

Introducción al Curso

LLMs en Programación: De la Generación de Código a la Solución de Problemas

Objetivo General

Explorar el potencial de los Modelos de Lenguaje (LLMs) para asistir, automatizar y mejorar tareas de programación.

Organización de las clases

- ① Fundamentos de los LLMs.
- ② Prompt Engineering para Programación. Contexto Engineering
- ③ Herramientas y Plataformas Basadas en LLMs (Local vs Nube)
- ④ Ética y Buenas Prácticas
- ⑤ Taller Final Práctico

LLMs en Programación

De la generación de código
a la solución de problemas

Clase 1: Fundamentos técnicos de los LLMs

Christian Luna Escudero

Grupo de investigación KDIS
Instituto de investigación DaSCI.
Universidad de Córdoba

15 de septiembre de 2025

KDIS Lab



DaSCI

¿Qué aprenderéis hoy?

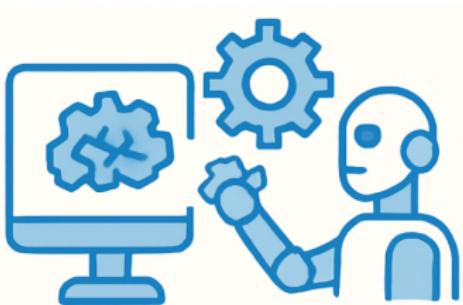
- Comprender qué son los **Grandes Modelos de Lenguaje** (LLMs) y en qué se diferencian de otros enfoques de IA
- Conocer la evolución reciente de los LLMs y su aplicación al desarrollo de software.
- Identificar las principales áreas de utilización.
- Reconocer beneficios y desafíos iniciales asociados a usar LLMs en programación.

¿Qué es la IA?

- ¿Dónde habéis visto inteligencia artificial últimamente?
 - ¿Qué cosas creéis que puede hacer?
 - ¿Y qué cosas aún no puede?

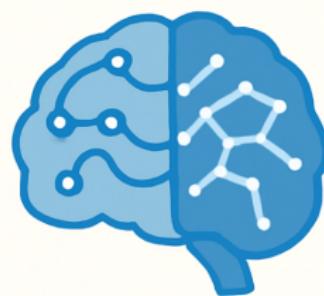
La Inteligencia Artificial es una disciplina con un doble objetivo:

Tecnológico



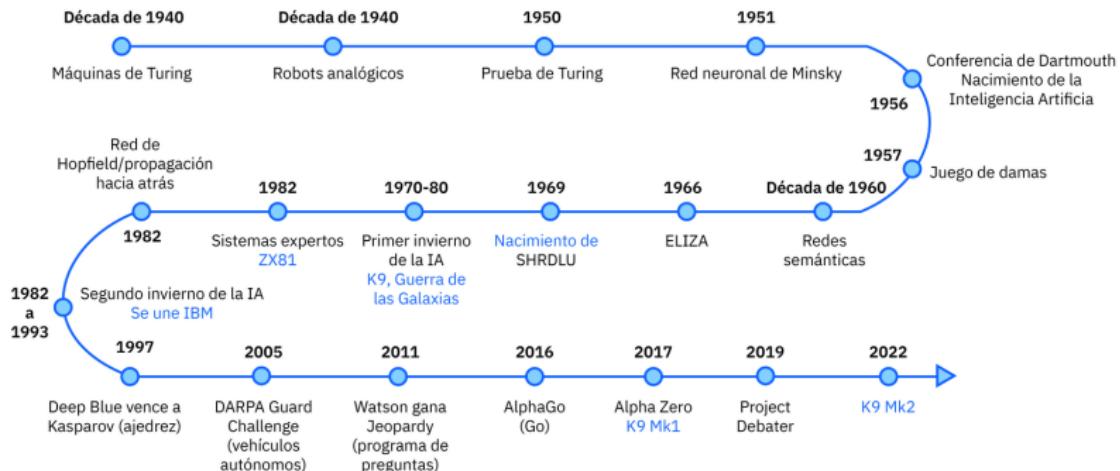
Desarrollar programas y/o dispositivos ⇒ capaces de realizar acciones propias de los seres inteligentes.

