

# Inteligencia Artificial Generativa Para la Ciencia de Datos



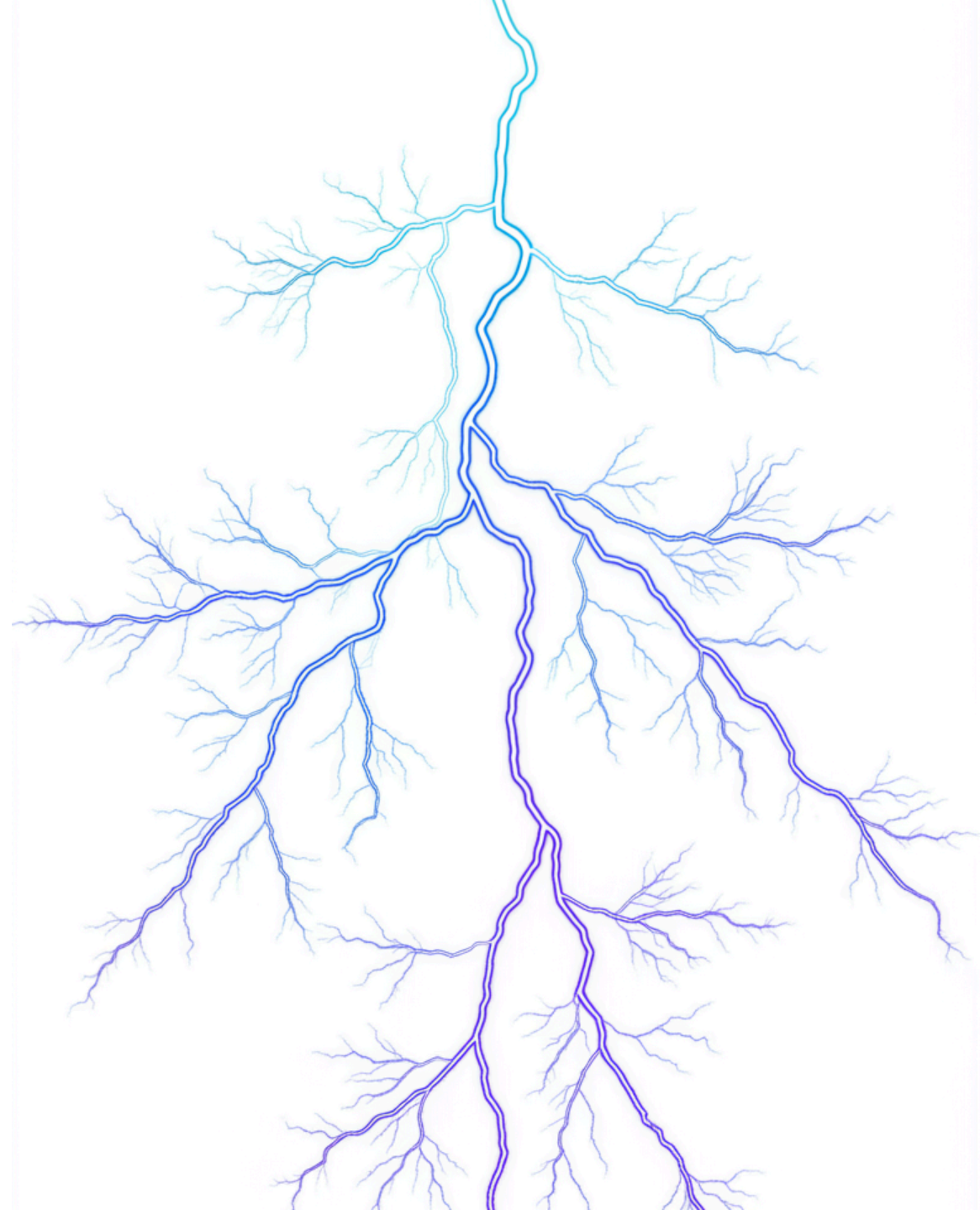
Profesor

Juan Camilo Vega Barbosa

*Consultor IA - Ingeniero IA/ML*

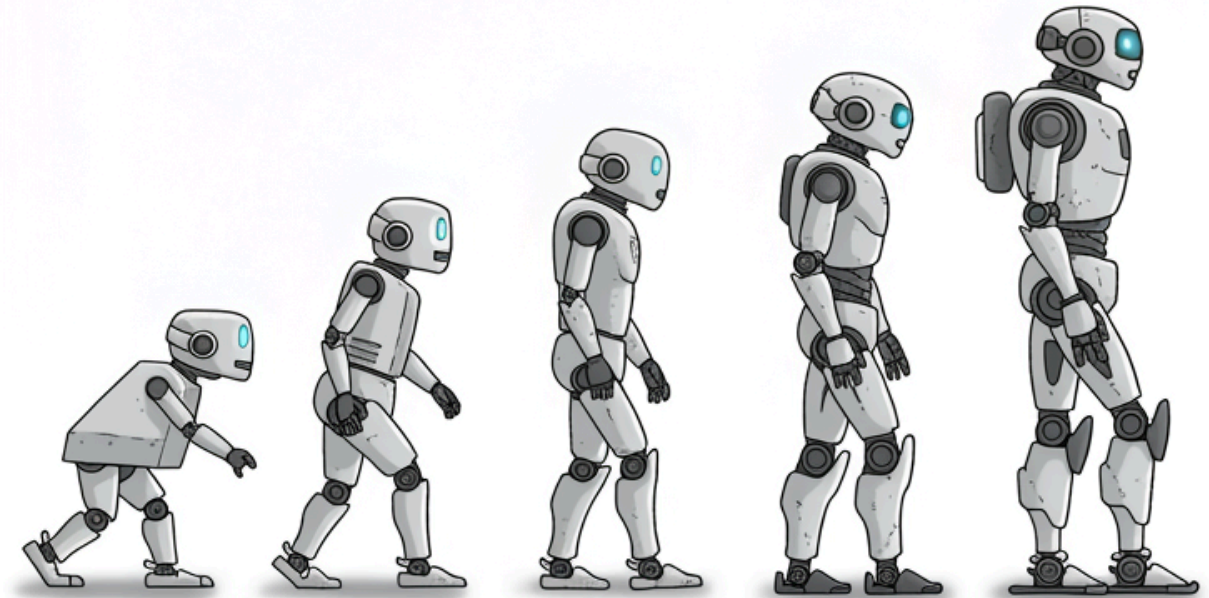


LinkedIn



# Optimización y Ejecución de Modelos de Lenguaje

Stack de trabajo para LLM





## Google Colab: Nuestro laboratorio en la nube

Google Colab es nuestra **plataforma principal** para experimentar con LLMs 🚀. Ofrece **GPUs** gratuitas (Tesla T4) y elimina barreras técnicas para acceder a hardware de deep learning.



### Ventajas clave

- GPU gratuita: Tesla T4 con 16GB VRAM
- Setup cero: Python preinstalado
- Colaborativo: Compartir notebooks
- Escalable: Upgrade a Colab Pro



**Tips de experiencia:** - Secrets: Usar 🔑 para API keys - Restart runtime: Si error de memoria - Mantener ventana activa



**Acceso:** [colab.research.google.com](https://colab.research.google.com)

# Hugging Face: El GitHub de la IA

Hugging Face es el ecosistema central para modelos de lenguaje 🏗️. Combina repositorio de modelos, datasets y herramientas que democratizan el acceso a IA de vanguardia.

## Componentes principales

- **Model Hub:** +500k modelos pre-entrenados
- **Transformers:** Biblioteca unificada
- **Accelerate:** Optimización automática
- **Spaces:** Demos interactivos

**Importancia estratégica** HF eliminó la fricción para usar modelos state-of-the-art. Antes: semanas. Ahora: 3 líneas de código.

