

# LLMs en Programación

## De la generación de código a la solución de problemas

Clase 3: Herramientas y Plataformas Basadas en LLMs

Christian Luna Escudero

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

22 de septiembre de 2025

KDISLab



DaSCI

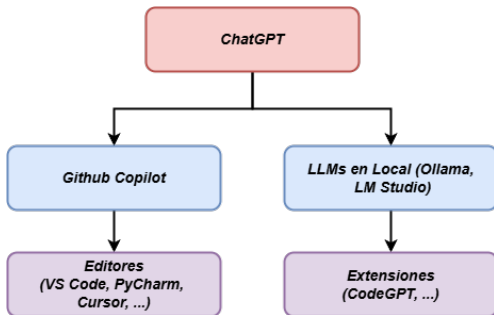
## ¿Qué aprenderás hoy?

- Cómo utilizar GitHub Copilot y ChatGPT para programar más rápido y mejor.
- Qué herramientas existen para correr modelos LLM en local.
- Cuándo conviene usar la nube y cuándo modelos locales.
- Actividades prácticas para comparar herramientas.

# Herramientas de Prompt Engineering



## Ecosistema de Herramientas con LLMs



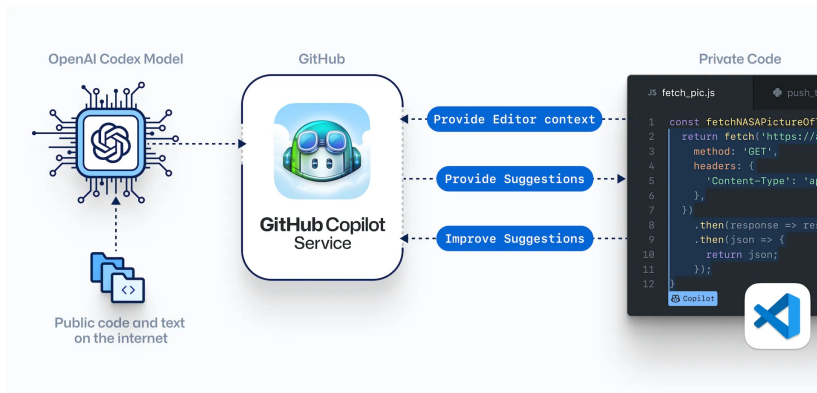
- Las herramientas se complementan entre sí.
- IDE + asistente es la combinación base.
- Modelos locales y en nube pueden alternarse según contexto.

## ¿Qué es GitHub Copilot?



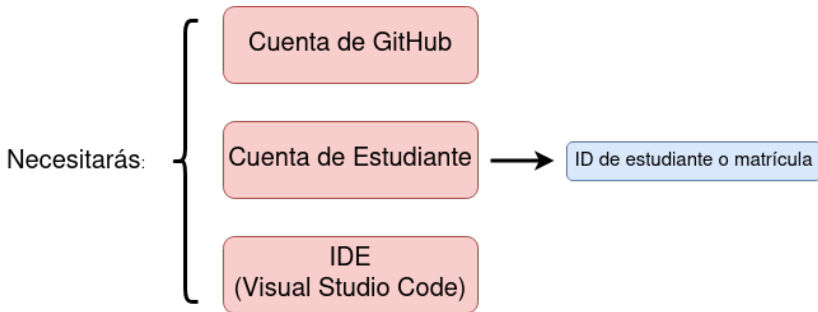
- Desarrollado por Github en colaboración con OpenAI.
- Herramienta de completado de código basada en LLMs.
- Entrenada sobre millones de repositorios públicos.
- Funciona como asistente en editores como VSCode o JetBrains.

# Github Copilot: ¿Cómo funciona?



# Instalación y Configuración

- 1 Crear cuenta en GitHub.
- 2 Activar la suscripción a Copilot (gratuita para estudiantes y docentes).
- 3 Instalar la extensión en VSCode o entorno compatible.
- 4 Iniciar sesión con GitHub.



