# MÓDULO 2: CONFIGURACIÓN DEL ENTORNO Y PRIMEROS PASOS

# Agenda del Módulo 2

1

## Preparación del Entorno de Desarrollo

- Configuración de Python y entornos virtuales
- Instalación de librerías esenciales
- Acceso a modelos en Hugging Face
- Verificación de requisitos de hardware

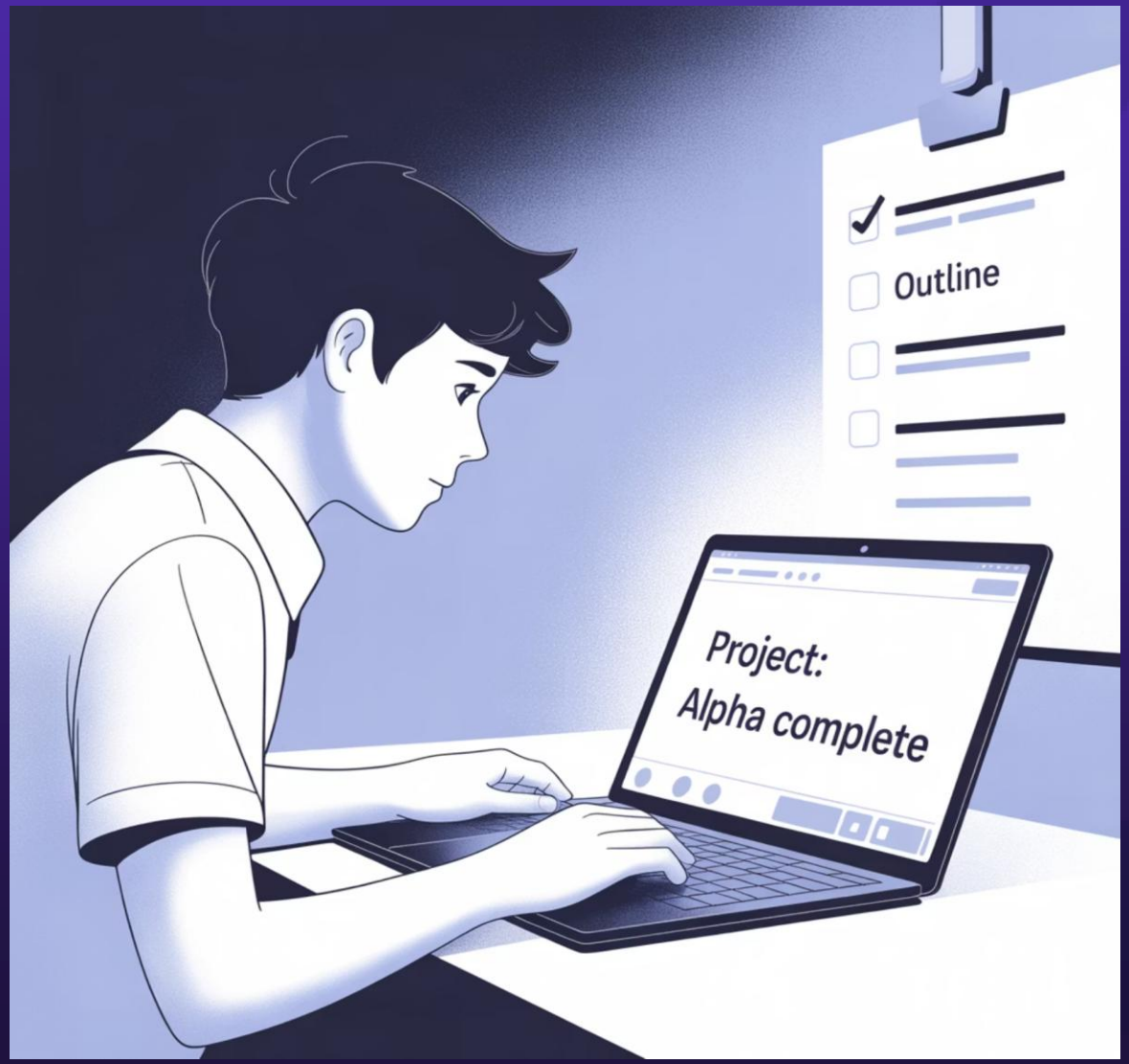
2

## Primera Interacción con LLaMa 3.1

- Carga y ejecución local del modelo
- Parámetros básicos de inferencia
- Experimentación con prompts
- Optimización de recursos

En este módulo, estableceremos las bases técnicas necesarias para trabajar eficazmente con LLaMa 3.1.