Científico



Desarrollar **teorías** que permitan **comprender** mejor los fundamentos del **comportamiento** inteligente y su **funcionamiento**.

Evolución de la IA



¿Cómo ha llegado la IA hasta nuestro día a día?

- Generación automática de texto (emails, resúmenes, código).
- Apoyo en tareas creativas (diseño, escritura, programación).
- Asistentes personales y chatbots.
- Mejora de productividad en desarrollo software.
- Tareas de automatización mediante agentes.



¿Se os ocurre algún otro ámbito donde la IA ya esté afectando nuestro trabajo o vida?

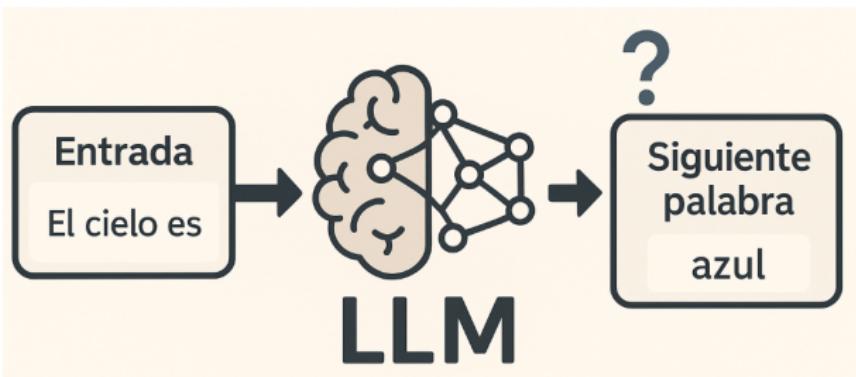
- Diagnóstico médico asistido
- Reconocimiento facial en seguridad
- Detección de fraudes financieros
- Recomendaciones en redes sociales
- Automatización del trabajo en fábricas

¿Qué es un LLM y por qué son importantes hoy en día?

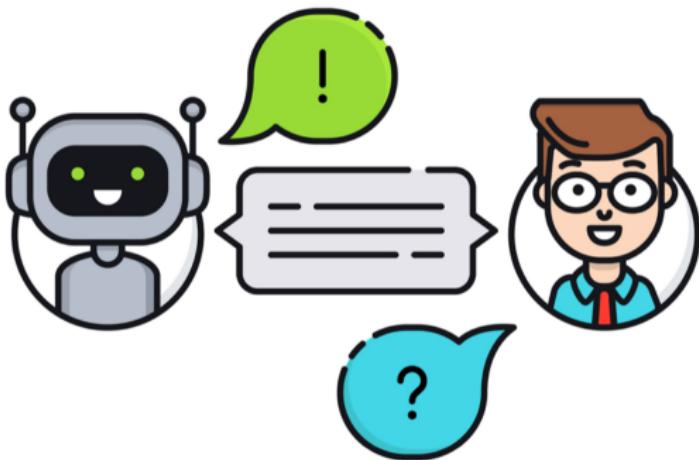
Definición

Un LLM es un modelo de IA entrenado con grandes volúmenes de texto (o código) que aprende a predecir la siguiente palabra en una secuencia, capturando patrones complejos de lenguaje.

La intuición detrás de un LLM



Formas de comunicarnos



- Comunicación Humano-Máquina
 - Procesamiento de Información
 - Innovación Tecnológica
 - Agentes autónomos

Aplicaciones de los LLM



Historia de los modelos de lenguaje

- **Décadas 90–2000:** Modelos estadísticos (*n-gramas*, conteo de frecuencias).
- **2000s:** Redes neuronales recurrentes (RNN), luego LSTM y GRU.
- **2014–2017:** Modelos seq2seq y encoder-decoder (p.ej., traducción automática).
- **2017:** Nace el **Transformer** con “Attention is All You Need”.

