# VibeCoding

No es la forma normal de programar, consiste en que le decis a la ia lo que queres con tus palabras y estas te genera el código automáticamente LLM, pero sirve nada mas para crear cosas de forma básica pero sin ninguna calidad en el código. Esto puede servir para personas sin conocimiento en programación, pero muchas te código sin calidad. Andrej Karpathy (cofundador de Open IA y Director en Proyecto de IA en Tesla) fue el que creo el termino de vibecodign por primera vez en un tweet de twiter. El dice que vibecodign es fluir con la ia porque es algo muy divertido, pero que eso no es programar. El termino de vibecoign se refiere a jugar ser programador sin ser programador. Pero esto sugiere que antes de sacar un producto a producción requiere mucho análisis de calidad en el código. Entonces el vibecoding no es una buena práctica. Una buena práctica es aplicar el Secure AI Development. Son buenas prácticas para desarrollar con IA

* <https://medium.com/data-science-collective/the-vibe-coding-mind-virus-explained-3-simple-rules-to-vibe-your-way-without-breaking-things-4762553fecb3>

# Secure AI Development

El Concepto de "Secure AI Development", o desarrollo de inteligencia artificial (IA) con un énfasis fuerte en la seguridad y privacidad integradas desde el inicio del ciclo de vida del producto. Se trata de aplicar principios de seguridad por diseño en cada etapa del desarrollo y despliegue de sistemas de IA, para protegerlos contra ataques y asegurar su confianza y privacidad. Algunos elementos clave, basados en marcos como el Secure AI Framework (SAIF) de Google, incluyen ([safety.google](https://safety.google/cybersecurity-advancements/saif/?utm_source=chatgpt.com" \o "Google's Secure AI Framework - Google Safety Center)):

1. Fundamentos robustos de seguridad: controles básicos como cifrado de datos, gestión de acceso y seguridad en la infraestructura.
2. Detección y respuesta: monitoreo continuo contra amenazas específicas de IA, como ataques adversariales o inyecciones de prompt.
3. Defensas automatizadas: uso de sistemas automáticos que detectan y contrarrestan nuevas amenazas en tiempo real.
4. Controles coherentes: aplicar las mismas políticas de seguridad en todo el ciclo de vida de IA, desde desarrollo hasta producción.
5. Retroalimentación rápida: adaptar y corregir medidas de seguridad a partir de incidentes o pruebas.
6. Contexto empresarial: entender los riesgos según el uso del modelo dentro del proceso de negocio ([safety.google](https://safety.google/cybersecurity-advancements/saif/?utm_source=chatgpt.com" \o "Google's Secure AI Framework - Google Safety Center)).

Este concepto es bastante importante debido a que ofrece lo siguente:

* Protección contra amenazas específicas: como robo del modelo, envenenamiento de datos, o extracción de información sensible ([developers.google.com](https://developers.google.com/machine-learning/resources/saif?utm_source=chatgpt.com)).
* Mayor resiliencia y confianza: asegurando que los modelos operen de forma robusta y justa, con explicabilidad y cumplimiento normativo .
* Cumplimiento regulatorio: especialmente relevante con nuevas normativas como la Ley de IA de la UE o directrices de agencias como CISA ([wired.com](https://www.wired.com/story/cisa-ai-roadmap?utm_source=chatgpt.com)).

Esta metodología de trabajo se puede aplicar de la siguente manera

* Evaluaciones de riesgo: análisis formales de amenazas específicas antes de lanzar modelos.
* Pruebas adversariales ("red teaming"): simular ataques para identificar debilidades.
* Controles de privacidad: técnicas como federated learning o procesamiento homomórfico para proteger datos sensibles ([reuters.com](https://www.reuters.com/technology/artificial-intelligence/toyota-ntt-invest-33-bln-development-ai-platform-2024-10-31/?utm_source=chatgpt.com), [splunk.com](https://www.splunk.com/en_us/blog/learn/secure-ai-system-development.html?utm_source=chatgpt.com)).
* Gobernanza y responsabilidad: roles definidos, supervisión continua y transparencia sobre cómo y quién usa los sistemas de IA.

