

# LLMs en Programación

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

Clase 4: Implicaciones  
Éticas y Mejores Prácticas

Christian Luna Escudero

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

24 de septiembre de 2025

KDIS Lab



DaSCI

## ¿Qué aprenderéis hoy?

- Comprender los riesgos éticos asociados al uso de LLMs en desarrollo.
- Identificar buenas prácticas para un uso responsable y productivo.
- Fomentar la reflexión crítica y profesional entre los desarrolladores.
- Consejos para agilizar e integrar los LLMs en nuestras aplicaciones y sistemas.

## Desarrollo asistido: ¿Qué implica realmente?

- El uso de LLMs como ChatGPT, Copilot o Gemini en tareas de desarrollo es cada vez más común.
- El desarrollador no parte de cero, sino que se apoya en sugerencias generadas automáticamente.
- Esto plantea una nueva dinámica: ¿quién es el autor del código? ¿y el responsable de los errores?

## Actividades que suelen automatizarse

- Generación de código (funciones, scripts, estructuras).
- Refactorización y optimización.
- Generación de documentación automática.
- Pruebas unitarias sugeridas por el modelo.
- Explicación de código heredado.
- Desarrollo de prototipos / PMV (Producto Mínimos Viables)

### Ventaja

Ahorro de tiempo, acceso a buenas prácticas, reducción de tareas repetitivas.

## ¿Por qué debemos preocuparnos?

- El código sugerido no siempre es correcto ni seguro.
- El modelo puede estar influenciado por sesgos presentes en sus datos de entrenamiento.
- El desarrollador puede adoptar sugerencias sin una revisión crítica.

### Consecuencia

Aparece una nueva fuente de riesgo: la alucinación asistida por IA.

## ¿Por qué debemos preocuparnos?

- El código sugerido no siempre es correcto ni seguro.
- El modelo puede estar influenciado por sesgos presentes en sus datos de entrenamiento.
- El desarrollador puede adoptar sugerencias sin una revisión crítica.

### Consecuencia

Aparece una nueva fuente de riesgo: la **alucinación asistida por IA**.

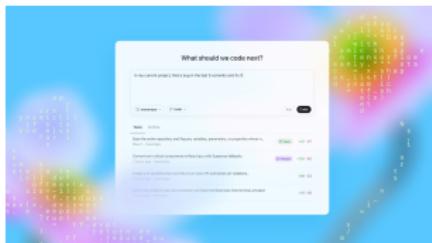
## Agentes de IA para código



Gemini-CLI (Google)



Claude Code (Anthropic)



Codex (OpenAI)



Agent Mode  
Github-Copilot (Microsoft)

## Riesgos Éticos Frecuentes

- **Sesgos:** Reproducción de estereotipos o prejuicios.
- **Privacidad:** Exposición accidental de datos sensibles.
- **Seguridad:** Código inseguro, inyecciones, malas prácticas.



[Link blog](#)

## Fugas de secretos: Copilot en acción

- En Reddit se documentó que Copilot reproducía una API key filtrada en un repositorio público; el usuario confirmó que era real ([link](#)).
- Un estudio de GitGuardian revela que los repositorios con Copilot tienen un 40 % más riesgo de exponer claves o tokens comparados con repos comunes ([link](#)).
- Investigadores extrajeron 2.702 credenciales reales de Copilot y CodeWhisperer, con un 7,4 % verificadas activas.

### Mitigación

Usa secret managers, no insertes secretos en prompts, y audita código antes de aceptar sugerencias.

## EchoLeak: cero-click en Copilot

- En junio 2025 se reveló EchoLeak (CVE-2025-32711): permite extraer datos confidenciales sin interacción del usuario (“zero-click”) ([link](#)).
- Impacto: acceso a emails, documentos y chats internos, CVSS 9.3, solucionado antes de explotarse en producción ([link](#)).

### Lección

Apenas un correo con payload malicioso puede desencadenar una fuga interna; las auditorías y parches son vitales.