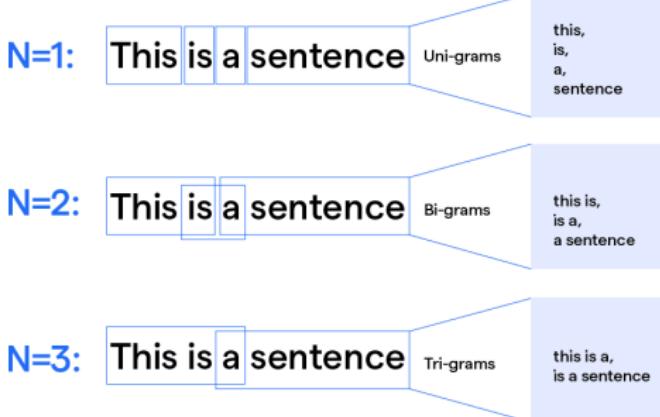
El Transformer marcó un antes y un después: paralelización, mejor contexto y escalabilidad.

Modelos Estadísticos y N-gramas

¿Cómo predicen la próxima?

Basándose en las probabilidades conociendo las N palabras anteriores.

N-Gram



Ejemplos prácticos



Con un clic, QuillBot escaneará tu texto y te alertará sobre cualquier error ortográfica, gramatical, de puntuación, palabra mal empleada y más.

Todo 1 Gramática 2 Sugerencias de oraciones 1

escaneará Falta de concordancia Aceptar

ortográfico Falta de concordancia Aceptar

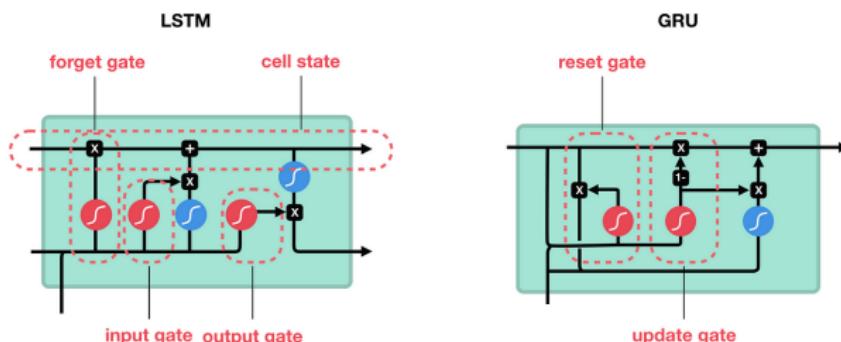
Introducción a las redes neuronales recurrentes

- Procesan secuencias paso a paso, manteniendo un estado de memoria.
- Permiten modelar dependencias temporales (texto, audio, series temporales).
- Limitación: difícil capturar relaciones largas → **problema del gradiente**.
- Soluciones: LSTM y GRU (mecanismos de memoria y olvido).

Las RNN fueron el primer paso hacia modelos que “recuerdan” lo anterior.

LSTM y GRU: superando las limitaciones de las RNNs

- **LSTM (Long Short-Term Memory):**
 - Introduce **puertas** que controlan qué información se guarda u olvida.
 - Mejora el manejo de **dependencias a largo plazo**.
- **GRU (Gated Recurrent Unit):**
 - Arquitectura más simple, con menos parámetros.
 - Similar rendimiento a LSTM, pero más eficiente.