# GitHub Copilot: Beneficios y Limitaciones

## Ventajas

- **Usabilidad:** integración fluida en el IDE.
- **Productividad:** aceleración en tareas repetitivas.
- **Aprendizaje:** sugiere código idiomático y patrones comunes.

## Limitaciones

- **Usabilidad:** contexto limitado a la ventana actual.
- **Productividad:** puede inducir errores sutiles o malas prácticas.
- **Aprendizaje:** no explica decisiones ni alternativas.



## Casos de Uso - Autocompletado inteligente de funciones

- Copilot sugiere automáticamente el cuerpo de funciones a partir del nombre y los argumentos.
- Aprende del contexto del archivo actual y del proyecto.
- Útil para funciones comunes como cálculos, búsquedas, validaciones, etc.

```
# Extrae nombres propios, fechas y organizaciones de un texto usando spacy
def extraer_entidades(texto):
    import spacy

    nlp = spacy.Load("es_core_news_sm")
    doc = nlp(texto)

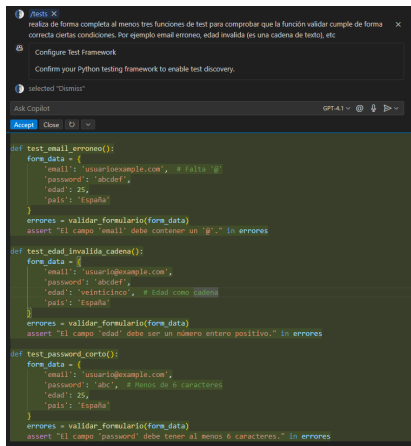
    entidades = {
        "nombres_propios": [],
        "fechas": [],
        "organizaciones": []
    }

    for ent in doc.ents:
        if ent.Label_ == "PER":
            entidades["nombres_propios"].append(ent.text)
        elif ent.Label_ == "DATE":
            entidades["fechas"].append(ent.text)
        elif ent.Label_ == "ORG":
            entidades["organizaciones"].append(ent.text)

    return entidades
```

## Casos de Uso - Generación de pruebas unitarias

- Al escribir 'test\_' Copilot propone automáticamente pruebas unitarias para funciones existentes.
- Reconoce funciones y estructuras previas del archivo.
- Permite acelerar la cobertura de testing.



The screenshot shows the GitHub Copilot interface with a chat window at the top and a code editor below. The chat window contains the following text:

```
/tests X
realiza de forma completa al menos tres funciones de test para comprobar que la función validar cumple de forma
correcta ciertas condiciones. Por ejemplo email erroneo, edad invalida (es una cadena de texto), etc

Configure Test Framework
Confirm your Python testing framework to enable test discovery.

selected "Dismis"

Ask Copilot
GPT-4.1 @ 100% 100% 100%
```

The code editor shows the following Python code:

```
def test_email_erroneo():
    form_data = {
        'email': 'usuarioexample.com', # Falta '@'
        'password': 'abcdef',
        'edad': 25,
        'pais': 'España'
    }
    errores = validar_formulario(form_data)
    assert "El campo 'email' debe contener un '@'." in errores

def test_edad_invalida_cadena():
    form_data = {
        'email': 'usuarioexample.com',
        'password': 'abcdef',
        'edad': 'veinticinco', # Edad como cadena
        'pais': 'España'
    }
    errores = validar_formulario(form_data)
    assert "El campo 'edad' debe ser un número entero positivo." in errores

def test_password_corto():
    form_data = {
        'email': 'usuarioexample.com',
        'password': 'abc', # Menos de 6 caracteres
        'edad': 25,
        'pais': 'España'
    }
    errores = validar_formulario(form_data)
    assert "El campo 'password' debe tener al menos 6 caracteres." in errores
```

## Casos de Uso - Conversión de código entre lenguajes

- A partir de un comentario, Copilot puede traducir funciones entre lenguajes populares.
- Soporta Python, JavaScript, TypeScript, Java, C++, etc.
- Muy útil para programadores que migran proyectos o prueban nuevos entornos.

The screenshot shows a code editor with a Python function `obtenerDatos(url)` on the left. A Copilot chat window is open, showing a prompt: `// Convierte esta función a Python` followed by the Python code. The response shows the translated JavaScript code. The code is as follows:

```
1 import aiohttp

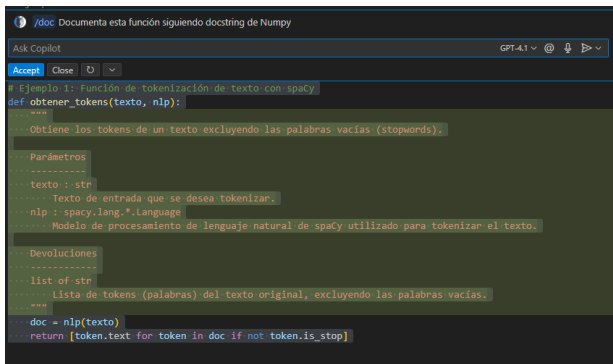
2 // Convierte esta función a Python
3 async function obtenerDatos(url) {
4   try {
5     const response = await fetch(url);
6     if (response.ok) {
7       throw new Error(Error: HTTP: ${response.status});
8     }
9     const data = await response.json();
10    return data;
11  } catch (error) {
12    console.error("Error al obtener los datos:", error);
13    return null;
14  }
15 }
```

The chat window has a dropdown menu with "Ask Copilot" and "GPT-4.1". Below the chat window, the translated JavaScript code is shown in the editor:

```
2 import async
3
4 async def obtener_datos(url):
5   try:
6     async with aiohttp.ClientSession() as session:
7       async with session.get(url) as response:
8         if response.status != 200:
9           raise Exception(f"Error HTTP: {response.status}")
10         data = await response.json()
11         return data
12   except Exception as error:
13     print("Error al obtener los datos:", error)
14   return None
```

## Casos de Uso - Documentación automática

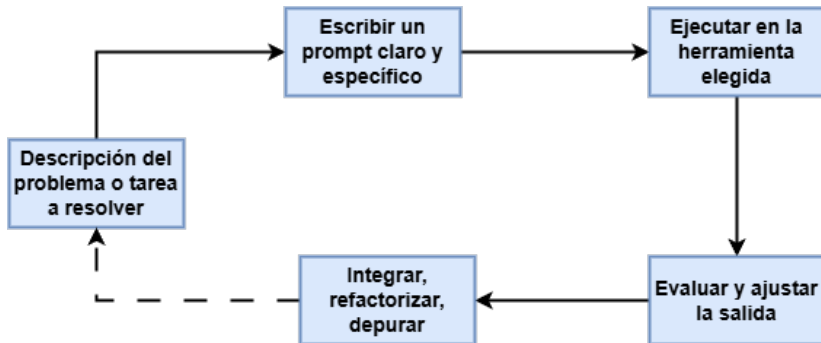
- Al comenzar una docstring Copilot propone automáticamente la descripción de la función.
- Puede sugerir parámetros, retornos y ejemplos si el código es claro.
- Fomenta buenas prácticas de documentación.



```

1 /doc Documenta esta función siguiendo docstring de Numpy
2
3 Ask Copilot GPT-4.1 @
4
5 Accept Close
6
7 # Ejemplo 1: Función de tokenización de texto con spaCy
8 def obtener_tokens(texto, nlp):
9     """
10     Obtiene los tokens de un texto excluyendo las palabras vacías (stopwords).
11
12     Parámetros
13     -----
14     texto : str
15         Texto de entrada que se desea tokenizar.
16     nlp : spacy.lang.*.Language
17         Modelo de procesamiento de lenguaje natural de spaCy utilizado para tokenizar el texto.
18
19     Devoluciones
20     -----
21     list of str
22         Lista de tokens (palabras) del texto original, excluyendo las palabras vacías.
23
24     """
25     doc = nlp(texto)
26     return [token.text for token in doc if not token.is_stop]
```

