

Inteligencia Artificial Generativa Para la Ciencia de Datos



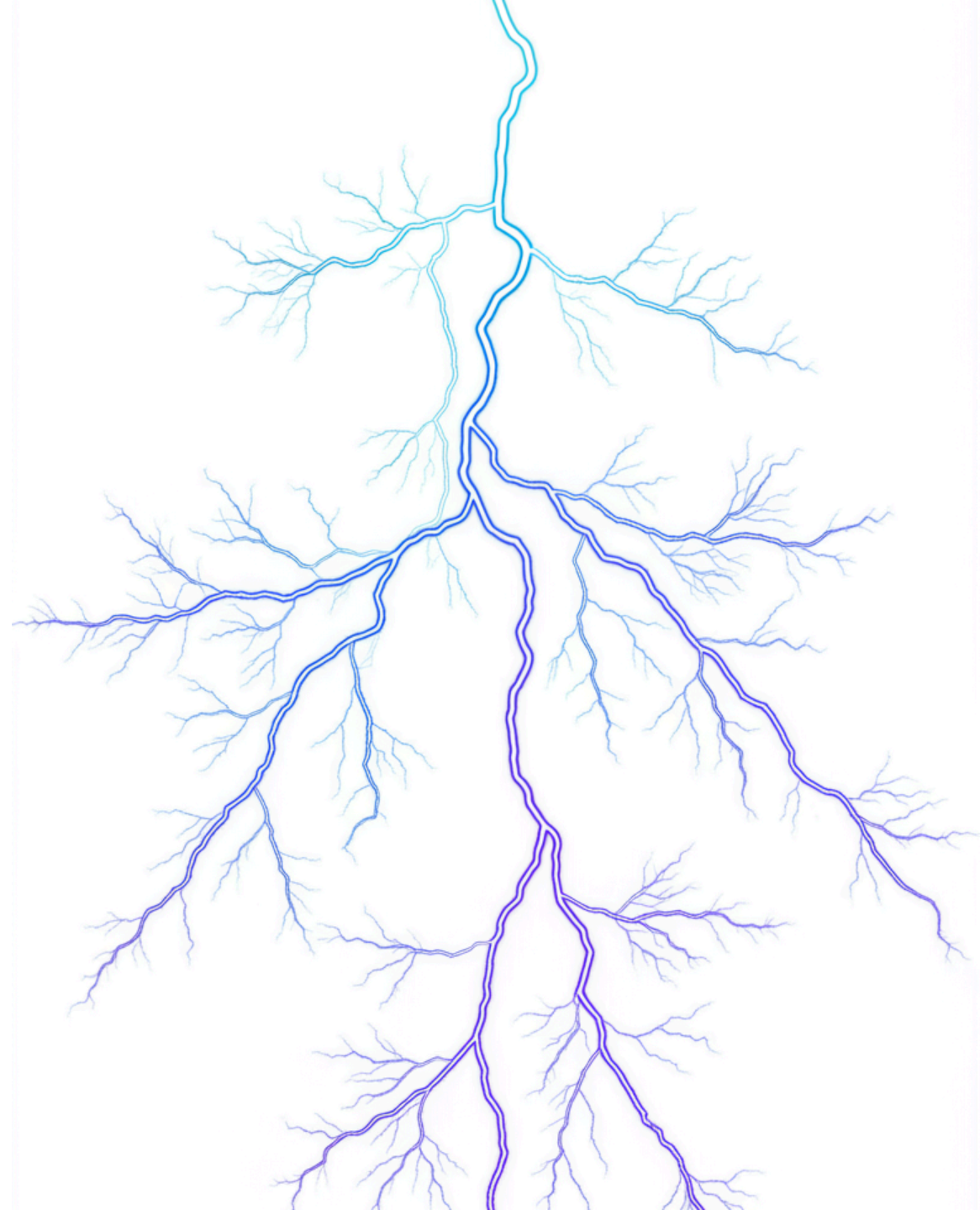
Profesor

Juan Camilo Vega Barbosa

Consultor IA - Ingeniero IA/ML

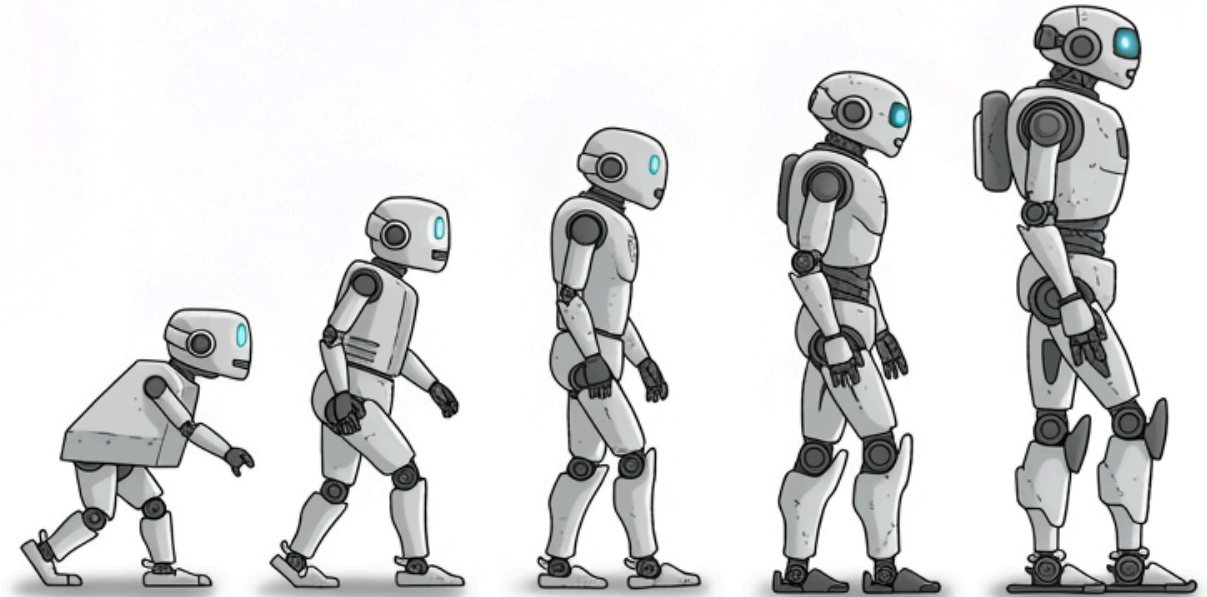


LinkedIn



Fine-Tuning y RAG para Ciencia de Datos

Adaptando LLMs a dominios
específicos





Superando las limitaciones de los LLM en ciencia de datos

Los **modelos de lenguaje tienen limitaciones inherentes** en análisis de datos 📊. Conocimiento desactualizado sobre metodologías, incapacidad para acceder a datasets privados empresariales, y tendencia a generar interpretaciones estadísticas incorrectas.

En ciencia de datos, la precisión no es opcional. Una interpretación errónea de correlaciones puede llevar a decisiones empresariales millonarias incorrectas.

Existen dos enfoques principales para superar estas barreras: el **Fine-tuning** permite enseñar al modelo metodologías específicas del dominio, mientras que **RAG** lo conecta con bases de datos actualizadas y documentación técnica especializada.



Modelos fundacionales: La base para especialización

Los modelos fundacionales son LLMs pre-entrenados a gran escala que sirven como base reutilizable 🏛️. En lugar de entrenar un modelo desde cero para cada dominio analítico, podemos especializar modelos existentes.



Beneficios económicos

- **99% reducción en costos:** De \$500K a \$5K para entrenar
- **Tiempo:** De 6 meses a 1 semana
- **Sustentabilidad:** 552 toneladas CO₂ → 5 toneladas



Ejemplos para ciencia de datos

