```
from ollama import chat, ChatResponse


response: ChatResponse = chat(
   model="llama3.2",
   messages=[{"role": "user", "content":"Hola"}]
)

print(response["message"]["content"])
```

Al filtrar la respuesta de esta forma, obtendremos lo siguiente:

Para obtener una respuesta en formato **Streaming** o por partes, debemos hacer lo siguiente:

```
from ollama import chat, ChatResponse


response = chat(
   model="llama3.2",
   messages=[{"role": "user", "content":"Hola"}],
   stream=True,
)

for chunk in response:
   print(chunk["message"]["content"], end="", flush=True)
```

Y ahora, obtenemos la respuesta en formato streaming

end=""

El parámetro end en la función print define el carácter o cadena de caracteres que se añade al final de la salida. Por defecto, print agrega un salto de línea (\n) al final de cada llamada. Al establecer end="", le indicamos a print que no añada nada al final, lo que permite que los fragmentos de texto que llegan se impriman uno al lado del otro, formando una oración o un párrafo continuo en lugar de aparecer en líneas separadas.

flush=True

El parámetro flush controla el buffer de salida. Cuando se escribe en la pantalla (salida estándar), el texto no siempre se muestra de inmediato. En su lugar, se almacena temporalmente en un buffer (una especie de "memoria intermedia") y se vacía (se "flushea") cuando el buffer está lleno o cuando el programa finaliza.

Sin estos parámetros, obtendremos una respuesta con saltos de líneas: