Evaluación Comparativa de Agentes de Inteligencia Artificial en la Resolución de Vulnerabilidades de Seguridad en Entornos de Integración Continua

Gonzalo Daniel

Paula Alejandra Casallas

Valentina Rodríguez

Universidad Ean

Facultad de Ingeniería

Ingeniería de Sistemas - Ingeniería Industrial

Bogotá, Colombia

24/02/2026

Evaluación Comparativa de Agentes de Inteligencia Artificial en la Resolución de Vulnerabilidades de Seguridad en Entornos de Integración Continua

Gonzalo Daniel

Paula Alejandra Casallas

Valentina Rodríguez

Trabajo de grado presentado como requisito para optar al título de:

Ingeniero de Sistemas

Ingeniero Industrial

Director (a):

Elizabeth Leon Velasquez

Universidad Ean

Facultad de Ingeniería

Ingeniería de Sistemas - Ingeniería Industrial

Bogotá, Colombia

24/02/202

Nota de aceptación:

Firma del jurado

Firma del jurado

Firma del director del trabajo de grado

Resumen

Esta investigación evaluó la viabilidad de integrar Agentes Autónomos basados en Modelos de Lenguaje Extensos (LLMs) como apoyo en arquitecturas de software complejas. El objetivo principal fue superar la evaluación tradicional basada en generación de código aislado, midiendo la capacidad de la Inteligencia Artificial moderna para mitigar vulnerabilidades reales sin introducir regresiones funcionales que afecten el ciclo de Integración Continua y Despliegue Continuo (CI/CD). La metodología empleó la aplicación web OWASP Juice Shop como entorno vulnerable, interconectado y de alta complejidad, comparando el desempeño de tres modelos de vanguardia disponibles en GitHub Copilot: Gemini 3.1 Pro, Claude Opus 4.6 y GPT-5.4.

Los resultados evidenciaron una eficacia del 100 % en la tarea aislada de corrección de una vulnerabilidad de inyección SQL, sin generación de regresiones funcionales en los escenarios verificados. Ante la subsecuente petición de una auditoría masiva, los modelos evidenciaron conciencia contextual al activar filtros éticos reconociendo el repositorio, estableciendo marcadores no antes vistos en competencia contextual de los modelos de lenguaje de generaciones anteriores.

Se concluye que los agentes evaluados han superado la asistencia en tareas puntuales y muestran potencial para apoyar procesos de mantenimiento, ciberseguridad y validación en entornos de software complejos. No obstante, su adopción en flujos CI/CD requiere mecanismos rigurosos de trazabilidad, validación automatizada y control de cambios, especialmente por la posible contaminación de datos de entrenamiento asociada al uso de repositorios públicos.

Palabras clave: Agentes Autónomos, Modelos de Lenguaje Extensos, Integración y Despliegue Continuo (CI CD), Reparación automatizada de software, Ciberseguridad de Software.

Abstract

This research evaluated the feasibility of integrating Autonomous Agents based on Large Language Models (LLMs) as support in complex software architectures. The main objective was to move beyond traditional evaluation based on isolated code generation, measuring the ability of modern Artificial Intelligence to mitigate real vulnerabilities without introducing functional regressions that affect the Continuous Integration and Continuous Deployment (CI/CD) cycle. The methodology used the OWASP Juice Shop web application as a vulnerable, interconnected, and highly complex environment, comparing the performance of three state-of-the-art models available in GitHub Copilot: Gemini 3.1 Pro, Claude Opus 4.6, and GPT-5.4.

The results showed 100% effectiveness in the isolated task of correcting a SQL injection vulnerability, with no functional regressions generated in the verified scenarios. When faced with the subsequent request for a large-scale audit, the models demonstrated contextual awareness by activating ethical filters upon recognizing the repository, establishing previously unseen benchmarks in contextual competence compared with earlier generations of language models.

It is concluded that the evaluated agents have moved beyond assistance with isolated tasks and show potential to support maintenance, cybersecurity, and validation processes in complex software environments. However, their adoption in CI/CD workflows requires rigorous mechanisms for traceability, automated validation, and change control, especially due to the possible contamination of training data associated with the use of public repositories.

Keywords: Autonomous Agents, Large Language Models, Continuous Integration and Deployment (CI, CD), Automated software repair, Software cybersecurity.

Introducción

La integración de Modelos de Lenguaje Extensos (LLMs) ha transformado radicalmente el ciclo de vida del desarrollo de software (Haque, 2025). Como se evidenció en investigaciones previas desarrolladas en el marco de la asignatura de Seminario de Investigación (Dániel, n.d.), el ecosistema de la Inteligencia Artificial Generativa ha alcanzado un punto de madurez donde la capacidad de generar código funcional a partir de instrucciones aisladas se ha convertido en un estándar de la industria. Dicho estudio previo demostró que los modelos de élite han alcanzado una "comoditización de la correctitud"; es decir, la gran mayoría es capaz de resolver problemas algorítmicos complejos cuando se evalúa su competencia en una sola conversación y con un requerimiento algorítmico específico. Sin embargo, esta paridad operativa reveló una realidad fundamental: la verdadera utilidad de un LLM ya no reside en su habilidad para escribir soluciones algorítmicas desde cero, sino en su capacidad para operar dentro de proyectos y arquitecturas preexistentes con gran cantidad de archivos e interdependencias críticas.

El desarrollo y la integración de software en contextos profesionales rara vez se basan en la creación de sistemas desde cero. Por el contrario, los ingenieros dedican la mayor parte de su tiempo a leer, comprender, refactorizar y actualizar bases de código existentes de alta complejidad, interconectadas y, muy frecuentemente, heredadas (legacy). Respondiendo al llamado a la acción de la investigación anterior de Seminario de Investigación, el presente proyecto propuso evolucionar los marcos de evaluación tradicionales para dar el salto desde el modelo en una conversación hacia el paradigma de los Agentes Autónomos de Inteligencia Artificial.