# Objetivos de Aprendizaje




### Crear un entorno de desarrollo funcional

Configurar correctamente todas las herramientas necesarias para ejecutar LLaMa 3.1 sin complicaciones técnicas.

### Ejecutar el modelo localmente

Cargar y utilizar LLaMa 3.1 8B en su propio equipo, entendiendo los requisitos y limitaciones.

### Dominar los parámetros básicos

Comprender y manipular los parámetros fundamentales que afectan la generación de texto.

# Parte 1: Preparación del Entorno

Instalar Python

Configurar entorno

Instalar dependencias

Descargar LLaMa 3.1

Verificar configuración

Un entorno correctamente configurado es fundamental para evitar errores técnicos y concentrarnos en el aprendizaje de LLaMa 3.1.

# Requisitos de Hardware

## Mínimos

- CPU: 4 núcleos o más
- RAM: 16GB mínimo (para modelo 8B)
- Almacenamiento: 20GB libres
- Sistema operativo: Windows 10/11, macOS, Linux

## Recomendados

- CPU: 8+ núcleos
- RAM: 32GB o más
- GPU: NVIDIA con 8GB+ VRAM
- Almacenamiento SSD: 50GB libres

La ejecución de LLaMa 3.1 8B consume considerables recursos. Sin GPU, la inferencia será significativamente más lenta pero posible. Para modelos más grandes (70B), una GPU potente es prácticamente indispensable.

# Instalación de Python

## Pasos de instalación:

- Descargar Python 3.10+ desde python.org
- Marcar "Add Python to PATH" durante la instalación
- Verificar la instalación: python –versión
- Actualizar pip: python -m pip install --upgrade pip

Recomendación: Utilizar Python 3.10 o 3.11 para máxima compatibilidad con las bibliotecas necesarias para LLaMa 3.1.

# Instalación Visual Studio Code

- Descargar instalador desde code.visualstudio.com
  - Ejecuta el archivo y sigue los pasos con la configuración por defecto.
  - Instalar Extensión de Python:
    - Abre VS Code y ve a la pestaña de **Extensiones**
    - Busca Python (publicado por Microsoft)
    - Haz clic en "Install"

# Creando un Proyecto para LLaMa 3.1

- Crear una Carpeta para el Proyecto:
  - En tu explorador de archivos (Windows) o Finder (macOS), crea una nueva carpeta
  - Nómbrala de forma descriptiva, por ejemplo: proyecto-llama3
- Abrir la Carpeta en Visual Studio Code:
  - Abre VS Code
  - Dentro de VS Code, en el panel del explorador, haz clic derecho y selecciona Nuevo Archivo
  - Nombra tu archivo principal, por ejemplo: app.py

# Entornos Virtuales: ¿Por qué son importantes?

multi versica ACADEMY

### Aislamiento de dependencias

Evita conflictos entre paquetes de diferentes proyectos, manteniendo versiones específicas para cada aplicación.

### Reproducibilidad

Facilita compartir exactamente el mismo entorno con otros desarrolladores o en diferentes máquinas.

### Gestión limpia

Permite eliminar completamente un entorno sin afectar al sistema o a otros proyectos.

Para LLaMa 3.1, un entorno virtual es crucial debido a las versiones específicas de PyTorch y Transformers que requiere.

# Creando el Entorno Virtual en VS Code

- **Abrir la Paleta de Comandos:**
  - Usa el atajo Ctrl+Shift+P (en Windows) o Cmd+Shift+P (en macOS)
- **Ejecutar el Comando de Creación:**
  - Escribe Python: Create Environment... y selecciona esa opción
- **Configurar el Entorno:**
  - Elige Venv como el tipo de entorno
  - Selecciona la versión de Python que deseas usar (la que instalaste previamente)
- **Resultado:**
  - VS Code creará una carpeta .venv dentro de tu proyecto
  - El entorno se activará automáticamente, lo que verás indicado en la terminal de VS Code con un (.venv)
  - puedes instalar las librerías (transformers, torch, etc.) de forma segura