- **CodeLlama** → Fine-tuned para SQL empresarial
- **Llama-2** → Especializado en interpretación estadística
- **Mistral** → Adaptado para análisis de series temporales

Fine-Tuning: Especializando modelos para dominios analíticos

El Fine-tuning permite adaptar LLMs a metodologías específicas de ciencia de datos 🧠, ajustando sus parámetros para aprender patrones de análisis, terminología técnica, y mejores prácticas de interpretación de resultados en contextos específicos como finanzas, salud o marketing.

Casos de uso en Data Science

- **Interpretación de modelos ML:** Explicar Random Forest, XGBoost
- **Análisis estadístico:** Tests de hipótesis especializados
- **Código domain-specific:** SQL para retail, Python para fintech


Ejemplo real: Un modelo fine-tuneado para análisis financiero puede distinguir entre correlación espuria y causal en series temporales, algo crítico para modelos de riesgo.

Proceso técnico del Fine-Tuning

1. Preparación de datasets especializados: - Pares pregunta-respuesta sobre metodologías específicas - Código comentado con mejores prácticas del dominio - Interpretaciones correctas de resultados estadísticos

2. Entrenamiento continuo

- **Tasas de aprendizaje bajas:** Evitar "catastrophic forgetting"
- **Validación cruzada:** Evaluar en datasets hold-out
- **Early stopping:** Prevenir sobreajuste

 **Tip crítico:** En ciencia de datos, el fine-tuning debe balancear conocimiento general con especialización técnica para evitar sesgos metodológicos.