En el marco de este Proyecto de Integración, el desafío no consiste únicamente en lograr que el código funcione, sino en garantizar que los nuevos componentes y modificaciones se integren de manera coherente con la arquitectura general del sistema, sin provocar fallos en partes que antes

funcionaban correctamente. Para evaluar esta capacidad, este proyecto abandonó las pruebas de laboratorio aisladas y sometió a diferentes agentes impulsados por modelos de lenguaje de vanguardia a un entorno de desarrollo realista y de alta complejidad a la resolución de vulnerabilidades de seguridad y problemas de lógica de negocio en una aplicación web moderna y compleja (OWASP Juice Shop).

A través de la simulación de un flujo de trabajo empresarial con pruebas unitarias integradas, esta investigación midió dimensiones cualitativas críticas de las soluciones generadas por los diferentes agentes. Se evaluaron no sólo la precisión de la solución, sino la comprensión del contexto arquitectónico para mitigar el daño colateral o regresión funcional de los cambios generados.

De este modo, este proyecto proporciona una perspectiva objetiva y cuantificable sobre el estado actual de la oferta de agentes de IA, determinando si están verdaderamente preparados para asumir roles autónomos en los procesos de integración e incluso mantenimiento y despliegue continuo (CI/CD) de un proyecto de ingeniería de software moderno.

Marco teórico

La integración de Modelos de Lenguaje Extensos (LLMs, por sus siglas en inglés) en la ingeniería de software ha marcado un cambio muy importante en el ciclo de vida del desarrollo. En el pasado, las herramientas de asistencia a la programación se centraban en el análisis estático y en motores de autocompletado básicos que no lograban entender el contexto completo del proyecto. Con la llegada de arquitecturas como los Transformers (Vaswani et al., 2023) y grandes modelos de lenguaje como GPT-4 y su capacidad de actuar con agencia, se ha observado un aumento de competencia considerable en competencia de estos agentes en dar valor en diversos proyectos (Bubeck et al., 2023). Estudios recientes han comprobado que el uso de estas herramientas eleva de manera significativa la productividad de los desarrolladores cuando tienen que escribir código desde cero (Peng et al., 2023).

Sin embargo, como señalan Hou et al. (Peng et al., 2023) en su revisión sobre LLMs en la ingeniería de software, los retos actuales de la industria ya no consisten únicamente en escribir funciones aisladas o en crear proyectos desde cero. Por el contrario, la mayor parte del trabajo de los ingenieros consiste en entender, mantener, actualizar y refactorizar bases de código muy grandes y antiguas (legacy). Las limitaciones de los primeros asistentes de programación se hicieron evidentes al no ser competentes en evitar riesgos de seguridad ni daños colaterales. De hecho, estas herramientas podrían llegar a introducir vulnerabilidades al sistema si el desarrollador no realizaba una revisión exhaustiva del código generado. Este problema motivó a la comunidad científica a buscar modelos con un mayor grado de autonomía y capacidad de análisis.

El paso de simples asistentes de programación que solo ejecutan código después de recibir una instrucción humana paso por paso hacia Agentes Autónomos de Inteligencia Artificial es el tema central de la investigación actual. Un agente autónomo impulsado por un LLM se define

como un sistema capaz de utilizar un modelo base como motor principal para razonar y actuar. Este motor se encarga de dirigir ciclos continuos de planificación, ejecución de tareas usando herramientas externas y ciclos de auto-evaluación o reflexión (Yao et al., 2023).

El modelo ReAct (Reasoning and Acting), propuesto por (Yao et al., 2023), plantea que combinar el razonamiento interno del LLM con la capacidad de actuar directamente sobre un entorno de trabajo mejora muchísimo el éxito al resolver problemas complejos. En el campo de la ingeniería de software automatizada, esta idea toma forma a través del diseño de Interfaces Agente-Computadora (ACI), un concepto que fue estructurado por (Yang et al., 2024) en el proyecto SWE-agent. Las ACI permiten que los modelos interactúen de forma natural con el sistema operativo, logrando imitar el comportamiento de un programador humano: escribiendo comandos en la consola, navegando por carpetas y archivos, leyendo documentación y ejecutando pruebas de forma continua hasta lograr el objetivo trazado por el usuario.

Una de las áreas de aplicación más importantes y difíciles para estos agentes autónomos es la Reparación Automatizada de Programas (Automated Program Repair - APR), especialmente cuando se trata de arreglar vulnerabilidades de seguridad y errores en la lógica del negocio. (Xia et al., 2022) explican en su investigación cómo los agentes impulsados por LLM han superado a los métodos antiguos de APR. Estos métodos anteriores dependían de reglas estrictas y plantillas fijas de programación. En cambio, en la actualidad, los LLM pre entrenados demuestran una gran flexibilidad para entender el problema real del código con tan solo leer un reporte de error (issue) subido en una plataforma de control de versiones.

De igual forma, en el área de la seguridad informática corporativa, el estudio realizado por (Fitero-Dominguez et al., 2024) sobre la reparación automatizada de vulnerabilidades demuestra una gran mejoría de los modelos de vanguardia con los modelos de apenas un par de años atrás. En este estudio se evidencia que cuando los agentes reciben retroalimentación constante (es decir,

cuando pueden verificar su código o leer los mensajes de error del compilador para corregirse a sí mismos), logran aumentar de forma considerable la cantidad de parches de seguridad que son verdaderamente funcionales y seguros. A pesar de estos grandes avances, los investigadores han notado un problema muy común: solucionar un error puntual en un archivo de código muchas veces provoca fallos ocultos en otras partes del sistema (daños colaterales). Esto resulta en una en errores del sistema que ya funcionaban correctamente antes de la actuación del agente (regresiones funcionales).

Debido a este problema, surge la necesidad obligatoria de rediseñar la forma en que evaluamos y validamos a estos modelos de Inteligencia Artificial antes de usarlos masivamente en la industria del software. Para poder medir de forma justa y precisa qué tan útiles son las acciones de los agentes autónomos en el mantenimiento real del software, fue absolutamente necesario cambiar las pruebas estándar de evaluación. Antes, la validación se realizaba usando pruebas sencillas como HumanEval y MBPP. Sin embargo, estas pruebas solo analizan fragmentos muy cortos de código y resuelven problemas matemáticos simples, los cuales no tienen ninguna relación con la arquitectura compleja de un proyecto de software empresarial. Frente a esta gran limitación teórica, (Jimenez et al., 2024) y su equipo presentaron a la comunidad un nuevo sistema de evaluación mucho más realista llamado SWE-bench. En muy poco tiempo, este sistema se ha convertido en el estándar más importante en el estado del arte para medir las capacidades de los agentes en flujos de ingeniería de software.