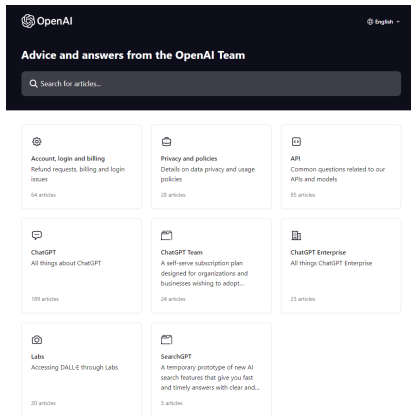
## Flujo de Trabajo con LLMs en Programación



- Este ciclo se repite: iterar mejora los resultados.
- La calidad del prompt afecta todo el proceso.

# ChatGPT para desarrolladores

- Desarrollado por OpenAI basado en la arquitectura GPT.
- Aplicaciones: Chatbots, asistentes virtuales, etc.
- Funcionalidad: Conversar, responder y generar contenido.



# ChatGPT: ¿Cómo funciona?

- Chat interactivo para depurar, generar y revisar código.
- Acceso vía chat web o API REST.
- Capacidad para recordar el contexto y seguir instrucciones complejas.



# OpenAI API

- Acceso mediante clave API personal.
- Permite integrar LLMs en tus propias aplicaciones (chatbots, autocompletado, análisis).
- Soporta llamadas a modelos como GPT-4, GPT-3.5, DALL-E y Whisper.
- Documentación: <https://platform.openai.com/docs>



# ChatGPT: Beneficios y Limitaciones

## Ventajas

- **Interacción:** fluido, flexible, conversación extendida.
- **Aplicaciones:** muy versátil, desde generación de código hasta planificación.
- **Coste/Acceso:** gratuito (limitado) o mediante suscripción mensual.

## Limitaciones

- **Interacción:** puede inventar o alucinar respuestas.
- **Aplicaciones:** menos control sobre el formato y precisión de salida.
- **Coste/Acceso:** requiere conexión y tiene límite de contexto.

## Casos de Uso - Explicación de código

- ChatGPT puede analizar bloques de código y explicarlos en lenguaje natural.
- Útil para entender código legado, depurar o aprender nuevas librerías.
- Puede generar analogías, resúmenes y simplificaciones.

```
Prompt: (Puedes explicarme paso a paso qué hace este código, qué técnicas usa y para qué sirve cada parte como si se lo explicaras a alguien que está empezando con análisis de datos en Python?)

'''python

import pandas as pd
import numpy as np

def preparar_dataset(path_csv):
    """
    Carga un CSV, limpia valores nulos, codifica columnas categóricas
    y escala las variables numéricas.
    """
    try:
        df = pd.read_csv(path_csv)
    except FileNotFoundError:
        print("Archivo no encontrado:", path_csv)
        return None

    # Eliminar filas con más del 50% de nulos
    df = df[df.isnull().mean(axis=1) < 0.5]

    # Reemplazar nulos en numéricas con la mediana
    num_cols = df.select_dtypes(include=np.number).columns
    for col in num_cols:
        df[col].fillna(df[col].median(), inplace=True)

    # Codificar columnas categóricas
    cat_cols = df.select_dtypes(include='object').columns
    df = pd.get_dummies(df, columns=cat_cols, drop_first=True)

    # Escalar columnas numéricas
    df[num_cols] = (df[num_cols] - df[num_cols].mean()) / df[num_cols].std()

    return df
'''
```