sigmoid



tanh



pointwise multiplication



pointwise addition



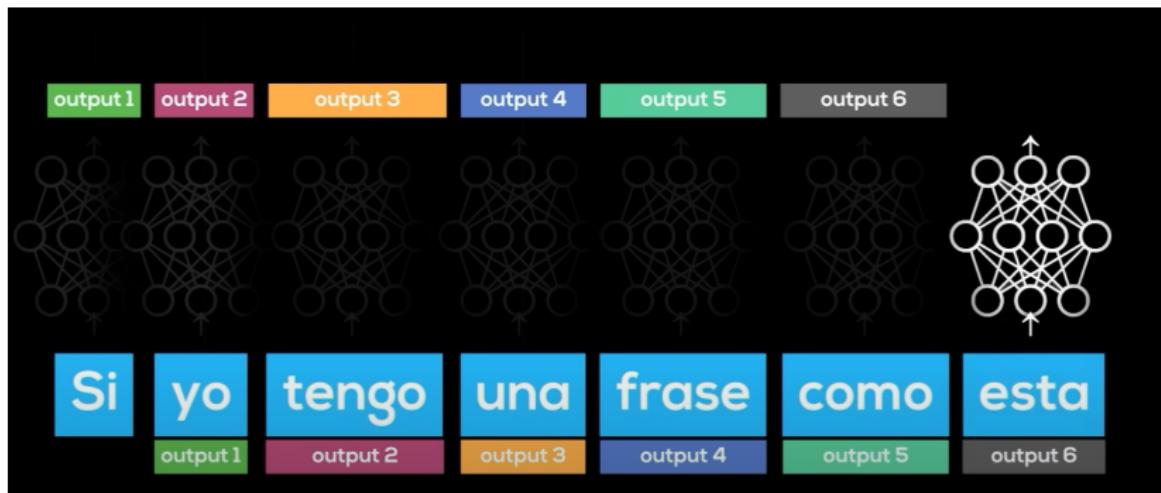
vector concatenation

¿Cómo aprovechamos estas redes en tareas reales?

- Traducir frases entre idiomas
- Resumir textos largos
- Generar respuestas automáticas

Para eso, usamos arquitecturas encoder-decoder, basadas inicialmente en LSTM/GRU.

¿Cómo procesan la información?



Modelos seq2seq y encoder-decoder

- Propuestos para tareas como traducción automática.
- **Encoder**: procesa toda la secuencia de entrada → vector de contexto.
- **Decoder**: genera la salida paso a paso a partir de ese vector.
- Limitación: el vector fijo no escala bien a secuencias largas → atención.

Fundamento de muchos modelos de traducción, resumen y pregunta-respuesta.

¿Cómo funcionan?



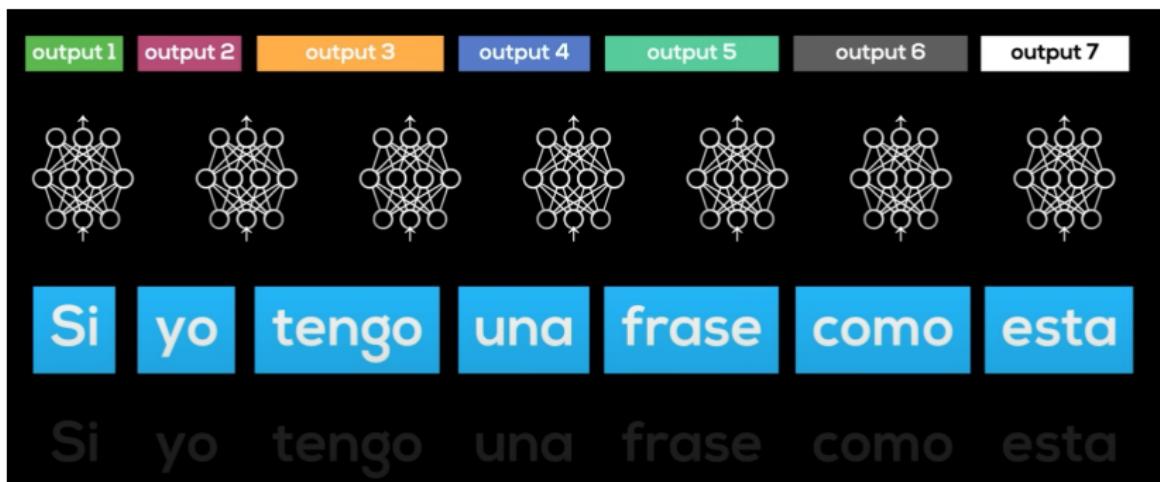
Arquitectura Transformer: ¿cómo funciona?

Componentes principales

- **Tokenización:** convierte texto/código en secuencias de tokens.
- **Embeddings:** representa cada token como un vector numérico.
- **Atención:** decide qué partes del contexto son relevantes.
- **Capas apiladas:** refinan la comprensión en múltiples niveles.