 Acceso: [huggingface.co](https://huggingface.co)

Token: Requerido para Llama-2

# Stack tecnológico completo

Nuestro stack integra lo mejor de cada herramienta para crear un ambiente completo de desarrollo con LLMs .

Arquitectura: Colab (compute) + HF (modelos) + APIs externas (escalabilidad)

## Infraestructura

- Google Colab: Compute y GPUs
- Hugging Face: Modelos y datasets
- Ollama: Deployment local
- Groq: APIs ultra-rápidas

## Enlaces importantes

- Colab: [colab.research.google.com](https://colab.research.google.com)
- HF Hub: [huggingface.co](https://huggingface.co)
- Ollama: [ollama.ai](https://ollama.ai)
- Groq: [console.groq.com](https://console.groq.com)



## ¿Por qué necesitamos optimización?


Los modelos de lenguaje modernos son masivos 🏗️. Por ejemplo, Llama-2-7b tiene **7 mil millones de parámetros**, requiriendo **~28 GB solo para cargar**. Durante inferencia se agregan **activaciones y cache KV**, duplicando o triplicando el uso de memoria.


Sin optimización, ejecutar Llama-2-7b requiere GPUs con +40GB de VRAM, limitando acceso a hardware empresarial costoso.

## El costo computacional real

### Números que importan

- Llama-2-7b: ~28 GB parámetros
- + Activaciones: ~15-25 GB adicionales
- + Cache KV: Crece con contexto
- Total: 50-80 GB inferencia óptima

 **Objetivo optimizaciones:** Ejecutar en hardware accesible sin sacrificar rendimiento

 **Experiencia:** Cache KV crece exponencialmente. 2K tokens = 4GB, 8K tokens = 16GB adicionales.

## 🔥 ¿Por qué las GPUs son cruciales?

Las GPUs fueron diseñadas para gráficos 3D pero resultan perfectas para deep learning 🎮➡️🧠. CPUs: pocos núcleos potentes. GPUs: miles de núcleos simples trabajando en paralelo.

### 💻 CPU vs GPU

- CPU: 8-16 núcleos, ~3 GHz
- GPU: 2,000-10,000 núcleos, ~1 GHz
- Paralelismo: Secuencial vs Masivo

💡 **Tip clave:** Para LLMs, \*\*VRAM es más crítica que velocidad\*\*. Si no cabe en memoria, es imposible ejecutarlo.

**Prioridad:** VRAM > Núcleos CUDA > Velocidad



## Técnicas: Cuantización

La cuantización reduce precisión numérica  $\frac{1}{32}$   $\rightarrow$   $\frac{1}{32}$ . En lugar de 32 bits (float32), usar 16 bits (float16) o 8 bits (int8) con pérdida mínima.



### Tipos de cuantización

- FP32: Precisión completa
- FP16: Mitad memoria, ~mismo rendimiento
- INT8: Cuarto memoria, ligera pérdida




### Impacto en Llama-2-7b

- FP32: ~28 GB
- FP16: ~14 GB
- INT8: ~7 GB

**BitsAndBytes** mantiene parámetros críticos en alta precisión mientras cuantiza el resto.

## Técnicas: Accelerate

Accelerate maneja distribución automática del modelo y optimiza memoria . Permite **device mapping** inteligente y **offloading** a CPU.

### Beneficios

- Auto-distribución de capas
- Memory offloading inteligente
- Mixed precision automática

### Casos de uso


- GPU limitada: Offload a CPU
- Multi-GPU: Distribución automática
- Memoria insuficiente: Checkpointing

# Nuestros 4 Métodos

Implementamos **cuatro enfoques diferentes** para ejecutar Llama-2-7b, optimizados para distintos escenarios 🧠.