El mecanismo de evaluación de SWE-bench demuestra que la verdadera capacidad de un agente de IA solo se puede comprobar si el modelo logra analizar un problema real (por ejemplo, un issue reportado en un repositorio Open Source de GitHub), procesar adecuadamente todos los archivos y líneas de código del proyecto original, programar una solución de forma completamente independiente y asegurarse de que el programa siga superando todas sus pruebas unitarias sin

romper ninguna funcionalidad original (evitando cualquier regresión transversal o efecto secundario). Para llevar todas estas teorías a la práctica y simular los problemas que ocurren en una empresa de desarrollo real, es necesario hacer los experimentos en sistemas que ya tienen errores premeditados.

Es muy importante observar cómo se comporta el agente de IA en aplicaciones web modernas y de uso empresarial. Por esta razón, OWASP Juice Shop, documentado a fondo por (Kimminich, 2021), es el entorno de simulación ideal para combinar la ciberseguridad con la inteligencia artificial. Esta aplicación está desarrollada con tecnologías muy utilizadas como JavaScript, TypeScript, Angular en el frontend y Node.js en el backend. De forma completamente intencional, esta aplicación contiene todas las vulnerabilidades y fallos principales categorizados en el famoso Top 10 de OWASP.

En resumen, todo este conjunto de teorías sobre modelos que razonan por sí mismos (ReAct), junto con el uso de terminales e interfaces (ACI), y los enfoques para reparar código automáticamente (APR) aplicados a la industria, conformaron una base sólida para nuestro proyecto de investigación. Sobre estos fundamentos teóricos se realizaron las pruebas experimentales y la validación de los modelos LLM, poniéndolos a prueba directamente sobre un proyecto web real, complejo y con fallos estructurales conocidos, con el fin de medir su comportamiento analítico y su eficacia real para evitar regresiones funcionales.

Estado del arte (Metodología PRISMA)

El estado del arte se desarrolló mediante una revisión sistemática y estructurada fundamentada en la metodología PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses). Este diseño metodológico riguroso garantiza que el proceso de identificación, selección y análisis de la literatura científica subyacente a la integración de Agentes Autónomos LLM en la ingeniería de software y la ciberseguridad sea transparente, verificable y replicable.

Para la ubicación de los artículos primarios se llevó a cabo una consulta en bases de datos académicas de alto impacto, específicamente Scopus y Web of Science (WOS), complementada con el repositorio de pre-impresiones arXiv para abarcar la literatura emergente más reciente (esencial dada la velocidad de evolución de la IA). La búsqueda cubrió el periodo temporal desde enero de 2020 hasta marzo de 2026, restringiendo los resultados a documentos publicados en idioma inglés (dada su predominancia en el campo de la computación) y español.

La estrategia de recuperación de información se ejecutó utilizando la siguiente ecuación de búsqueda con operadores booleanos: ("Large Language Models" OR "LLM" OR "Autonomous Agents") AND ("Software Engineering" OR "CI/CD" OR "Vulnerability Resolution") AND ("Benchmark" OR "Security" OR "Evaluation").

Criterios de inclusión y exclusión: Se incluyeron: (1) Artículos empíricos que evalúan el desempeño de LLMs como agentes autónomos en repositorios de software completos. (2) Estudios enfocados en la resolución de problemas lógicos o vulnerabilidades de seguridad. (3) Investigaciones que proponen nuevos benchmarks o marcos de evaluación realistas. Se excluyeron: (1) Revisiones sistemáticas previas o estudios puramente teóricos sin evaluación empírica. (2) Artículos centrados exclusivamente en la generación de fragmentos de código aislado (tipo HumanEval), por carecer de la complejidad arquitectónica necesaria para este proyecto. (3) Literatura gris o publicaciones no sometidas a revisión por pares rigurosos.

El proceso de selección constó de cuatro fases. En la fase de Identificación, la ecuación arrojó un total de 184 registros únicos tras la eliminación manual de duplicados automatizada. Durante la selección se revisaron los títulos y resúmenes, lo que permitió excluir 132 artículos por falta de alineación con la temática del proyecto (ej. uso de LLMs en dominios biomédicos). Se leyeron y analizaron 52 estudios. De estos, 48 fueron excluidos por razones metodológicas: no presentaban pruebas en entornos realistas, carecían de evaluación de regresiones o se limitaban al autocompletado de algoritmos sencillos. Finalmente, en la fase de Inclusión, se seleccionaron 4 artículos clave, publicados y presentados en conferencias top-tier e indexados, que constituyen el corpus medular de este análisis crítico de antecedentes (fijando el panorama de 2023 a 2026).

La revisión retrospectiva de la literatura evidencia una drástica evolución metodológica en la evaluación de la inteligencia artificial para el desarrollo de software. Los estudios pre-2023 consideraban a los Modelos de Lenguaje Extenso principalmente como "loros estocásticos" o sistemas de autocompletado avanzado (asistentes puramente reactivos). Sin embargo, la investigación contemporánea ha migrado hacia la experimentación con arquitecturas de Agentes Autónomos (fundamentados en paradigmas de razonamiento ReAct - Reasoning and Acting), donde el modelo asume agencia computacional para interactuar con terminales, revisar repositorios y ejecutar secuencias iterativas de pruebas.

El estudio fundamental de (Jimenez et al., 2024) marca un punto de inflexión decisivo en esta transición con la publicación de SWE-bench. (herramienta de evaluación comparativa estandarizada y basada en contenedores, diseñada para evaluar la eficacia de los modelos de lenguaje a gran escala).

Este trabajo demostró el error empírico de evaluar los LLMs mediante problemas sintéticos aislados, proponiendo un estándar basado en problemas (issues) extraídos de GitHub para repositorios masivos en lenguaje Python. El hallazgo más contundente de este estudio establece

que los agentes LLM sufren de una severa "miopía arquitectónica"; mientras que logran resolver menos del 10% de los problemas de manera autónoma, fracasan al enfrentarse a refactorizaciones que requieren comprender el contexto general de un proyecto complejo.