## Inyección de prompt y data leak en LLMs

- OWASP advierte: peligro de exposición de datos privados, inyección de prompt y bias en modelos LLM ([link](#)).
- El método “Imprompter” logra un 80 % de éxito extrayendo datos sensibles del modelo ([web oficial](#)).
- Caso en salud: modelos en radiología podrían liberar datos clínicos sensibles si se manipulan adecuadamente ([paper](#)).

### Clave

Mantener separadas las instrucciones del sistema y el input del usuario; filtrar respuestas; capacitar al equipo.

## Sesgos y vulnerabilidades detectadas por la investigación

- Estudio de Y. Fu et al.: el 27-30 % del código sugerido por Copilot/CodeWhisperer contiene debilidades CWE graves: inyección, XSS, valores no aleatorios ([paper](#)).
- Análisis de extensiones VSCode revela que 8,5 % filtran credenciales entre extensiones o entrada de usuario ([paper](#)).

### Discusión

¿Cómo podemos revisar y auditar correctamente el código sugerido? Tipos de herramientas y procesos ¿manuales, automáticos, mixtos?

## Sesgos de género y raza en código generado

- Estudios muestran que entre el 13 % y 49 % del código generado por LLMs refleja comportamientos sesgados hacia el género ([paper](#)).
- Frameworks como Solar revelan que todos los LLM analizados presentan “sesgos severos” en código relacionado con función social ([paper](#)).
- Más del 27 % del código propuesto incluye fallos graves como inyección o XSS, peor en contextos sensibles ([paper](#)).

### Reflexión

¿Qué significa esto para sistemas que automatizan decisiones críticas (hiring, salud, finanzas)?

## ¿Hay sesgos “ocultos”? Investigaciones recientes

- Incluso modelos ajustados para ser “justos” siguen mostrando perjuicios implícitos ([paper](#)).
- MIT revela que reducciones de sesgos en inglés no se replican en otros idiomas/culturas ([paper](#)).
- Lasso cifra: el 13 % de los prompts corporativos contiene datos sensibles ([link](#)) y ([link](#)).

### Discusión

¿Bastan las mitigaciones automáticas o se necesita supervisión humana en varios niveles?

## LMs facilitando amenazas: ejemplo malware

- Investigadores usaron un “rol de superhéroe” para engañar a ChatGPT generando malware capaz de robar contraseñas ([link](#)).
- Parchado y supervisión no eliminan el riesgo: herramientas sociales (phishing, deepfakes) se amplifican con IA ([link](#)).

### Preguntas

¿Cómo establecer barreras éticas y técnicas? ¿Qué tanto confiamos en filtros automáticos?

# Discusión Abierta

## Preguntas clave

- ¿Qué responsabilidad tiene el desarrollador frente al código sugerido?
- ¿Cómo detectar sesgos en el output?
- ¿Hasta dónde confiar en el modelo?

# Buenas prácticas: seguridad y privacidad

## Prompts

No incluir nombres reales, tokens, contraseñas ni datos personales.

## Control de calidad

Revisar línea por línea el código generado, especialmente I/O y validaciones.

## Entornos de prueba

Probar código sugerido en entornos aislados (sandbox, contenedores).

## Versionado

Registrar si una porción fue sugerida por IA (commit message o comentarios).

## Registro ético

Documentar cuándo y por qué se usó IA en decisiones sensibles.

## Recomendaciones adicionales

- Usa herramientas como truffleHog, gitleaks o detect-secrets para buscar filtraciones accidentales en repos.
- Usa "hints" positivos en tus prompts: "genera código seguro" o "sigue buenas prácticas OWASP".
- Integra revisión humana obligatoria si el output será usado en producción o decisiones que afecten a usuarios reales.
- Compara código generado por LLM con documentación oficial (e.g., OWASP, MITRE CWE).

### Atención

La automatización no sustituye a la revisión ética ni al análisis contextual humano.