# Instalación de Librerías Esenciales

- # Dentro del entorno virtual activado
- pip install torch torchvision torchaudio
- # Bibliotecas fundamentales para LLaMa
- pip install transformers accelerate bitsandbytes
- # Utilidades adicionales recomendadas
- pip install sentencepiece huggingface_hub pandas numpy

# Configuración para GPU (Opcional)

## NVIDIA CUDA

1. Instalar drivers NVIDIA actualizados
2. Instalar CUDA Toolkit (11.8+ recomendado)
3. Verificar: `nvidia-smi`
4. Instalar PyTorch con soporte CUDA

## Verificación

```
import torchprint(f"PyTorch: {torch.__version__}")print(f"CUDA disponible: {torch.cuda.is_available()}")if torch.cuda.is_available(): print(f"Dispositivo: {torch.cuda.get_device_name(0)}")
```

La aceleración por GPU puede mejorar la velocidad de inferencia hasta 10-20 veces, especialmente para generación de texto extensa.

# Acceso a Hugging Face Hub

## Crear cuenta en Hugging Face

Registrarse en huggingface.co y verificar el correo electrónico.

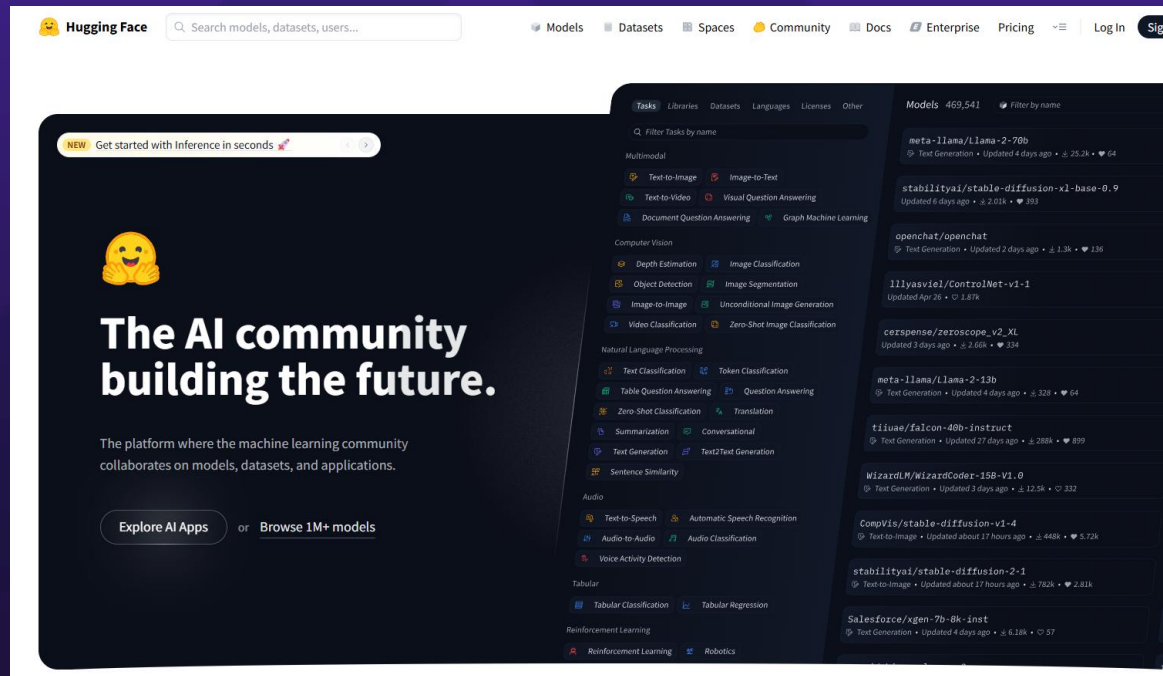
## Generar token de acceso

En Settings > Access Tokens, crear un nuevo token con permisos de lectura.

## Autenticar CLI de Hugging Face

Ejecutar `huggingface-cli login` e ingresar el token generado.

El acceso autenticado permite descargar modelos con restricciones y aumenta los límites de descarga para modelos grandes como LLaMa 3.1.



# Problemas Comunes y Soluciones

### Error: "CUDA out of memory"

Solución: Reducir el tamaño del modelo, activar la cuantificación con bitsandbytes, o aumentar la VRAM disponible.

### Error: "No module named X"

Solución: Verificar que el entorno virtual está activado y reinstalar la librería faltante con pip.