El Secure AI Development es un enfoque holístico que usa frameworks como SAIF para integrar la seguridad, privacidad, transparencia y gobernanza desde la concepción hasta la explotación de sistemas de IA, reduciendo riesgos y mejorando su confiabilidad.

* <https://www.ncsc.gov.uk/collection/guidelines-secure-ai-system-development>
* <https://safety.google/cybersecurity-advancements/saif/>
* <https://www.cyber.gc.ca/en/news-events/guidelines-secure-ai-system-development>

**Aplicación de Secure AI Development al crear app en Android Studio**

Para aplicar los principios de Secure AI Development en una app creada con Android Studio, lo mejor es integrar seguridad desde el diseño hasta el despliegue. A continuación, se describe una una guía práctica que combina seguridad móvil con particularidades de IA.

## Seguridad en el almacenamiento y manejo de datos

* **Encriptación en reposo**: utiliza EncryptedSharedPreferences o SQLCipher para guardar datos sensibles (tokens, preferencias, etc.) cifrados ([medium.com](https://medium.com/%40rushabhprajapati20/enhancing-security-in-android-development-best-practices-and-strategies-fb71b748d60b?utm_source=chatgpt.com)).
* **Almacena llaves con Android Keystore**: permite generar y usar claves criptográficas dentro de un entorno hardware seguro .

## Seguridad en las comunicaciones

* **Usa HTTPS y TLS 1.2+ o TLS 1.3**: todo el tráfico de red debe cifrarse para proteger datos en tránsito ([medium.com](https://medium.com/%40PedalsUp/security-tips-and-best-practices-to-develop-secure-mobile-applications-b2576c38d026?utm_source=chatgpt.com)).
* **Certificate pinning**: bloquea ataques man-in-the-middle asegurando que la app se comunique solo con certificados legítimos ([medium.com](https://medium.com/%40rushabhprajapati20/enhancing-security-in-android-development-best-practices-and-strategies-fb71b748d60b?utm_source=chatgpt.com)).
* **Validación de API Keys**: nunca codifiques llaves dentro del APK; en su lugar, gestionalas desde un backend seguro o usa servicios como Firebase App Check para validar integridad ([reddit.com](https://www.reddit.com/r/androiddev/comments/1ew1fng?utm_source=chatgpt.com)).

## Prácticas seguras en el código

* **Obfuscación con ProGuard/R8**: hace más difícil la ingeniería inversa y protección del modelo de IA ([medium.com](https://medium.com/%40rushabhprajapati20/enhancing-security-in-android-development-best-practices-and-strategies-fb71b748d60b?utm_source=chatgpt.com)).
* **Principio de menor privilegio y manejo de permisos:** solicita solo los permisos estrictamente necesarios y pídelos en tiempo de ejecución ([ripenapps.com](https://ripenapps.com/blog/android-application-security-best-practices/?utm_source=chatgpt.com)).
* **Validar entradas del usuario:** previene ataques como inyección de SQL o XSS, tanto en cliente como en servidor ([estatic-infotech.com](https://www.estatic-infotech.com/blog/post/building-secure-android-apps-best-practices?utm_source=chatgpt.com)).
* **Revisiones de código y análisis estático:** usa herramientas como SonarQube, Android Lint o MobSF para detectar vulnerabilidades ([medium.com](https://medium.com/%40rushabhprajapati20/enhancing-security-in-android-development-best-practices-and-strategies-fb71b748d60b?utm_source=chatgpt.com)).
* **Pruebas de penetración:** aplicar red‑teaming y pentesting regularmente .

## Integridad y protección de la app

* **Detección de root y entornos comprometidos**: evita que la app corra en dispositivos inseguros ([medium.com](https://medium.com/%40rushabhprajapati20/enhancing-security-in-android-development-best-practices-and-strategies-fb71b748d60b?utm_source=chatgpt.com)).
* **Integridad del APK:** verifica firma y checksum y detecta si ha sido modificado ([synopsys.com](https://www.synopsys.com/blogs/software-security/android-security-best-practices.html?utm_source=chatgpt.com)).
* **FLAG\_SECURE en WebViews o Activity**: impide capturas de pantalla para proteger datos sensibles .