Método	Complejidad	Recursos	Velocidad	Caso de uso
Pipeline 🚀	Baja	Alta GPU	Media	Prototipos
Optimizado ⚡	Media	Media GPU	Alta	Producción
Ollama 🐳	Baja	Baja GPU	Media	Desarrollo
Groq ☁️	Mínima	Sin GPU	Muy Alta	Apps rápidas

# Método 1: Pipeline

El pipeline es la forma más simple . Hugging Face abstrae complejidad pero **sacrifica control** y eficiencia.

## Ventajas

- **Setup inmediato:** 3 líneas código
- **Sin configuración:** Todo automático
- **Ideal prototipos:** Pruebas rápidas

## Limitaciones

- **Alto uso memoria:** Sin optimizaciones
- **Velocidad limitada:** No aprovecha hardware

¿Cuándo usarlo? - Primeros experimentos con LLMs - Validación rápida de conceptos - Aprendizaje conceptos básicos

Perfecto para entender LLMs sin preocuparse por detalles técnicos.

## Método 2: Accelerate + BitsAndBytes

Combinación estado del arte en optimización 🏆. Accelerate maneja distribución mientras BitsAndBytes aplica cuantización inteligente.

### Características

- Cuantización 8-bit: Reduce 75% memoria
- Mixed precision: FP16 velocidad
- Device mapping: Distribución automática

### Beneficios



- Cabe en T4: 16GB suficientes
- Velocidad óptima: Máximo rendimiento

#### Técnica favorita producción

Configuración que usan startups IA para ejecutar modelos grandes en hardware accesible.

**Ideal para:** Aplicaciones producción, máximo rendimiento, calidad crítica

## Método 3: Ollama

Ollama está diseñado para ejecución local , pero lo ejecutamos en Google Colab usando comandos Linux . Abstrae complejidad técnica completamente.

### Características

- Instalación: `curl | sh` en Colab
- Gestión: `ollama pull llama2:7b`
- Servidor: API REST localhost:11434


### En Colab

- Simula entorno local
- Comandos: `curl` , `nohup` , `pkill`
- Aprendizaje: Deploy en servidores

¿Por qué en Colab? - Simular servidor Linux  
- Aprender comandos administración -  
Entender APIs REST locales

 Info: [ollama.ai](https://ollama.ai)

## Método 4: Groq

Groq ofrece inferencia **ultra-rápida** mediante hardware especializado (LPUs) . **Arquitectura determinística** elimina variabilidad en latencia.

### Ventajas técnicas

- **Velocidad extrema:** 500+ tokens/segundo
- **Latencia baja:** <100ms respuesta
- **Sin hardware local:** Todo en nube

### Modelo económico

- **Tier gratuito:** Generoso desarrollo
- **Pay-per-use:** Solo pagas lo usado

**Opción enterprise** Para aplicaciones que requieren respuestas instantáneas.

 **Registro:** [console.groq.com](https://console.groq.com)

**Casos:** Chatbots tiempo real, APIs alta frecuencia

# Comparación de Métodos

Método	Velocidad	Uso GPU	Configuración	Costo	Ideal para
Pipeline	Media	Alta	Fácil	Gratis	Prototipos
Optimizado	Alta	Media	Intermedia	Gratis	Producción local
Ollama	Media	Baja	Fácil	Gratis	Desarrollo
Groq	Muy Alta	Ninguna	Fácil	Freemium	Apps rápidas

## Recomendaciones

- Para aprender: Pipeline
- Para máximo rendimiento: Optimizado
- Para desarrollo local: Ollama
- Para producción: Groq




## Parte práctica


Abran el  [Script\\_Sesion2](#) que está dentro del repo del curso.

- **Hands-on:** Implementación 4 métodos
- **Configuración:** Setup APIs y secretos
- **Comparación:** Medición rendimiento

## Recursos del Curso

### Repositorio GitHub

 **Acceso permanente** a notebooks, scripts, datasets y documentación técnica completa.

 **Actualizaciones continuas** con las últimas versiones y nuevos casos de uso empresariales.

### Enlaces:

 **GitHub del curso**

 **Profesor:**

[LinkedIn - Camilo Vega](#)