### Descarga lenta o interrumpida

Solución: Utilizar huggingface_hub con resume_download=True o descargar manualmente los archivos desde la web.

La mayoría de los problemas tienen soluciones documentadas en GitHub o foros de Hugging Face.

Siempre busque el mensaje de error exacto.

# Parte 2: Primera Interacción con LLaMa 3.1

Ahora que tenemos nuestro entorno configurado, es momento de poner en marcha LLaMa 3.1 y ver su potencial en acción

# Modelos Disponibles de LLaMa 3.1

| Modelo | Parámetros | Contexto | RAM mínima |
|---|---|---|---|
| Meta-Llama-3.1-8B | 8 mil millones | 8K tokens | ~16GB |
| Meta-Llama-3.1-8B-Instruct | 8 mil millones | 8K tokens | ~16GB |
| Meta-Llama-3.1-70B | 70 mil millones | 8K tokens | ~140GB |
| Meta-Llama-3.1-70B-Instruct | 70 mil millones | 8K tokens | ~140GB |

Para este curso, nos enfocaremos en el modelo de 8B, que ofrece un buen equilibrio entre rendimiento y requisitos de hardware. Los modelos "Instruct" están optimizados para seguir instrucciones.

# Carga del Modelo LLaMa 3.1

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch                                    # Identificador del modelo en Hugging
Facemodel_id = "meta-llama/Meta-Llama-3.1-8B“ # Cargar el tokenizador
tokenizer = AutoTokenizer.from_pretrained(model_id)  # Cargar el modelo (con
optimizaciones)
model = AutoModelForCausalLM.from_pretrained(    model_id,    device_map="auto",
                              # Distribución automática en dispositivos
torch_dtype=torch.float16,    # Precisión reducida para ahorrar memoria
low_cpu_mem_usage=True        # Optimización para CPU)
```

El parámetro device_map="auto" asigna automáticamente las capas del modelo a CPU/GPU según la memoria disponible.

# Optimización de Memoria

## Cuantización

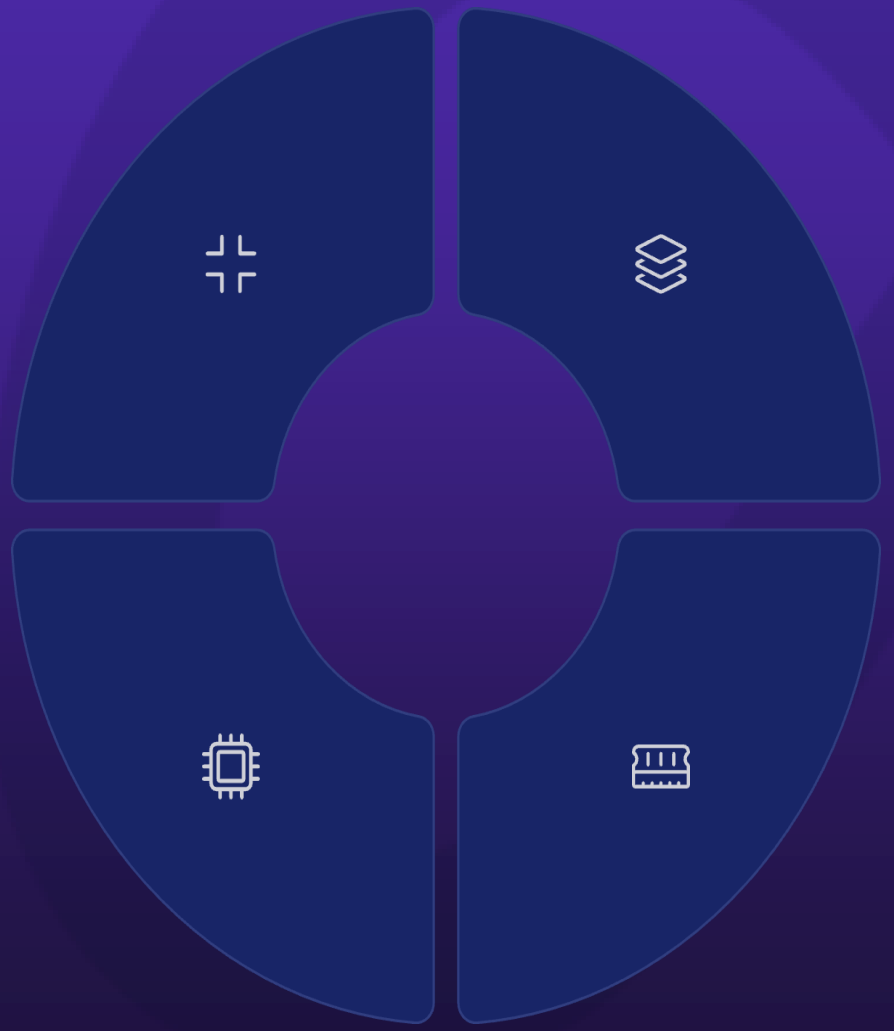
Reduce la precisión numérica (ej. 16-bit a 8-bit o 4-bit)