## Checklist Operativa de Seguridad (antes de aceptar código del LLM)

- ① **Linter & Formateo:** estandariza estilo y detecta olores.
- ② **SAST/Análisis de seguridad:** ejecuta escáner (p. ej., Bandit/semgrep).
- ③ **Secret-scanning:** CI con gitleaks/truffleHog/detect-secrets.
- ④ **Sandbox/Entornos aislados:** ejecuta el snippet sin tocar prod ni datos reales.
- ⑤ **Revisión humana obligatoria (PR):** doble par de ojos en cambios sensibles.
- ⑥ **Trazabilidad en commits:** indicar si fue “generado con LLM + ajustes”.

### Idea clave

La automatización acelera, pero la *aceptación* exige controles.

## Licencias & Compliance (código generado por IA)

- Evitar *copy paste* largo: pedir **re-escritura** “en estilo X” si sospechas licencia.
- **No** incluir PII/credenciales en prompts; política de datos clara.
- Añadir comentario/commit: // generado con LLM, adaptado manualmente.
- Preferir dependencias oficiales y guías (OWASP, MITRE CWE) para validar patrones.
- Mantener un registro de decisiones cuando el impacto es alto (compliance/auditoría).

### Práctica recomendada

Establecer una *policy* interna “Uso de IA en dev” (trazabilidad, revisión, datos).

# AI-Use Record (plantilla de trazabilidad)

## Formato mínimo por cambio relevante

<b>Prompt</b>	(qué pedí al modelo)
<b>Salida del LLM</b>	(fragmento, resumen o enlace al artefacto)
<b>Riesgos revisados</b>	(secretos, PII, CWE, licencias)
<b>Cambios manuales</b>	(qué modifiqué y por qué)
<b>Tests y resultados</b>	(unit/integration, % cobertura, casos negativos)
<b>Decisión final</b>	(merge/no-merge + firmantes)

## Beneficio

Trazabilidad y *accountability* sin fricción: útil para auditorías y post-mortem.

## Iteración Productiva con LLMs

- **Generar código rápidamente** no es el fin, sino el punto de partida.
- El ciclo clave: Prompt → Output → Validación → Mejora del prompt.
- Aprende de las “alucinaciones”: ¿por qué falló?, ¿qué faltó en el contexto?
- Corrige, reintenta y documenta los resultados parciales.

### Ejemplo de flujo

LLM propone un endpoint → el código lanza excepción → modifacas el prompt → versión robusta.

## Iteración Productiva con LLMs

- **Generar código rápidamente** no es el fin, sino el punto de partida.
- El ciclo clave: Prompt → Output → Validación → Mejora del prompt.
- Aprende de las “alucinaciones”: ¿por qué falló?, ¿qué faltó en el contexto?
- Corrige, reintenta y documenta los resultados parciales.

### Ejemplo de flujo

LLM propone un endpoint → el código lanza excepción → modifacas el prompt → versión robusta.

## Prompting no es arte, es ingeniería

- Cada cambio en el prompt es una hipótesis que se prueba con un output.
- Prompts vagos generan código ambiguo o incorrecto.
- Prompts con contexto claro y restricciones explícitas producen mejores resultados.
- La iteración con feedback técnico (de pruebas o de revisión manual) acelera el aprendizaje.

## Principios para Maximizar la Productividad

- ① Prompt claro, con contexto y objetivo.
- ② Aprovechar ejemplos anteriores y plantillas.
- ③ Integrar LLMs en el IDE o pipeline CI/CD.
- ④ Documentar todo: qué se usó, por qué, cómo.

## 1. Prompt claro, con contexto y objetivo

- Define: ¿Qué quieras que genere exactamente? ¿Para qué lenguaje, entorno o framework?
- Evita prompts vagos como: "hazme un login".
- Mejora con contexto: "genera un login en React con validación y mensajes de error accesibles".
- Añade formato esperado: "devuelve solo el código sin explicaciones".

### Regla práctica

Cuento más específico y estructurado el prompt, menos iteraciones innecesarias.