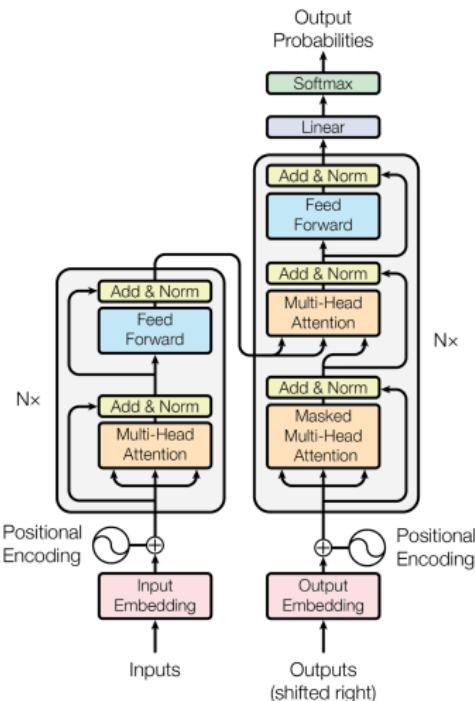
“No leen línea por línea: lo procesan todo a la vez con múltiples focos de atención.”

¿Cómo lo procesan frente a las redes neuronales?



Auto-atención: query, key y value

- Cada token se convierte en:
 - **Query** (qué busca),
 - **Key** (a qué representa),
 - **Value** (la información que aporta).
- La atención se calcula como similitud entre Query y Key.
- El resultado se usa para ponderar los Values.



Esquema básico de auto-atención.

¿Por qué es tan eficaz el Transformer?

- **Paralelización:** no necesita procesar secuencialmente, lo que acelera el entrenamiento.
- **Atención múltiple:** usa múltiples “cabezas” de atención para captar relaciones distintas.
- **Representaciones profundas:** cada capa refina el entendimiento del contexto.
- **Escalabilidad:** permite construir modelos con cientos de capas y miles de millones de parámetros.

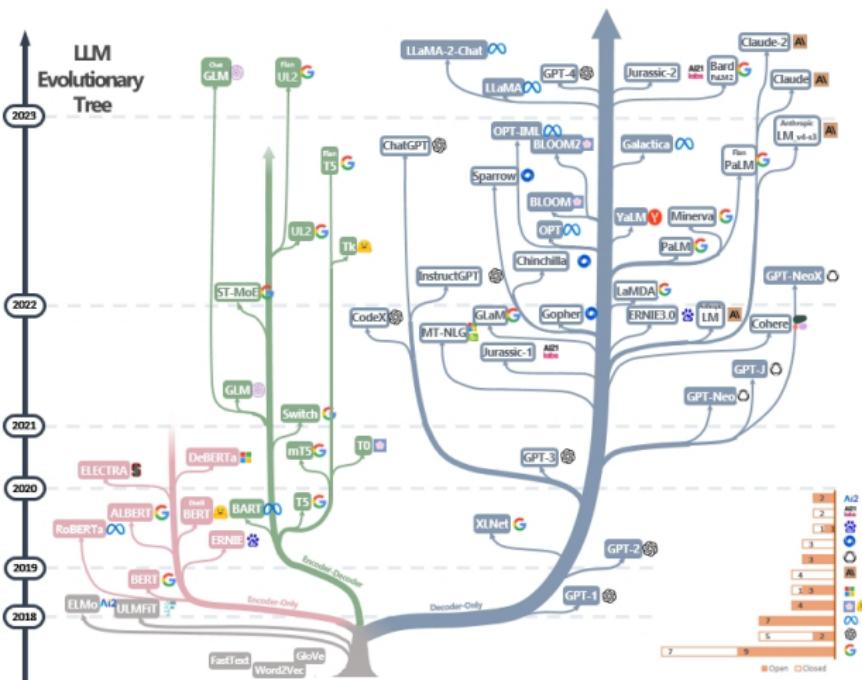
La arquitectura Transformer es la base de todos los LLMs actuales.

Video demo

Evolución reciente de los LLMs

- **GPT-1 (2018)**: Generación autoregresiva con corpus general.
- **GPT-2 (2019)**: Coherencia a largo plazo y primeras capacidades emergentes.
- **GPT-3 (2020)**: 175B parámetros. Capaz de few-shot y tareas complejas.
- **Codex (2021)**: Entrenado con código — base de GitHub Copilot.
- **GPT-4, LLaMA, Claude (2023–2024)**: modelos más robustos, multimodales y con más contexto.