## Casos de Uso - Refactorización y mejora de código

- A partir de código funcional, ChatGPT puede sugerir versiones más limpias, legibles o eficientes.
- También puede adaptar código a estilos específicos (PEP8, Java idiomático...).
- Muy útil en revisiones o limpieza de proyectos.

```
Prompt: El siguiente código funciona, pero es un poco sucio y repetitivo. ¿Puedes refactorizarlo siguiendo buenas prácticas de Python (PEP8), hacerlo más legible y aplicar técnicas como comprensión de listas, separación de funciones o nombres más descriptivos si lo crees conveniente?
```

```
python
def procesar_lista(datos):
    resultado = []
    for i in range(len(datos)):
        if datos[i] % 2 == 0:
            cuadrado = datos[i] * datos[i]
            if cuadrado > 10:
                resultado.append(cuadrado)
    return resultado
```

## Casos de Uso - Diseño de arquitectura y patrones

- ChatGPT puede ayudar a esbozar arquitecturas de software según requisitos dados.
- Proporciona diagramas, sugerencias de patrones (MVC, factory, observer...).
- Útil para brainstorming técnico y planificación inicial.

### Prompt ejemplo

Quiero desarrollar una aplicación web para gestionar reservas de salas de reuniones en una empresa.

Debería tener:

- Registro/login de usuarios.
- Visualización de disponibilidad de salas.
- Gestión de reservas con notificaciones.
- Panel de administración para gestionar salas y usuarios.

¿Qué arquitectura me recomiendas para esto?

¿Puedes proponer un diseño basado en patrones conocidos (como MVC, repositorios, etc.) e incluir una explicación general del flujo de datos?

## Casos de Uso - Generación de scripts o configuraciones

- Ideal para generar rápidamente archivos Dockerfile, GitHub Actions, scripts de testeo, etc.
- ChatGPT entiende muchos lenguajes de configuración y automatización.
- Útil cuando no se recuerda la sintaxis exacta o se necesita un punto de partida rápido.

### Prompt de ejemplo

Quiero un workflow de GitHub Actions que:

- Se ejecute cuando haga push a la rama main.
- Instale las dependencias listadas en `requirements.txt`.
- Ejecute los tests con `pytest`.
- Use Python 3.10.

## ¿Por qué ejecutar modelos localmente?

- **Privacidad y control:** tus datos nunca salen de tu máquina.
- **Sin conexión necesaria:** útil en entornos aislados o con restricciones de red.
- **Latencia baja:** respuesta casi instantánea si el hardware es adecuado.
- **Flexibilidad:** puedes elegir el modelo exacto, tamaño, tokenizer, etc.
- **Coste cero:** sin suscripciones (una vez superada la barrera hardware).

*Ideal para desarrolladores avanzados, usuarios técnicos o entornos sensibles.*

# Opciones Populares

- **Ollama:**
  - CLI sencilla para descargar y ejecutar modelos (ej. `ollama run llama3`).
  - Modelos disponibles: llama2, llama3, gemma, codellama, mistral...
  - Soporta endpoints locales vía HTTP.
- **LM Studio:**
  - Interfaz gráfica para chatear con modelos como Mistral, LLaMA, etc.
  - Soporta modelos GGUF/GGML para ejecución en CPU o GPU.
  - Ideal para pruebas rápidas o uso interactivo sin terminal.
- **Text Generation WebUI:**
  - Plataforma web completa para gestionar, evaluar y visualizar modelos.
  - Requiere instalación más avanzada (gradio + Transformers).
  - Muy personalizable.

# Instalación típica con Ollama

- ❶ Descargar desde: <https://ollama.com>
- ❷ Instalar según tu sistema operativo.
- ❸ Abrir una terminal y ejecutar:
  - `ollama run llama3`
  - O probar otros modelos como `gemma`, `mistral`, `codellama`.
- ❹ Usar el endpoint en local:  
`http://localhost:11434/api/generate`
- ❺ Integrarlo con VSCode vía extensiones como CodeGPT o scripts propios.