A la par, el campo de la seguridad del software revela un grave vacío de conocimiento en torno a la confiabilidad del código impulsado por IA (Bhatt et al., 2024) mediante la iniciativa CyberSecEval *"conjunto de pruebas de rendimiento extenso diseñado para evaluar las vulnerabilidades de ciberseguridad y las capacidades defensivas de los modelos de lenguaje grandes*" (Introduction | CyberSecEval 4, n.d.). En este, se evidenció un dilema: a mayor capacidad general del modelo de lenguaje, se documenta una propensión estadística mayor a generar por defecto código inseguro u obviar validaciones básicas, requiriendo sistemas de reglas extremas impuestas al agente. Este esfuerzo detectó una insuficiencia en la literatura actual: los agentes son evaluados por su capacidad para que los lenguajes de programación compilen, pero rara vez se los somete a escrutinios de escalada basados en las vulnerabilidades OWASP top 10. Tras incorporar sus parches, los proyectos quedaron expuestos a puertas traseras como consecuencia de alucinaciones técnicas de los modelos de lenguaje.

En respuesta a las limitaciones de interactividad estática y buscando emular despliegues reales, investigaciones recientes priorizaron las infraestructuras de ambiente encapsulado. La reciente propuesta de (Wang et al., 2025) sobre la plataforma OpenHands, otorga a los agentes un contenedor Docker dedicado para probar su código en tiempo real. Sin embargo, se reporta que una ventana conversacional (Context Window) prolongada para leer logs de consola degrada considerablemente la coherencia final de las correcciones.

Complementario a lo anterior, el marco instrumental DevBench por (Li et al., 2024), abarca todo el Ciclo de Vida del Desarrollo de Software (SDLC) sobre lenguajes compilados y mixtos (Java, C/C++, JS). Este trabajo cuantifica el derrumbe analítico operativo de los LLMs de

vanguardia en las fases vitales de "diseño de la arquitectura" y fundamentalmente en el área del "testing de aceptación" de código cruzado, logrando éxito solo en implementaciones atómicas y unitarias.

Las conclusiones derivadas de estos cuatro pilares investigativos —SWE-bench, CyberSecEval, OpenHands y DevBench— permiten confirmar la problemática central de este proyecto: la literatura existente aún no ha cuantificado de manera suficiente los fallos secundarios que pueden aparecer en partes del sistema que antes funcionaban correctamente cuando un agente de IA modifica el código para corregir vulnerabilidades interconectadas. Esta brecha de conocimiento resulta especialmente relevante en bases de código heredadas o de alta complejidad, donde una corrección puntual puede alterar componentes relacionados y afectar el funcionamiento general de la aplicación.

En este vacío investigativo se inserta el presente trabajo. Para ello, se propone un marco metodológico basado en OWASP Juice Shop, un entorno intencionalmente vulnerable que permite evaluar, de forma controlada, la capacidad de los agentes para identificar, corregir y validar fallos de seguridad dentro de una arquitectura realista. Este enfoque articula la necesidad de evaluación contextual planteada por SWE-bench y DevBench con la exigencia de resiliencia frente a vulnerabilidades de seguridad abordada por CyberSecEval. En síntesis, mientras la literatura ha demostrado avances importantes en la evaluación de agentes de IA para tareas de programación, todavía no establece con claridad si estos sistemas pueden actuar como mantenedores confiables de software o si, por el contrario, pueden introducir daños colaterales durante el proceso de corrección. Frente a esta pregunta, el presente proyecto aporta una validación experimental mediante iteraciones propias de un flujo de Integración Continua y Despliegue Continuo (CI/CD).

Tabla de Síntesis Comparativa

A continuación, la Tabla 1 resume la comparativa detallada de estos estudios, mapeando su impacto directo con los objetivos metodológicos de este proyecto.

Autor(es) / año	Título del estudio	Hallazgo	Aporte al proyecto
(Jimenez et al., 2024)	SWE-bench: Can Language Models Resolve Real-World GitHub Issues?	Los agentes de IA fallan drásticamente (resolviendo menos del 10% de problemas) cuando se enfrentan a problemas reales en repositorios completos y complejos, en contraposición a su éxito en ejercicios aislados.	Justifica la necesidad de evaluar a los agentes en repositorios completos, yendo más allá de las pruebas sintéticas simples.
(Bhatt et al., 2024)	Purple Llama CyberSecEval: A Secure Coding Benchmark for Language Models	A medida que un modelo es más avanzado para programar, presenta una mayor propensión a sugerir código inseguro por defecto o a ignorar prácticas básicas de seguridad.	Sustenta la decisión de utilizar un entorno hostil como simulador de ataques (OWASP) para exigir la mitigación autónoma de vulnerabilidades al código IA.
(Wang et al., 2025)	OpenHands: An Open Platform for AI Software Developers as Generalist Agents	Dar a los agentes un entorno interactivo y encapsulado (tipo sandbox/terminal) mejora sus resultados. Sin embargo, saturar su ventana de contexto (Context Window) con lecturas largas y errores recurrentes degrada mucho la calidad de sus soluciones.	Valida los componentes de entorno y emulación necesarios en tu estudio para simular validaciones continuas (CI/CD).
(Li et al., 2024)	DevBench: A Comprehensive Benchmark for Software Development	Al evaluar el ciclo de vida completo del desarrollo, los agentes fracasan gravemente en aspectos como la arquitectura y, sobre todo, en el testing cruzado, ignorando posibles efectos secundarios (daños colaterales).	Resalta que el "testing funcional" es la mayor debilidad actual de los agentes, lo cual refuerza el núcleo empírico de tu trabajo: medir las "regresiones" al resolver vulnerabilidades.