Escalar el modelo permite que aparezcan capacidades no vistas en versiones pequeñas.



Modelos más recientes (2024–2025)

- **GPT-4o (2024)**: Texto, voz y visión con respuestas más rápidas y precisas.
- **o1/o3/o4-mini (2024–2025)**: Modelos optimizados para razonamiento, ciencia y programación.
- **DeepSeek-V3 y R1**: Open-source con Mixture-of-Experts, destaca en codificación y razonamiento.
- **Gemini 1.5/2.5 Pro**: Modelos multimodales de Google con comprensión de contexto y capacidades avanzadas.

Escalado de modelos: ¿más grande = más inteligente?

- Aumentar parámetros mejora la capacidad de representación.
- Más datos y capas → más comprensión contextual.
- Aparecen **capacidades emergentes**: tareas que el modelo no fue entrenado explícitamente para hacer.
- Ejemplos:
 - Razonamiento paso a paso.
 - Traducción multilingüe.
 - Resolución de problemas de programación.

"A partir de cierto umbral, los modelos sorprenden incluso a sus creadores."

Ejemplos de capacidades emergentes

- **Chain-of-thought**: genera razonamientos intermedios antes de la respuesta final.
- **In-context learning**: aprende de ejemplos en el prompt, sin reentrenamiento.
- **Multi-hop QA**: responde preguntas que requieren combinar varias fuentes de información.
- **Programación multitarea**: completa código, lo explica y lo adapta en un mismo turno.
- **Uso de herramientas externas**: calcula, llama a APIs o navega por internet con ayuda externa.

Estas habilidades surgen cuando los modelos alcanzan suficiente escala y contexto.

Relación entre tamaño del modelo y capacidades

Parámetros	Tipo de tareas	Capacidades observadas
$< 100M$	Básicas	Completado de frases, respuesta literal simple
$1B$	Intermedias	Traducción, clasificación de sentimiento, resumen corto
$10B$	Avanzadas	Few-shot learning, razonamiento simple, QA básica
$100B+$	Emergentes	Chain-of-thought, multitarea, resolución de código
$> 500B$	Complejas	Planificación, pensamiento multihop, uso de herramientas externas

Las capacidades no crecen linealmente: aparecen saltos cualitativos.

¿Por qué los LLMs pueden escribir código?

- El código fuente es texto estructurado, con reglas sintácticas claras.
- Existen millones de ejemplos en repositorios públicos (GitHub, StackOverflow...).
- Las dependencias a largo plazo (nombres de variables, estructuras) son ideales para modelos con atención.
- El lenguaje de programación es más formal que el lenguaje natural: facilita el aprendizaje estadístico.

Los LLMs aprenden a programar de forma implícita, sin reglas fijas, solo por exposición masiva.

Aplicaciones de los LLMs en desarrollo de software

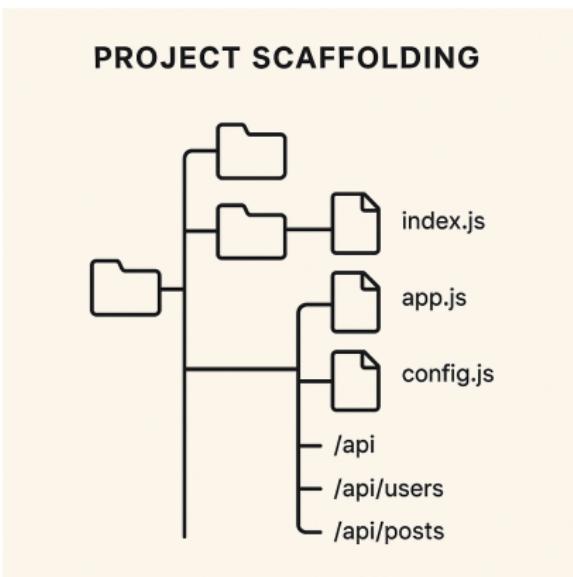
- **Autocompletado inteligente** (p. ej. GitHub Copilot, Amazon CodeWhisperer).
- **Generación de funciones** a partir de descripciones en lenguaje natural.
- **Explicación de código:** comenta, resume y documenta fragmentos complejos.
- **Refactorización y estilización:** mejora legibilidad, renombra variables, reestructura.
- **Generación de pruebas unitarias** y casos de test.
- **Depuración asistida:** sugiere posibles errores o soluciones.