## 1. Prompt claro, con contexto y objetivo

- Define: ¿Qué quieras que genere exactamente? ¿Para qué lenguaje, entorno o framework?
- Evita prompts vagos como: "hazme un login".
- Mejora con contexto: "genera un login en React con validación y mensajes de error accesibles".
- Añade formato esperado: "devuelve solo el código sin explicaciones".

### Regla práctica

Cuanto más específico y estructurado el prompt, menos iteraciones innecesarias.

## 2. Usa ejemplos anteriores y plantillas

- Si un prompt funcionó bien antes, reutilízalo como base.
- Mantén un “prompt log” o biblioteca de plantillas propias (Markdown, Obsidian, Snippet Manager).
- Crea “macroprompts” para tareas comunes: refactorizar, documentar, testear, generar boilerplate.

### Consejo

La productividad se construye sobre la experiencia acumulada, no sobre la improvisación continua.

## 2. Usa ejemplos anteriores y plantillas

- Si un prompt funcionó bien antes, reutilízalo como base.
- Mantén un “prompt log” o biblioteca de plantillas propias (Markdown, Obsidian, Snippet Manager).
- Crea “macroprompts” para tareas comunes: refactorizar, documentar, testear, generar boilerplate.

### Consejo

La productividad se construye sobre la experiencia acumulada, no sobre la improvisación continua.

### 3. Integra LLMs en tu flujo de trabajo

- Usa extensiones como **Copilot Chat** o **Continue** en VSCode para asistencia directa.
- Conecta ChatGPT vía API a tus scripts para generación automatizada de documentación o pruebas.
- Usa LLMs como parte de pipelines de CI/CD: generación de changelogs, tests sintéticos, revisión inicial de PRs.

#### Clave

Los LLMs no deben interrumpir tu flujo, deben potenciarlo sin fricción.

### 3. Integra LLMs en tu flujo de trabajo

- Usa extensiones como **Copilot Chat** o **Continue** en VSCode para asistencia directa.
- Conecta ChatGPT vía API a tus scripts para generación automatizada de documentación o pruebas.
- Usa LLMs como parte de pipelines de CI/CD: generación de changelogs, tests sintéticos, revisión inicial de PRs.

#### Clave

Los LLMs no deben interrumpir tu flujo, deben potenciarlo sin fricción.

## 4. Documenta todo: qué, por qué y cómo

- Anota qué partes del código fueron generadas, corregidas o inspiradas por IA.
- Documenta las decisiones tomadas a partir de la sugerencia (ej.: “adaptado para ser compatible con Django 4.x”).
- Deja trazabilidad en commits o comentarios para que el equipo entienda el origen.

### Recordatorio

Un equipo productivo necesita claridad, y eso se logra con documentación concisa y honesta.

## 4. Documenta todo: qué, por qué y cómo

- Anota qué partes del código fueron generadas, corregidas o inspiradas por IA.
- Documenta las decisiones tomadas a partir de la sugerencia (ej.: “adaptado para ser compatible con Django 4.x”).
- Deja trazabilidad en commits o comentarios para que el equipo entienda el origen.

### Recordatorio

Un equipo productivo necesita claridad, y eso se logra con documentación concisa y honesta.

## Herramientas de apoyo para un flujo eficiente

- **GitHub Copilot:**

- Sugiere código en tiempo real dentro del editor (VS Code, JetBrains).
- Muy útil para completar estructuras repetitivas o scaffolding.
- Ideal para integrarse en flujos ágiles o pruebas rápidas.

- **ChatGPT (Web / API):**

- Versátil para generar código, explicar errores, convertir entre lenguajes.
- Uso por API permite automatización (scripts, generación de tests, documentación).

- **Gemini (Google Docs, Gmail):**

- Refactoriza y explica directamente en documentos técnicos.
- Genera correos, informes, actas de reunión, etc.