PEFT: Eficiencia en el fine-tuning

Parameter-Efficient Fine-Tuning (PEFT) ajusta solo 0.1-1% de parámetros ⚡, manteniendo eficiencia computacional mientras logra especialización efectiva.

LoRA en acción

- **Matrices originales:** Congeladas (99.9%)
- **Adaptadores pequeños:** $A(4096 \times 8) \times B(8 \times 4096)$
- **Resultado:** Ajuste con mínimos recursos

Ventajas PEFT

- 10x menos memoria GPU
- 5x más rápido

Ejemplo práctico: Adaptar GPT-4 para análisis financiero requiere solo 50MB de parámetros adicionales vs 1.7TB del modelo completo.

Analogía: Como añadir "apps especializadas" a tu smartphone sin reemplazar el sistema operativo completo.

⚙️ **Hiperparámetros: Los controles maestros del Fine-Tuning**

Los hiperparámetros son configuraciones que controlan cómo aprende el modelo 🧠, determinando la velocidad, intensidad y estabilidad del entrenamiento.

🎯 **Hiperparámetros críticos**

- **Learning rate:** Velocidad de aprendizaje (0.00001 - 0.01)
- **Batch size:** Muestras por actualización (4, 8, 16)
- **Epochs:** Pasadas completas por el dataset (1-10)
- **r (rank):** Dimensión matrices LoRA (4, 8, 16)
- **alpha:** Factor de escalado LoRA (16, 32, 64)

🎨 **Analogía del chef:**

Como ajustar fuego (learning rate), tiempo de cocción (epochs) y cantidad de ingredientes (batch size) para el plato perfecto.

⚠️ **Impacto real:** Learning rate muy alto = modelo "olvida" conocimiento base
Learning rate muy bajo = entrenamiento infinito sin mejoras

Implementación práctica: Fine-Tuning con PEFT

Setup básico con LoRA


```
from transformers import AutoModelForCausalLM
from peft import LoraConfig, get_peft_model

# Base model
model = AutoModelForCausalLM.from_pretrained(
    "microsoft/DialoGPT-medium"
)

# LoRA configuration
lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=["query", "value"]
)

# Apply PEFT
model = get_peft_model(model, lora_config)
```

RAG: Conectando LLMs con datos empresariales

Retrieval Augmented Generation permite a los LLMs acceder a bases de datos corporativas , combinando generación de texto con búsqueda en tiempo real para análisis precisos con información actualizada.

Caso de uso crítico: Un modelo RAG puede consultar la base de datos de ventas de los últimos 30 días para generar insights actualizados, mientras mantiene la capacidad analítica del LLM base.

Este enfoque resuelve el "data drift" en modelos analíticos al anclar respuestas en fuentes verificables y actualizadas, similar a cómo un analista consulta dashboards antes de emitir recomendaciones.



Proceso general RAG: De documentos a respuestas



1. Preparación de datos

- **Chunking:** División en fragmentos
- **Tokenización:** Conversión texto a tokens
- **Embeddings:** Vectorización semántica
- **Indexación:** BD vectorial



2. Recuperación

- **Query embedding:** Vectorizar pregunta usuario
- **Similarity search:** Buscar chunks relevantes
- **Ranking:** Ordenar por relevancia



3. Generación aumentada

- **Context assembly:** Combinar chunks recuperados
- **Prompt construction:** Crear prompt enriquecido
- **LLM generation:** Generar respuesta contextualizada
- **Source citation:** Referencias verificables

Flujo completo: Documentos → Chunks → Embeddings → Vector DB → Query → Retrieve → Generate → Response

RAG completo: Pipeline en 3 pasos

1. Preparación e indexación

```
from langchain.vectorstores import Chroma
from langchain.embeddings import OpenAIEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter

# Dividir documentos en chunks
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,      # Tamaño de cada fragmento
    chunk_overlap=200     # Solapamiento para mantener contexto
)
chunks = text_splitter.split_documents(documents)

# Crear vector store
vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=OpenAIEmbeddings(),
    persist_directory="./vectordb" # Guardar en disco
)
```

RAG completo: Pipeline en 3 pasos

2. Recuperación y generación

```
from langchain.chains import RetrievalQA
from langchain.llms import OpenAI

# Setup retriever - busca los 5 chunks más similares
retriever = vectorstore.as_retriever(search_kwargs={"k": 5})

# Create QA chain - combina LLM + retriever
qa_chain = RetrievalQA.from_chain_type(
    llm=OpenAI(temperature=0.2),      # Respuestas consistentes
    retriever=retriever,
    return_source_documents=True      # Incluir fuentes
)

# Execute query
response = qa_chain({"query": "¿Cuáles son las métricas de Q3?"})
```