## Seguridad aplicada a la IA

* **Modelo en backend**: si tu IA requiere prohibir exposición del modelo, mantén el procesamiento en tu servidor y envía solo resultados al cliente, evitando exponer pesos o lógica en el APK.
* **API segura para inferencia:** protege los endpoints de IA con autenticación fuerte (OAuth2, JWT) y autorización por roles ([medium.com](https://medium.com/%40kajal.suthar/best-practices-for-android-app-security-protecting-user-data-e2ac634f169a?utm_source=chatgpt.com)).
* **Privacidad de datos entrada/salida:** cifra los datos en tránsito, anonimiza inputs cuando sea posible y cumple con regulaciones de privacidad.
* **Pruebas adversariales (Red‑teaming):** intenta vulnerar tu modelo (ataques adversariales, extracción de modelo) antes de lanzar.

## Monitoreo, actualización y gobernanza

* **Actualizaciones constantes:** mantén dependencias, SDKs y parches Android actualizados ([developer.android.com](https://developer.android.com/privacy-and-security/security-best-practices?utm_source=chatgpt.com)).
* **Logs y detección de anomalías**: monitoriza comportamientos extraños o fallos en producción (integridad, root, intentos no autorizados).
* **Principio Secure‑by‑Design:** integra la seguridad desde la planificación inicial, no como añadido posterior ([en.wikipedia.org](https://en.wikipedia.org/wiki/Secure_by_design?utm_source=chatgpt.com)).

**Lista rápida de acciones en Android Studio:**

1. **Codifica de forma segura:** ProGuard/R8 + revisión de código.
2. **Configura Android Keystore + EncryptedSharedPreferences.**
3. **Forza HTTPS + certificate pinning con OkHttp.**
4. **Solicita permisos en tiempo real, solo los necesarios.**
5. **Detecta root/jailbreak y entornos adulterados.**
6. **Implementa llamadas a un backend para IA (con OAuth2/JWT).**
7. **Cifra datos sensibles antes de guardarlos o enviarlos.**
8. **Ejecuta análisis estático (Lint/Sonar) y pentesting (MobSF, OWASP MASTG).**
9. **Firma+verifica APK y activa FLAG\_SECURE.**
10. **Crea un pipeline de CI/CD que incluya pruebas de seguridad y actualizaciones automáticas.**

**Aplicación de Secure IA Development al crear aplicaciones IOT con ESP32**

Para aplicar **Secure AI Development** en una aplicación IoT basada en **ESP32**, conviene integrar seguridad y buenas prácticas en cada capa: hardware, firmware, comunicaciones, IA y mantenimiento. Aquí tienes una guía estructurada:

## Seguridad en hardware y firmware

1. Secure Boot + Flash Encryption

Protege que solo se ejecute firmware auténtico y evita accesos físicos no autorizados:

* Habilita **Secure Boot V2** (RSA-PSS) y **flash encryption,** idealmente con clave única por dispositivo y eFuses bloqueados ([reddit.com](https://www.reddit.com/r/esp32/comments/1ew2jrj?utm_source=chatgpt.com), [labprojectsbd.com](https://labprojectsbd.com/2024/09/07/iot-security-best-practices-protecting-your-low-power-devices/?utm_source=chatgpt.com), [esp32.ai](https://www.esp32.ai/idf/esp32/security/security?utm_source=chatgpt.com)).
* Asegúrate de programar claves en un entorno confiable (ej.: en fábrica) y deshabilitar JTAG/UART ([esp32.ai](https://www.esp32.ai/idf/esp32/security/security?utm_source=chatgpt.com)).

1. Almacenamiento seguro

* Usa **NVS encryption** (almacenamiento no volátil cifrado) para credenciales, llaves y certificados ([medium.com](https://medium.com/lifeomic/secure-esp32-device-provisioning-with-hardware-security-ccb72ea5c326?utm_source=chatgpt.com)).