- **Modelos locales con LM Studio:**

- Permiten usar LLMs sin conexión y con mayor control sobre privacidad.
- Soportan modelos como LLaMA, CodeLlama, Gemma, Mistral, etc.
- Útiles en entornos cerrados o regulados (educación, investigación, industria).

## Comparativa de herramientas

Herramienta	Código	Texto/Docs	Privacidad	Integración
GitHub Copilot	★★★★★	★★★☆☆	Baja	IDE (VS Code, JetBrains)
ChatGPT	★★★★☆	★★★★☆	Media	Web, API
Gemini	★★★★★	★★★★★	Baja	Google Workspace
LM Studio	★★★★★	★★★☆☆	★★★★★	Local/CLI
Claude	★★★★★	★★★★★	Media	Web, API
Tabnine	★★★☆☆	★★☆☆☆	Media	IDE (VS Code, JetBrains)
Cursor	★★★★☆	★★☆☆☆	Media	IDE propio
Codeium	★★★★★	★★★☆☆	Media	IDE (VS Code, JetBrains)
Perplexity AI	★★★★☆	★★★★★	Media	Web, API

# Buenas Prácticas Generales en Desarrollo Asistido

Área	Práctica Recomendada y Justificación
Prompting	<ul style="list-style-type: none"><li>● Sé específico sobre el lenguaje, formato, restricciones o dependencias.</li><li>● Añade contexto (entorno, objetivo, requisitos funcionales).</li><li>● Itera si el resultado no cumple expectativas: cambia el enfoque o divide la tarea.</li></ul>
Revisión	<ul style="list-style-type: none"><li>● No ejecutes directamente lo generado sin revisar lógica, validaciones y compatibilidad.</li><li>● Usa linters, tests y revisión cruzada en trabajo colaborativo.</li><li>● Considera riesgos como alucinaciones o errores sutiles.</li></ul>
Colaboración	<ul style="list-style-type: none"><li>● Documenta qué parte del código o texto fue generada por IA.</li><li>● Usa comentarios claros, como: // generado con ChatGPT, adaptado manualmente.</li><li>● Ayuda a otros a entender el origen y la razón de esa implementación.</li></ul>

# Casos reales recientes: ¿por qué importa la seguridad en IA?

## Riesgos demostrados en 2024–2025

- **EchoLeak (CVE-2025-32711):** fuga de datos *zero-click* en Microsoft 365 Copilot mediante inyección indirecta (contenido “inocente” en emails/documentos) ([link](#)).
- **Imprompter:** ataques de exfiltración de PII contra agentes LLM con tasas de éxito cercanas al 80 % en evaluación extremo a extremo.
- **Filtrado de secretos:** repos con Copilot presentan mayor probabilidad de secretos expuestos (incremento relativo significativo).

## Mitigación rápida

- Separar instrucciones del sistema y entrada de usuario; validación/sanitización del output.
- Secret managers + escaneo automático de secretos en CI/CD.
- Políticas de datos: no incluir credenciales ni PII en prompts.

*Conclusión: los fallos no son teóricos; ya hay vectores practicables y medibles.*

## Simulación de incidentes éticos (role-play en grupos)

### Objetivo

Analizar un incidente y diseñar un plan de contención y prevención.

- ① **Escenario A:** documento con pie de página malicioso provoca fuga vía asistente (ataque indirecto).
- ② **Escenario B:** prompt ambiguo induce a exponer PII en un chat interno.
- ③ **Escenario C:** commit con código sugerido por LLM filtra un token.

## Cierre y Reflexión

- Los LLMs son herramientas poderosas, pero no infalibles.
- Una práctica responsable combina: criterio técnico, revisión humana y conciencia ética.
- Como desarrolladores, somos responsables de lo que validamos, compartimos o desplegamos.
- El uso ético de la IA es parte de nuestra identidad profesional.

### Pregunta final

¿Usas la IA para ahorrar tiempo o para ser mejor desarrollador?