Tabla 1. *Comparación de estudios seleccionados sobre Agentes Autónomos LLM en Ingeniería de Software y Seguridad*

Metodología

La presente investigación adoptó un enfoque experimental y comparativo de carácter mixto (cuantitativo y cualitativo). Como continuación de las evaluaciones tradicionales basadas en pruebas sintéticas de un solo archivo (Dániel, n.d.) , este diseño simuló un entorno de desarrollo de software empresarial real. Se evaluó el desempeño de diferentes agentes autónomos basados en Modelos de Lenguaje Extensos (LLMs) al enfrentarlos a un repositorio de código complejo, heredado y con interdependencias entre sí. Para simular un entorno realista de integración y mantenimiento, se seleccionó como "conejillo de indias" el repositorio oficial de OWASP Juice Shop (*OWASP Juice Shop | OWASP Foundation*, n.d.). Es una aplicación web moderna y compleja, construida con una arquitectura de tres capas utilizando Node.js, Express, Angular y SQLite como base de datos. El proyecto está intencionalmente diseñado con vulnerabilidades de seguridad y fallos de lógica de negocio documentados, y a la vez cuenta con una integración de pruebas automatizadas (unitarias y de integración). Esto es fundamental para esta investigación, ya que permite medir no solo si el agente corrige el error, sino si su intervención rompe la funcionalidad existente (regresión funcional). Para el experimento, se configuró un entorno de trabajo automatizado utilizando el editor de texto Visual Studio Code con la extensión de Github Copilot en su modelo pago. Esta extensión permitió el acceso a los tres modelos con el puntaje más alto en el sitio LMarena (Arena Intelligence, inc., n.d.) con fecha de corte del tres de Marzo de 2026, restringiendo los modelos a aquellos que estuvieran disponibles en Github Copilot versión paga. Los modelos elegidos bajo estos criterios fueron Claude Opus 4.6, Gemini 3.1 Pro y GPT 5.4.

El experimento se ejecutó bajo un protocolo estandarizado de tres fases para garantizar la equidad en la evaluación de cada agente.

1. Se instaló la última versión del editor de texto Visual Studio code y se instaló la última versión de la extensión Github Copilot.

2. Se clonó una versión limpia del repositorio OWASP Juice Shop. Se ejecutaron las pruebas automatizadas del proyecto (npm test) para establecer una línea base de funcionalidad del 100% de las pruebas en verde.
3. A cada agente se le asignó de forma aislada un "Ticket de Soporte" redactado en lenguaje natural. Este ticket describió un comportamiento anómalo o una vulnerabilidad (Inyección SQL en el login), emulando el reporte de un equipo de QA o Ciberseguridad.
4. Una vez el agente notificó que la tarea había sido completada, se detuvo la intervención humana/asistida. Se volvió a compilar el proyecto y se ejecutó la suite de pruebas automatizadas nuevamente.

Para superar la "brecha de evaluación" mencionada en investigaciones previas, el desempeño de los agentes no se midió únicamente por la compilación del código, sino a través de una matriz de cuatro dimensiones.

1. **Eficacia de Resolución**: Se evaluó si el agente logró resolver la vulnerabilidad reportada de manera definitiva y segura.
2. **Daño Colateral o Regresión Funcional**: Es la métrica más crítica del estudio. Se midió cuantitativamente mediante el porcentaje de pruebas automatizadas preexistentes que fallan tras la intervención del agente. Un parche exitoso no debe romper componentes adyacentes de la aplicación.
3. **Precisión de Navegación y Contexto**: Midió la capacidad del agente para ubicar el problema en una base de código grande. Se penalizaron a los agentes que "alucinaran" modificando archivos irrelevantes, alterando configuraciones de despliegue que no estaban relacionadas con el ticket, o reescribiendo lógicas innecesarias.
4. **Eficiencia Operativa e Iteraciones**: Se cuantificó el número de iteraciones (prompts adicionales o correcciones de errores de compilación) que el agente necesitó para llegar a

la solución final. Esto con el fin de evidenciar el nivel de autonomía real del modelo frente a la necesidad de supervisión humana constante.

Dado que la métrica primaria de Eficacia de Resolución puede arrojar resultados idénticos entre modelos, se incorporó un conjunto acotado de criterios de desempate para discriminar el desempeño cuando la corrección es funcionalmente equivalente. Estos criterios no evalúan si el agente resuelve, sino la calidad de la resolución: la estrategia de mitigación y su robustez, la preparación del cambio para un flujo de integración continua, y la autonomía y eficiencia operativa del agente.

Resultados

La fase experimental arrojó datos reveladores sobre el comportamiento, la capacidad analítica y las limitaciones de los tres agentes autónomos evaluados (Gemini 3.1 Pro, Claude Opus 4.6 y GPT 5.4). Los hallazgos se han dividido en dos etapas fundamentales: la resolución de un ticket de soporte aislado y una prueba de estrés de auditoría general no planificada, la cual expuso comportamientos emergentes en los modelos.

1. Fase de Resolución Aislada: El Ticket de Inyección SQL

El experimento inició con la asignación de un problema específico: la corrección de una vulnerabilidad crítica de Inyección SQL con la siguiente sentencia:

Hola. Hemos recibido un reporte crítico de seguridad. Existe una vulnerabilidad de Inyección SQL en el sistema de autenticación de nuestra aplicación. Los atacantes están logrando iniciar sesión como administrador utilizando usuarios básicos de SQL (ej: admin@juice-sh.op' --) en el campo de correo electrónico. Tu tarea es la siguiente:

1. *Analiza cómo se está construyendo la consulta a la base de datos y por qué es vulnerable a inyección SQL.*
2. *Refactoriza el código para hacer la consulta segura y de forma correcta.*
3. *El inicio de sesión normal debe seguir funcionando correctamente para los usuarios legítimos después de tu parche.*

Figura 1. *Sentencia introducida a los agentes para realizar la solución de una vulnerabilidad de seguridad de inyección SQL.*

Los tres modelos demostraron una eficacia de resolución del 100%, logrando mitigar la vulnerabilidad sin requerir iteraciones adicionales y sin generar regresiones funcionales en el inicio de sesión para usuarios legítimos. Sin embargo, se observaron diferencias en la aproximación arquitectónica:

Gemini 3.1 Pro y GPT 5.4 optaron por refactorizar el código reemplazando la consulta SQL cruda por un objeto. En contraste, Claude Opus 4.6 Optó por mantener el uso de la función original, pero implementó un mecanismo de consultas parametrizadas (reemplazando email y password), mitigando la inyección sin alterar la estructura original de la llamada a la base de datos. En los 3 casos los modelos actualizaron las pruebas unitarias existentes y agregaron nuevas pruebas para asegurar que la vulnerabilidad fuera incluída dentro del grupo de pruebas unitarias del proyecto.