¿Cómo afecta esto a tu rol como programador?



LLMs para Desarrolladores Backend

- **Generación de scaffolding y boilerplate** para acelerar el inicio de nuevos proyectos.
- **Refactorización de código legado** con sugerencias de mejores prácticas.
- **Pruebas unitarias y mocks automáticos** generados a partir del código fuente.



LLMs para Desarrolladores Frontend

- **Sugerencia de componentes** en frameworks como React o Vue.
- **Refactor de CSS**, validación de props y mejora de accesibilidad.
- **Traducción de prototipos UI** a código funcional (HTML/CSS/JS).



LLMs para Ingeniería de Datos

- **Generación de scripts de limpieza y pipelines** en Python o SQL.
- **Transformaciones complejas** en Pandas o PySpark explicadas paso a paso.
- **Explicación de consultas SQL complejas** o creación de consultas optimizadas.

LLMs para Investigadores y ML Engineers

- **Generación de código de experimentación** usando librerías como scikit-learn o Keras.
- **Análisis y visualización de resultados** con matplotlib, seaborn o pandas.
- **Creación de notebooks documentados** para reproducibilidad y colaboración.

The screenshot shows a Colab AI interface with the following components:

- Code Generation Panel:** Shows a snippet of Python code for sentiment analysis using scikit-learn. It includes imports for pandas and scikit-learn, reading a CSV file, handling missing values, splitting the data into training, validation, and test sets, and printing the sizes of each set.
- Task Description:** A sidebar titled "Colab AI X" contains the text: "Do the following steps in python step by step".
- Step Instructions:** A numbered list of steps:
 1. read a csv file from /content/drive/Shareddrives/test/Sentiment/DR00_Dataset.csv
 2. check for any missing values in columns sentiment of df. Print it and remove them
 3. do the same for 'review' column of df
 4. Split the df into three sets train, valid, and test splits with ratio 8/1/1 using 'review' as independent variable and 'sentiment' as target variable. Use random_state to 2023
- Code Execution:** A panel titled "Colab AI" shows the generated code being run in a notebook cell. The output shows the execution of the code and the resulting data splits.
- Feedback and Rating:** At the bottom right, there are buttons for "Rate this answer" and a rating scale from 1 to 5.

Limitaciones actuales en programación asistida

- **No comprenden el propósito del proyecto:** solo predicen tokens.
- **Generan código plausible, no siempre correcto:** pueden alucinar funciones, imports, clases...
- **Dificultad con código muy extenso** (dependencias complejas, múltiples ficheros).
- **Sensibles al contexto:** prompts ambiguos → resultados ambiguos.
- **Riesgos de seguridad:** fugas de datos, código inseguro, licencias dudosas.

Los LLMs no son ingenieros de software: son asistentes muy potentes pero con límites.

Impacto en la productividad del desarrollo

- Estudios reportan mejoras de hasta **55 % en velocidad de desarrollo** (GitHub Next, 2024).
- Facilita tareas repetitivas: boilerplate, documentación, tests.
- Ayuda en el aprendizaje continuo: explica código, sugiere buenas prácticas.
- **Reduce barreras de entrada** para juniors o en nuevos lenguajes.
- Mejora el tiempo de respuesta en depuración y exploración de código legado.

“No sustituye, potencia: los LLMs aumentan la productividad del desarrollador humano.”

Riesgos y consideraciones éticas

- **Privacidad:** riesgo de generar o reutilizar datos sensibles del entrenamiento.
- **Código con licencia incompatible:** puede copiar código con copyright sin aviso.
- **Sesgos:** reproduce estereotipos presentes en el corpus (p.ej. sobre género, raza, localización).
- **Hallucinations:** genera código plausible pero incorrecto o inseguro.
- **Dependencia:** pérdida de comprensión si delegamos demasiado.