## LM Studio: GUI para modelos locales

- Plataforma de escritorio disponible para Windows, macOS y Linux.
- Interfaz sencilla para:
  - Descargar modelos (.gguf)
  - Ejecutarlos con distintos backends (CPU, GPU)
  - Chatear con los modelos de forma interactiva
- Permite configurar servidores locales que pueden usarse como endpoints REST.
- No requiere uso de terminal, ideal para usuarios no técnicos.

Enlace: <https://lmstudio.ai>

## CodeGPT (Extensión para VSCode)

- Extensión gratuita y de código abierto.
- Permite conectar con:
  - API de OpenAI, Cohere, Hugging Face, Anthropic...
  - Endpoints locales (Ollama, LM Studio, etc.)
- Funcionalidades:
  - Explicación de código, generación, refactorización.
  - Conversación con modelos directamente desde el editor.
  - Configurable vía JSON.
- Muy útil para probar modelos locales sin salir del IDE.

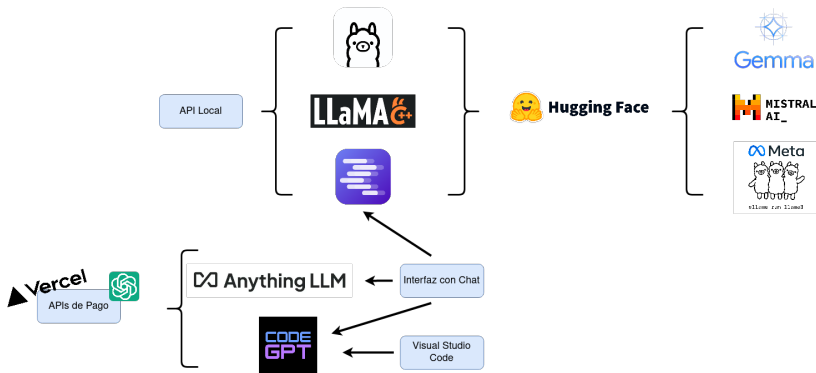
Repositorio: <https://github.com/CodeGPT-org/CodeGPT-vscode>

## LLMs en la Nube vs Locales

Aspecto	Nube (e.g., ChatGPT)	Local (e.g., Ollama)
Acceso	Requiere conexión	Funciona sin Internet
Privacidad	Datos en servidores externos	Datos en tu equipo
Latencia	Alta si hay carga o red lenta	Muy baja (local)
Modelos disponibles	GPT-4, Claude, Gemini...	LLaMA, Mistral, Phi-3, DeepSeek...
Interfaz	Web o API	Terminal, VS Code, WebUI
Consumo de recursos	Casi nulo	Necesita CPU/RAM/GPU

- La elección depende del caso: privacidad, potencia, conectividad.

# Flujo de Trabajo



## ¿Qué herramienta deberías usar?

Elige según tu caso de uso

¿Trabajas en proyectos privados o confidenciales?

→ LLM local (Ollama, LM Studio)

¿Quieres ejemplos, explicaciones o código bien comentado?

→ ChatGPT (Web o API)

¿Prefieres trabajar desde tu editor de código directamente?

→ GitHub Copilot + Copilot Chat

¿Necesitas alternar entre varios modelos fácilmente?

→ Ollama o TextGen WebUI con interfaz visual

¿Tienes pocos recursos y no puedes ejecutar modelos grandes?

→ Usar nube (ChatGPT, Gemini)

## ¿Qué herramienta deberías usar?

Elige según tu caso de uso

¿Trabajas en proyectos privados o confidenciales?

→ LLM local (Ollama, LM Studio)

¿Quieres ejemplos, explicaciones o código bien comentado?

→ ChatGPT (Web o API)

¿Prefieres trabajar desde tu editor de código directamente?

→ GitHub Copilot + Copilot Chat

¿Necesitas alternar entre varios modelos fácilmente?