RAG completo: Pipeline en 3 pasos

3. Hiperparámetros clave:

- **chunk_size**: 1000 tokens (balance contexto/precisión)
- **chunk_overlap**: 200 tokens (continuidad semántica)
- **k**: 5 fragmentos más similares enviar al LLM
- **temperature**: 0.2 (respuestas consistentes)

Flujo de datos:

1. Documentos → Chunks → Embeddings
2. Query → Vector search → Top-k chunks
3. LLM + Contexto → Respuesta + Fuentes



Arquitectura RAG para bases de datos empresariales



Componentes especializados

- Database Connectors: SQL, NoSQL, APIs
- Semantic Layer: Embeddings de esquemas y metadatos
- Vector Search: Búsqueda híbrida (semántica + keyword)
- Query Translator: NL → SQL → Insights

Flujo híbrido: "¿Producto más vendido Q3?" → Embeddings de esquemas → Identifica tablas → Genera SQL → Ejecuta → Contexto vectorial → Respuesta enriquecida



RAG paso a paso con bases de datos

1. Indexación de metadatos: - Extracción esquemas BD (tablas, columnas, relaciones) - Generación embeddings para metadatos y documentación - Indexación semántica de estructuras de datos

2. Procesamiento inteligente

- **Query understanding:** NLP de intención
- **Schema mapping:** Vector search en metadatos
- **SQL generation:** LangChain SQL Agent
- **Hybrid retrieval:** Vector + keyword search

Ejemplo verificado: "Analiza churn por segmento" → Vector search: "churn" + "customer" → Mapea: customers, transactions → SQL Agent: window functions → Interpretación contextual

Sistemas híbridos: Lo mejor de ambos mundos

Los sistemas de producción modernos combinan Fine-tuning y RAG para maximizar precisión y eficiencia .

Arquitectura híbrida típica

1. **Modelo base fine-tuneado** con metodologías del dominio
2. **RAG layer** para acceso a datos actualizados
3. **Router inteligente** que decide qué información usar
4. **Validation layer** que verifica consistencia de resultados

Resultado: Sistema que "piensa como analista experto" (fine-tuning) pero con acceso a información actualizada (RAG).

Stack tecnológico recomendado

Para Fine-Tuning

- Hugging Face Transformers: Framework base
- PEFT Library: Técnicas eficientes
- Weights & Biases: Tracking experimentos
- Modal/RunPod: GPUs cloud

Para RAG

- LangChain: Orchestration framework
- ChromaDB/Pinecone: Vector databases
- Sentence-Transformers: Embeddings
- SQLAlchemy: Database connectivity

Pipeline de desarrollo típico: 1. Prototipo con OpenAI API (validar concepto) 2. Fine-tune modelo open-source (reducir costos) 3. Implementar RAG con datos propios (especialización) 4. Deploy con FastAPI + Docker (producción)

Consideraciones para implementación

Desafíos técnicos

- **Data quality:** Garbage in, garbage out
- **Latencia:** RAG añade 200-500ms por consulta
- **Costos:** Embeddings + vector search + LLM inference
- **Consistencia:** Resultados reproducibles

Mejores prácticas

- **Eval datasets:** Métricas específicas del dominio
- **A/B testing:** Comparar vs baselines humanos
- **Monitoring:** Tracking performance en producción
- **Feedback loops:** Mejora continua con uso real

Regla de oro: Comenzar simple (API calls), validar valor agregado, luego optimizar para costos y latencia.

ROI y métricas de éxito

Métricas técnicas vs métricas de negocio:

- **Técnicas:** BLEU score, perplexity, latencia
- **Negocio:** Tiempo analista ahorrado, precisión decisiones, revenue impact

El futuro de LLMs en ciencia de datos

Los modelos especializados serán la norma, no la excepción 🧠. Veremos agentes autónomos que combinan fine-tuning, RAG, y herramientas especializadas para análisis end-to-end.


Tendencias emergentes

- **Multimodal RAG:** Texto + gráficos + tablas
- **Federated learning:** Fine-tuning distribuido
- **AutoML integration:** Optimización automática pipelines
- **Causal reasoning:** Más allá de correlaciones

Predicción 2024-2025: Cada empresa Fortune 500 tendrá su "Data Scientist AI" personalizado, combinando conocimiento institucional con capacidades analíticas avanzadas.

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](#)

Herramientas clave:

- [Hugging Face](#)
- [LangChain](#)
- [OpenAI API](#)