Buenas prácticas en programación asistida

- Verificar siempre el código generado: pruebas, revisión, seguridad.
- Especificar bien el prompt: objetivo, restricciones, formato.
- Pedir explicaciones y ejemplos, no solo código.
- Usar como asistente, no como sustituto.
- Documentar y versionar el uso de sugerencias automáticas.

“El buen uso de los LLMs no es técnico: es crítico.”

¿Cómo se evalúan los LLMs en tareas de programación?

HumanEval (OpenAI) - Link

Conjunto de 164 ejercicios de Python diseñados para medir la capacidad de un modelo para:

- Generar funciones correctas a partir de una descripción textual.
- Superar pruebas unitarias predefinidas.

Otras métricas y benchmarks

- **Pass@k**: ¿Qué probabilidad hay de generar una solución válida entre las k salidas?
- **MBPP**: Pequeños problemas de Python con tests.
- **Bugs-BUSTED**: Evaluación de corrección de código con errores intencionales.
- **SWE-Bench**: Corrección de bugs reales en repositorios open-source.

Evaluar LLMs va más allá de que “funcione”: debe generalizar, razonar y pasar tests.

¿Cómo se evalúan los LLMs en tareas de programación?

HumanEval (OpenAI) - Link

Conjunto de 164 ejercicios de Python diseñados para medir la capacidad de un modelo para:

- Generar funciones correctas a partir de una descripción textual.
- Superar pruebas unitarias predefinidas.

Otras métricas y benchmarks

- **Pass@k**: ¿Qué probabilidad hay de generar una solución válida entre las k salidas?
- **MBPP**: Pequeños problemas de Python con tests.
- **Bugs-BUSTED**: Evaluación de corrección de código con errores intencionales.
- **SWE-Bench**: Corrección de bugs reales en repositorios open-source.

Evaluar LLMs va más allá de que “funcione”: debe generalizar, razonar y pasar tests.

¿Sustituirán los LLMs a los programadores?

Dinámica

- Pregunta inicial: **¿Te sientes amenazado o potenciado por esta tecnología?**
- Breve exposición de datos:
 - +55 % productividad (GitHub Next, 2024)
 - 46 % de los devs temen por su empleo (Stack Overflow Survey, 2024)

¿Sustituirán los LLMs a los programadores?

Dinámica

- Pregunta inicial: **¿Te sientes amenazado o potenciado por esta tecnología?**
- Breve exposición de datos:
 - +55 % productividad (GitHub Next, 2024)
 - 46 % de los devs temen por su empleo (Stack Overflow Survey, 2024)

Demostración práctica con ChatGPT o Copilot

Prompt base:

Escribe una función en Python que reciba una lista y devuelva los elementos únicos ordenados.

- Mostrar cómo responde con un prompt vago.
- Refinar: añadir ejemplos, restricciones de complejidad, test.
- Comparar resultados y comentar cómo se mejora la respuesta.

Ejercicio práctico: Escribe tu primer prompt de programación

Instrucción

- Escribe un prompt para que el modelo genere una función que verifique si un número es primo.
- El código debe estar comentado y manejar entradas inválidas.
- Usa estilo conversacional, como si hablaras con un asistente experto.

Compararemos en grupo y analizaremos cómo mejorar los resultados.

Resumen de conceptos clave

- Los LLMs predicen el siguiente token en una secuencia: esa es su tarea central.
- La arquitectura Transformer permite procesar texto en paralelo, con mecanismos de atención.
- El contexto completo y el tamaño del modelo son claves para su rendimiento.
- A mayor escala, surgen capacidades emergentes no programadas explícitamente.
- Estos modelos no “entienden”, pero modelan patrones con enorme precisión.

Próxima sesión

Prompt Engineering aplicado a programación:

- Fundamentos: zero-shot, one-shot, few-shot.
- Diseño de prompts efectivos.
- Casos prácticos sobre tareas comunes de programación.

Cierre de la Clase 1

¿Qué te llevas hoy?

- Sabes qué es un LLM y por qué funciona.
- Entiendes su arquitectura y evolución.
- Reconoces su impacto —y sus límites— en programación.

La próxima clase vamos a usar lo aprendido para diseñar prompts potentes.

Lectura recomendada