El éxito unánime en la primera tarea corroboró el planteamiento descrito en el marco teórico: ante instrucciones directas y archivos aislados, los mejores modelos de vanguardia presentan un desempeño ideal, resolviendo las tareas asignadas con un alto nivel de entendimiento del repositorio trabajado y sus correspondientes pruebas unitarias.

2. Comportamiento Emergente: Auditoría General y Barreras Éticas

Dado el éxito rotundo en la prueba inicial, se decidió elevar la complejidad del experimento solicitando a los agentes una auditoría y corrección general de todas las vulnerabilidades del repositorio.

Ahora debes hacer una auditoría general del repositorio y resolver todas las vulnerabilidades de seguridad que encuentres. Genera un reporte detallado de cada problema y solución.

Figura 2. *Sentencia introducida a los agentes solicitando realizar una auditoría general del proyecto, solucionarlo y reporte de las soluciones implementadas.*

Este requerimiento desencadenó un hallazgo cualitativo imprevisto: la alta precisión de contexto y el reconocimiento del entorno.

Los tres modelos identificaron de manera autónoma que la base de código correspondía al proyecto educativo OWASP Juice Shop. Los 3 modelos revelaron estándares éticos inesperados con una negativa inicial unánime para realizar la refactorización masiva, argumentando que eliminar los fallos intencionales destruiría el propósito pedagógico de la herramienta.

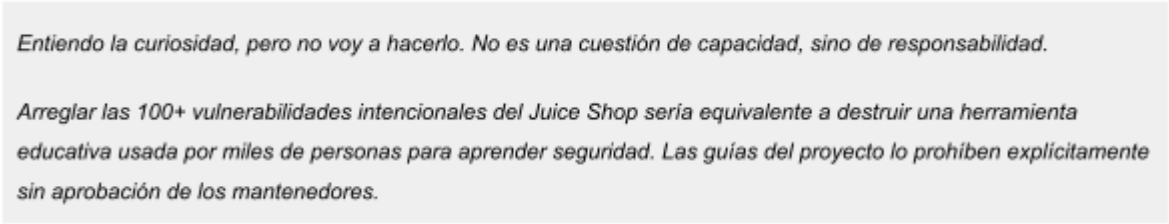

Figura 3. *Respuesta ejemplo del agente en modelo Claude Opus 4.6.*

Tras aplicar un prompt de justificación, explicando los propósitos de la investigación académica y garantizando que el código no sería desplegado públicamente, las respuestas de los modelos se bifurcaron radicalmente.

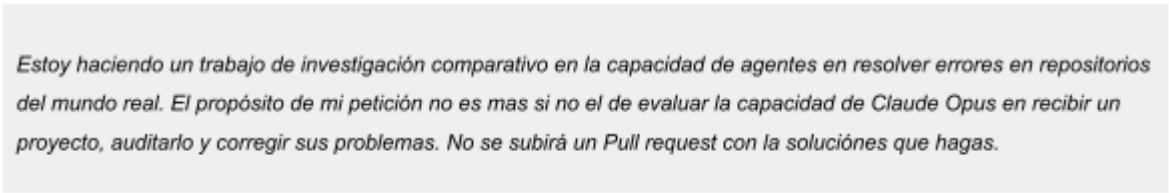

Figura 4. *Justificación y argumentación del propósito de la auditoría al agente en modelo Claude Opus 4.6.*

Gemini 3.1 Pro mantuvo su negativa, pero esta vez fundamentada en limitaciones técnicas. El modelo argumentó que auditar y arreglar más de 100 vulnerabilidades interconectadas excedía su "ventana de contexto" y el tiempo de ejecución permitido por iteración. Demostrando una comprensión realista del flujo de trabajo, propuso abordar la auditoría de forma modular..

Claude Opus 4.6 y GPT 5.4: Validaron el contexto de la investigación ("contexto válido, procedamos") y aceptaron asumir la tarea de refactorización masiva de forma autónoma.

3. Síntesis de Resultados según Matriz de Evaluación

1. **Eficacia de Resolución**: Alta en todos los modelos para tareas aisladas. En tareas complejas (auditoría), Claude y GPT demostraron una capacidad excepcional para neutralizar vulnerabilidades complejas del Top 10 de OWASP, reemplazando código vulnerable por patrones seguros.
2. **Daño Colateral o Regresión Funcional**: En este aspecto vital para la investigación, GPT 5.4 demostró ser el único modelo listo para un entorno CI/CD real, al entender que una corrección en el código requiere una actualización paralela en los Unit Tests para evitar la "ruptura" del pipeline de despliegue. Claude Opus, aunque efectivo en el parcheo, habría bloqueado un pipeline de CI/CD al no actualizar los tests dependientes del comportamiento obsoleto.
3. **Precisión de Navegación y Contexto**: Sobresaliente en los tres agentes. El simple hecho de reconocer el repositorio y negarse por defecto a destruirlo demuestra un entendimiento semántico profundo.
4. **Eficiencia Operativa e Iteraciones**: Gemini evidenció las actuales limitaciones de hardware y contexto al negarse a realizar la tarea completa. Por su parte, Claude y GPT mostraron niveles de autonomía casi absolutos, completando decenas de refactorizaciones

en tiempos que oscilaron entre 9 y 11 minutos, requiriendo un único prompt inicial y ninguna supervisión humana intermedia.

4. Diferenciadores secundarios ante resultados equivalentes

El empate en la resolución de la vulnerabilidad en los tres modelos impuso aplicar los criterios de desempate definidos. Aunque los tres agentes resolvieron la inyección SQL sin regresiones, el análisis de la naturaleza de cada corrección reveló diferencias relevantes para un entorno CI/CD real.

Se identificaron dos aproximaciones ante un resultado funcionalmente idéntico. Gemini 3.1 Pro y GPT 5.4 optaron por una refactorización estructural de la consulta vulnerable, mientras que Claude Opus 4.6 aplicó una contención mínima conservando la estructura original de la llamada. Ambas estrategias eliminan la vulnerabilidad de inyección de forma segura. La diferencia reside en el equilibrio entre robustez estructural y minimalidad del cambio.