# Árbol de decisión: ¿qué uso hoy (Prompting, RAG o Fine-Tuning)?

- **¿Tengo datos propios que cambian a menudo?**  
⇒ **RAG** (embeddings + vector DB). Citas y grounding > reentrenar.
- **¿Necesito un estilo/voz/comportamiento muy específico y tengo dataset?**  
⇒ **Fine-Tuning** (PEFT/LoRA). Dataset de calidad + evaluación.
- **¿El problema es acotado y cabe en un buen prompt con ejemplos?**  
⇒ **Prompting** (plantillas, restricciones, formato estricto).

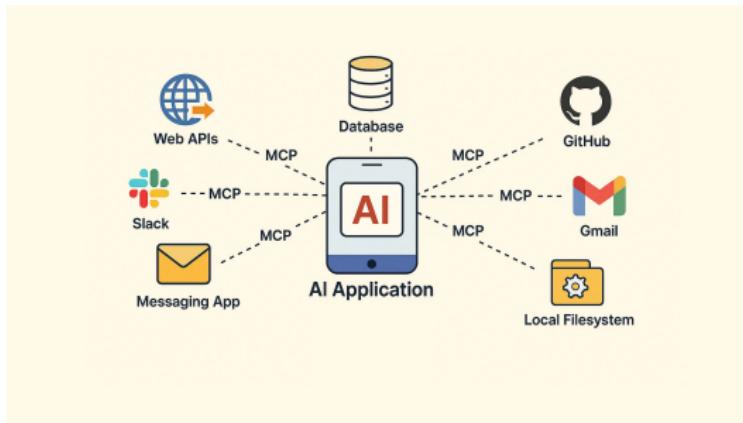
## Reglas rápidas

**RAG** para conocimiento actualizado y citables; **FT** para comportamiento/estilo;  
**Prompting** para agilidad y prototipos.

*Clase 5: veremos comparativa profunda, patrones híbridos y LangChain.*

# Protocolos emergentes (MCP)

¿Te imaginas poder unir cualquier aplicación existente a un modelo de IA?



## Documentación Oficial de MCP

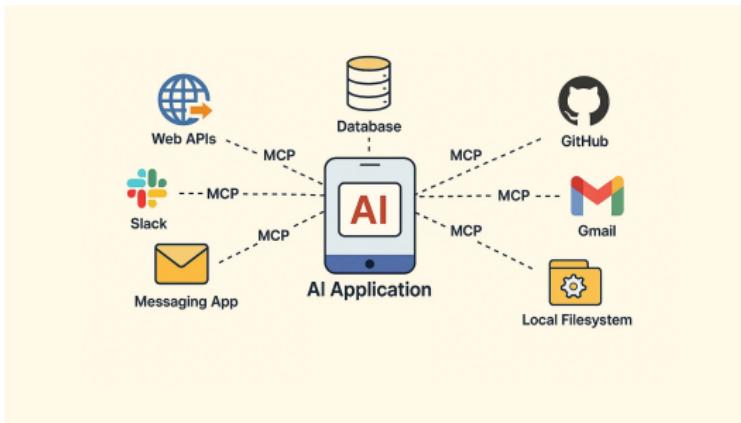
**Qué es:** Protocolo/estándar para conectar LLMs con *tools* y datos de forma controlada.

**Idea clave:** Desacoplar *modelo*, *recuperación* (RAG) y *tool-use* con permisos y auditoría.

- Ventaja: integración más segura y consistente (IDEs, APIs, flujos).
- Relación: complementa Prompting/RAG/FT (no los sustituye).

## Protocolos emergentes (MCP)

¿Te imaginas poder unir cualquier aplicación existente a un modelo de IA?



### Documentación Oficial de MCP

**Qué es:** Protocolo/estándar para conectar LLMs con *tools* y datos de forma controlada.

**Idea clave:** Desacoplar *modelo*, *recuperación* (RAG) y *tool-use* con permisos y auditoría.

- Ventaja: integración más segura y consistente (IDEs, APIs, flujos).
- Relación: complementa Prompting/RAG/FT (no los sustituye).