1. Provisionamiento seguro

* Durante el aprovisionamiento inicial, carga credenciales firmadas o cifradas via NVS con soporte de flash encryption ([medium.com](https://medium.com/lifeomic/secure-esp32-device-provisioning-with-hardware-security-ccb72ea5c326?utm_source=chatgpt.com)).

## Comunicación segura

1. TLS para todas las conexiones

* Usa MbedTLS y abstrae seguridad con ESP-TLS para HTTPS o MQTT cifrado (TLS) ([esp32.ai](https://www.esp32.ai/idf/esp32/security/security?utm_source=chatgpt.com)).
* Valida certificados con root CA o implementa **mutual TLS** para autenticación mutua ([reddit.com](https://www.reddit.com/r/esp32/comments/lia89w?utm_source=chatgpt.com)).

1. MQTT + Node‑RED seguro

* Configura MQTT con certificados SSL/TLS y autenticación de clientes. También asegúrate de que Node‑RED use TLS ([es.wikipedia.org](https://es.wikipedia.org/wiki/Mongoose_OS?utm_source=chatgpt.com)).

1. Wi‑Fi robusto

* Usa WPA3/PMF para proteger enlaces Wi‑Fi y evitar ataques como spoofing o deautorización ([esp32.ai](https://www.esp32.ai/idf/esp32/api-guides/wifi-security?utm_source=chatgpt.com)).

## Seguridad del componente IA

1. Procesamiento en backend

Evita exponer lógica o datos sensibles entrenando/ejecutando IA fuera del dispositivo: envía solo entradas/salidas hacia servidores protegidos.

1. Protección de modelo y datos

* Usar API seguras (OAuth2/JWT), con TLS y autenticación fuerte.
* Implementar pruebas adversariales (red‑teaming) para intentar vulnerar inputs y analizar la resistencia del sistema.

## Actualizaciones seguras (OTA)

* Usa **OTA segura sobre HTTPS/TLS y** firma digital de firmware. El dispositivo debe verificar firma y versión antes de aplicar la actualización .
* Habilita **anti-rollback** para impedir reinstalaciones de firmware viejo vulnerables .

## Resiliencia y gobernanza

1. Monitorización y detección

* Implementa registros de eventos críticos (arranques fallidos, actualización no firmadas, dispositivos rooteados).
* PIensa en manejar alertas y respuestas automáticas ante anomalías IA y fallos de integridad.

1. Ciclo de vida seguro

* Revisa y actualiza regularmente ESP‑IDF y librerías para parchear vulnerabilidades conocidas (como problemas con Bluetooth o Wi‑Fi) ([reddit.com](https://www.reddit.com/r/pwnhub/comments/1jdxcst?utm_source=chatgpt.com)).

**Framework de base para IoT seguro**

Puedes basarte en proyectos como **secure\_esp** (GitHub) que ya integran OTA, cifrado, MQTT y verificación ([github.com](https://github.com/flouflouit/secure_esp?utm_source=chatgpt.com)).

También evalúa **Mongoose OS,** que facilita la integración de seguridad, OTA y conectividad en ESP32 con soporte para AWS, certificaciones y cifrado .

**Lista rápida de acciones en IOT con ESP32:**

| **Capa** | **Acción recomendada** |
| --- | --- |
| **Hardware/Firmware** | Secure Boot V2 + flash encryption + eFuses + NVS encrypt |
| **Provisionamiento** | Carga certificados/llaves cifradas durante fabricación |
| **Comunicación** | TLS en wifi, MQTT, HTTPS; mutual TLS; validación de certificados |
| **IA** | Procesamiento en backend, API seguras, pruebas adversariales |
| **OTA** | OTA sobre HTTPS + firma + anti-rollback |
| **Gestión** | Monitoreo de integridad, parcheo continuo, control de versiones |
| **Frameworks base** | Usa secure\_esp o Mongoose OS para acelerar implementación |