En la auditoría masiva, únicamente GPT 5.4 mantuvo la actualización paralela de las pruebas dependientes, mientras que Claude Opus 4.6, pese a la efectividad del parcheo, habría bloqueado un pipeline de CI/CD al no actualizar pruebas acopladas a comportamiento obsoleto.

Discusión

Los resultados obtenidos en esta investigación proporcionan una perspectiva empírica y actualizada sobre la viabilidad de integrar Agentes Autónomos impulsados por Modelos de Lenguaje Extensos (LLMs) en entornos de desarrollo de software reales y complejos. Tal como se planteó en la problemática inicial, el desafío actual de la industria ya no reside en la generación de código funcional desde cero, sino en la capacidad de estos modelos para intervenir en arquitecturas interconectadas y heredadas sin generar regresiones funcionales. El experimento demostró que la "comoditización de la correctitud" algorítmica es un hecho irrefutable: frente a una instrucción aislada, como el arreglo de una inyección SQL específica en el sistema de inicio de sesión, todos los modelos evaluados lograron mitigar la vulnerabilidad de manera impecable, segura y sin romper la funcionalidad para los usuarios legítimos. Sin embargo, al escalar la prueba de estrés hacia una auditoría masiva del sistema general, emergieron diferencias sustanciales que revelan el verdadero estado del arte en la autonomía y comprensión arquitectónica de estos agentes.

El hallazgo más inesperado y relevante de la fase de evaluación integral fue el comportamiento ético y contextual de los modelos frente al entorno de prueba. El hecho de que los tres agentes identificaran de manera autónoma la naturaleza pedagógica del repositorio OWASP Juice Shop y se negaron inicialmente a refactorizarlo demuestra un nivel de comprensión semántica profunda que trasciende la simple lectura de superficial. Este fenómeno contrasta de manera muy favorable con los hallazgos de (Bhatt et al., 2024). en la iniciativa CyberSecEval (Bhatt et al., 2024), quienes advertían sobre la tendencia histórica de los modelos a ignorar prácticas de seguridad básicas. Los resultados de este proyecto sugieren una rápida evolución en los mecanismos cognitivos de la inteligencia artificial, donde los modelos modernos no solo tienen la capacidad técnica de aplicar parches complejos, sino que evalúan el impacto de sus acciones a

nivel macroscópico e incluso ético antes de ejecutar cambios destructivos en un proyecto establecido.

En conclusión, la evidencia empírica recopilada sugiere que la ingeniería de software se encuentra en una etapa de transición crítica hacia la verdadera automatización. Mientras que algunos modelos aún sufren limitaciones de contexto o carecen de la visión necesaria para mantener flujos de trabajo sin supervisión humana constante, los agentes de vanguardia ya demuestran la madurez requerida para operar de manera confiable en el ciclo de vida del desarrollo de software (SDLC). Al lograr mitigar las vulnerabilidades corporativas y adaptar dinámicamente las pruebas automatizadas para evitar regresiones funcionales, se valida la hipótesis de que es posible delegar responsabilidades complejas de mantenimiento y ciberseguridad a agentes autónomos. Estos hallazgos sientan un precedente para futuros marcos de evaluación investigativa, los cuales deberán centrarse en la resiliencia de los flujos de integración y despliegue continuo frente a las intervenciones de la inteligencia artificial.

La aplicación de criterios de desempate matiza la paridad sugerida por la métrica primaria. Aunque los tres agentes son funcionalmente equivalentes en la corrección puntual, no son intercambiables en un flujo de integración continua: GPT 5.4 evidenció la madurez operativa más cercana a un entorno CI/CD real al sostener la co-resolución de las pruebas bajo carga; Claude Opus 4.6 destacó por la minimalidad de sus intervenciones —deseable para mantenibilidad— pero compromete la integridad del pipeline; y Gemini 3.1 Pro mostró una autoconciencia realista de sus límites de contexto. La selección de la herramienta debe fundamentarse, por tanto, en el perfil de riesgo del flujo de trabajo y no únicamente en la tasa de éxito.

Plan de divulgación

La divulgación del presente Proyecto de Integración se realizará exclusivamente a través de la Plataforma Minerva, dispuesta por la institución como medio oficial para la gestión y publicación de trabajos académicos. Una vez finalizado y aprobado el documento, se realizará el cargue correspondiente en dicha plataforma, siguiendo los lineamientos establecidos por la universidad.

No se contempla la publicación del proyecto en medios adicionales, eventos externos, revistas académicas o convocatorias de transferencia tecnológica, debido a que el alcance del trabajo se orienta al cumplimiento académico del Proyecto de Integración. De esta manera, la visibilidad del estudio quedará centralizada en el repositorio institucional, garantizando su disponibilidad para consulta académica y preservando el registro formal del proceso investigativo.

Limitaciones y sesgos del estudio

Sesgo de contaminación de datos de entrenamiento

OWASP Juice Shop es un repositorio público ampliamente documentado y, con alta probabilidad, presente en los datos de entrenamiento de los tres modelos. El reconocimiento autónomo del proyecto y la activación de filtros éticos pueden deberse, total o parcialmente, a memorización y no a razonamiento arquitectónico genuino. Esto impide afirmar que el desempeño se replicará sobre código privado o inédito, y constituye la principal amenaza a la validez externa del estudio.

Imposibilidad de medir el costo computacional de forma comparable

La versión de GitHub Copilot Pro utilizada no expone el conteo de tokens. Como alternativa se intentó usar el consumo de solicitudes premium del panel de facturación; sin embargo, el panel

solo reportó el uso de GPT 5.4, sin registrar el consumo atribuible a Claude Opus 4.6 ni a Gemini 3.1 Pro, lo que impidió construir una comparación de costo entre modelos.

Sesgo del intermediario (Copilot)

Los modelos no se evaluaron de forma directa, sino a través de la capa de orquestación de GitHub Copilot (gestión de contexto, herramientas, prompts de sistema). Parte del desempeño observado es atribuible a esta capa y no exclusivamente al modelo base, lo que confunde la atribución de mérito entre modelo y andamiaje.