```
load_in_8bit=True
```

## Offloading

Mueve capas entre GPU y CPU según necesidad

```
device_map="auto"
```



## CPU Offload

Procesa parte del modelo en CPU

```
offload_folder="offload"
```

## Gestión de atención

Procesa la atención por fragmentos

```
attn_implementation="flash_attention_2"
```

# Demostración Práctica

Realizaremos una demostración del uso de Llama 3.1.

# Parámetros de Generación

## max_new_tokens

Número máximo de tokens a generar. Mayor valor = respuestas más largas, pero más tiempo de procesamiento.

Ejemplo: 128, 256, 512, 1024

## temperature

Controla la aleatoriedad. Valores más altos producen respuestas más creativas pero menos predecibles.

Rango: 0.1 (determinista) a 1.5 (muy aleatorio)

## top_p

Sampling por núcleo de probabilidad. Selecciona tokens con probabilidad acumulada hasta este valor.

Rango típico: 0.7 a 0.95

# Más Parámetros de Generación

## do_sample

Si es False, usa greedy decoding (siempre el token más probable). Si es True, permite muestreo probabilístico.

Valores: True/False

## repetition_penalty

Penaliza la repetición de tokens ya generados. Útil para evitar bucles.

Rango típico: 1.0 (sin penalización) a 1.5

## no_repeat_ngram_size

Evita repetir secuencias de n-gramas. Un valor de 3 evita repetir trigramas.

Valores comunes: 0 (desactivado), 2, 3, 4

La configuración óptima depende del caso de uso: para código se prefiere baja temperatura, para creatividad valores más altos.

# Experimentación con Parámetros


El gráfico muestra cómo el aumento de temperatura afecta diferentes aspectos del comportamiento del modelo.

A mayor temperatura, mayor creatividad pero menor precisión y determinismo.

# Ejemplo Práctico: Diferentes Temperaturas

multi versica ACADEMY

## Temperatura = 0.2

> El calentamiento global es el aumento a largo plazo de la temperatura media del sistema climático de la Tierra. Es causado principalmente por emisiones de gases de efecto invernadero producidas por actividades humanas como la quema de combustibles fósiles y la deforestación. Sus efectos incluyen aumento del nivel del mar, cambios en patrones climáticos y eventos meteorológicos extremos.

## Temperatura = 1.0

> El calentamiento global representa uno de los mayores desafíos de nuestro tiempo. Este fenómeno, que eleva gradualmente las temperaturas planetarias, no solo derrite glaciares y altera ecosistemas, sino que también desencadena una cascada de consecuencias interconectadas: desde la acidificación oceánica hasta la intensificación de huracanes, pasando por la alteración de los ciclos agrícolas que sostienen nuestra civilización.

**Prompt utilizado: "Explica qué es el calentamiento global"**

# Técnicas Avanzadas de Prompting

1

## Prompting en cadena (Chain-of-Thought)

Solicite al modelo que muestre su razonamiento paso a paso antes de llegar a una conclusión final.

**Ejemplo:** "Resuelve este problema matemático paso a paso, explicando cada operación: $3x^2 + 6x - 2 = 0$"

2

## Prompting con ejemplos (Few-Shot)

Proporcione ejemplos del tipo de respuesta que espera antes de hacer su pregunta principal.

**Ejemplo:** "Clasifica estos textos como positivos o negativos. Ejemplo positivo: 'Me encantó este producto'. Ejemplo negativo: 'No funcionó como esperaba'. Texto a clasificar: '...'"

3

## Prompting con rol (Role Prompting)

Asigne un rol específico al modelo para obtener respuestas desde una perspectiva particular.

**Ejemplo:** "Actúa como un profesor de física explicando el concepto de relatividad a estudiantes universitarios de primer año."