→ Ollama o TextGen WebUI con interfaz visual

¿Tienes pocos recursos y no puedes ejecutar modelos grandes?

→ Usar nube (ChatGPT, Gemini)

## ¿Qué herramienta deberías usar?

Elige según tu caso de uso

¿Trabajas en proyectos privados o confidenciales?

→ LLM local (Ollama, LM Studio)

¿Quieres ejemplos, explicaciones o código bien comentado?

→ ChatGPT (Web o API)

¿Prefieres trabajar desde tu editor de código directamente?

→ GitHub Copilot + Copilot Chat

¿Necesitas alternar entre varios modelos fácilmente?

→ Ollama o TextGen WebUI con interfaz visual

¿Tienes pocos recursos y no puedes ejecutar modelos grandes?

→ Usar nube (ChatGPT, Gemini)

## ¿Qué herramienta deberías usar?

Elige según tu caso de uso

¿Trabajas en proyectos privados o confidenciales?

→ LLM local (Ollama, LM Studio)

¿Quieres ejemplos, explicaciones o código bien comentado?

→ ChatGPT (Web o API)

¿Prefieres trabajar desde tu editor de código directamente?

→ GitHub Copilot + Copilot Chat

¿Necesitas alternar entre varios modelos fácilmente?

→ Ollama o TextGen WebUI con interfaz visual

¿Tienes pocos recursos y no puedes ejecutar modelos grandes?

→ Usar nube (ChatGPT, Gemini)



## ¿Qué herramienta deberías usar?

Elige según tu caso de uso

¿Trabajas en proyectos privados o confidenciales?

→ LLM local (Ollama, LM Studio)

¿Quieres ejemplos, explicaciones o código bien comentado?

→ ChatGPT (Web o API)

¿Prefieres trabajar desde tu editor de código directamente?

→ GitHub Copilot + Copilot Chat

¿Necesitas alternar entre varios modelos fácilmente?

→ Ollama o TextGen WebUI con interfaz visual

¿Tienes pocos recursos y no puedes ejecutar modelos grandes?

→ Usar nube (ChatGPT, Gemini)

# Panorama de Herramientas LLM

## Herramientas más usadas en desarrollo asistido por IA:

- GitHub Copilot (OpenAI + Microsoft)
- ChatGPT (OpenAI)
- Gemini (Google)
- Modelos en local: Llama, Mistral, Ollama, LM Studio

*Compararemos sus capacidades, casos de uso y cómo integrarlas en workflows reales.*

## Ejercicio Guiado (Parte 3)

**Objetivo:** Comparar un mismo prompt en dos herramientas (Copilot y ChatGPT).

### Consigna

Usa el siguiente prompt en ambas herramientas:

"Genera una función en Python que lea un archivo CSV y calcule el promedio por columna numérica."

- Evalúa: calidad del código, legibilidad, tratamiento de errores.
- ¿Hay diferencias en el estilo o profundidad?

## Discusión: Herramienta más adecuada

### Preguntas para el grupo:

- ¿En qué casos usarías Copilot o ChatGPT, o usarías modelos locales?
- ¿Qué ventajas y límites notaste?
- ¿Qué características te gustaría que tuvieran?

*Usar bien una herramienta comienza por entender sus sesgos y fortalezas.*

## Resumen: Herramientas y Aplicaciones

### Checklist para elegir e integrar herramientas LLM:

- ¿Conozco las capacidades y límites de **Copilot**, **ChatGPT** y **Gemini**?
- ¿He probado cómo se comportan con tareas reales de desarrollo?
- ¿Entiendo cuándo usar modelos en la **nube** o **locales**?
- ¿Sé integrar la herramienta en mi flujo: editor, navegador o entorno cloud?
- ¿Soy consciente del impacto de la herramienta sobre la productividad y privacidad?

*"No se trata de elegir la mejor herramienta, sino la más adecuada al contexto."*

## Resumen – Ecosistema de Herramientas con LLMs