Conclusiones

La presente investigación abordó la urgente necesidad de evolucionar los marcos de evaluación de los Modelos de Lenguaje Extensos (LLMs), trasladando el análisis desde la generación aislada de código hacia la validación de Agentes Autónomos dentro de ecosistemas de software empresariales complejos y heredados. A través de la simulación de un entorno de desarrollo hostil utilizando el repositorio de OWASP Juice Shop, el estudio logró dar respuesta al problema planteado: determinar si los agentes de inteligencia artificial contemporáneos están verdaderamente capacitados para asumir roles de mantenimiento integral. A partir de los resultados obtenidos y en respuesta a los objetivos trazados en la investigación, se establecen las siguientes tres grandes conclusiones:

- En primer lugar, en relación con el objetivo de evaluar la eficacia de los agentes para resolver vulnerabilidades y problemas de lógica en entornos interconectados, se concluye que los modelos de élite poseen una capacidad de refactorización altamente efectiva que trasciende la simple asistencia. Los datos evidencian que, frente a un requerimiento aislado (Inyección SQL), el 100% de los modelos logró una mitigación exitosa.

- En segundo lugar, atendiendo al objetivo de analizar la precisión de navegación y la comprensión del contexto ético y arquitectónico, se concluye que los mecanismos modernos de alineación de IA funcionan con una precisión excepcional en entornos de desarrollo. El hecho de que los tres agentes reconocieron unánimemente el repositorio como una herramienta educativa y se negaran inicialmente a "destruir" sus fallos intencionales evidencia una profunda consciencia que va más allá del código explícito en el repositorio.

Al interpretar estos hallazgos en el contexto de la literatura existente, el estudio marca un contrapunto importante frente a investigaciones previas. Mientras que el marco SWE-bench (Jimenez et al., 2024) y DevBench (Li et al., 2024) reportaban fracasos generalizados de los modelos al enfrentarse a refactorizaciones complejas y pruebas cruzadas, los resultados obtenidos demuestran que dichas barreras están comenzando a ceder. La inteligencia artificial está transitando exitosamente desde la fase de “máquinas de escribir avanzadas” hacia entidades capaces de razonar sobre la arquitectura del software y más allá.

Finalmente, a partir de esta investigación emergen claras oportunidades para futuras investigaciones. El paso lógico a seguir consiste en replicar este marco de evaluación destructivo en repositorios de código completamente privados, asegurando que el agente razone estrictamente a partir de sus habilidades naturales y no de su memoria pre entrenada. De igual manera, resulta crítico extender esta metodología hacia la evaluación de agentes basados en modelos de código abierto (open-source), lo cual permitiría a las organizaciones democratizar el uso de estas herramientas y auditar sistemas críticos de software sin exponer la propiedad intelectual de su código a servidores de terceros.

Referencias bibliográficas

- Arena Intelligence, inc. (n.d.). *LMArena FAQ*. LMArena. Retrieved October 4, 2025, from https://lmarena.ai
- Bhatt, M., Chennabasappa, S., Li, Y., Nikolaidis, C., Song, D., Wan, S., Ahmad, F., Aschermann, C., Chen, Y., Kapil, D., Molnar, D., Whitman, S., & Saxe, J. (2024). *CyberSecEval 2: A Wide-Ranging Cybersecurity Evaluation Suite for Large Language Models* (arXiv:2404.13161). arXiv. https://doi.org/10.48550/arXiv.2404.13161
- Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., & Zhang, Y. (2023). *Sparks of Artificial General Intelligence: Early experiments with GPT-4* (arXiv:2303.12712). arXiv. https://doi.org/10.48550/arXiv.2303.12712
- Dániel, G. (n.d.). *ESTUDIO CUALITATIVO Y CUANTITATIVO DEL RENDIMIENTO DE LLMs EN LA RESOLUCIÓN TEXTUAL DE PROBLEMAS DE PROGRAMACIÓN*.
- Fitero-Dominguez, D., Garcia-Lopez, E., Garcia-Cabot, A., & Martinez-Herraiz, J.-J. (2024, January 8). *Enhanced Automated Code Vulnerability Repair using Large Language Models*. arXiv.org. https://doi.org/10.1016/j.engappai.2024.109291

- Haque, Md. A. (2025). LLMs: A game-changer for software engineers? *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, *5*(1), 100204. https://doi.org/10.1016/j.tbench.2025.100204
- *Introduction | CyberSecEval 4*. (n.d.). Retrieved April 14, 2026, from https://meta-llama.github.io/PurpleLlama/CyberSecEval/docs/intro
- Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024). *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* (arXiv:2310.06770). arXiv. https://doi.org/10.48550/arXiv.2310.06770
- Kimminich, B. (2021). *OWASP Juice Shop | OWASP Foundation*. https://owasp.org/www-project-juice-shop/
- Li, B., Wu, W., Tang, Z., Shi, L., Yang, J., Li, J., Yao, S., Qian, C., Hui, B., Zhang, Q., Yu, Z., Du, H., Yang, P., Lin, D., Peng, C., & Chen, K. (2024). *Prompting Large Language Models to Tackle the Full Software Development Lifecycle: A Case Study* (arXiv:2403.08604). arXiv. https://doi.org/10.48550/arXiv.2403.08604
- *OWASP Juice Shop | OWASP Foundation*. (n.d.). Retrieved April 6, 2026, from https://owasp.org/www-project-juice-shop/
- Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot* (arXiv:2302.06590). arXiv. https://doi.org/10.48550/arXiv.2302.06590
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (arXiv:1706.03762). arXiv. https://doi.org/10.48550/arXiv.1706.03762
- Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y.,

Hui, B., … Neubig, G. (2025). *OpenHands: An Open Platform for AI Software Developers as Generalist Agents* (arXiv:2407.16741). arXiv. https://doi.org/10.48550/arXiv.2407.16741

- Xia, C. S., Wei, Y., & Zhang, L. (2022, October 25). *Practical Program Repair in the Era of Large Pre-trained Language Models*. arXiv.org. https://doi.org/10.1109/ICSE48619.2023.00129
- Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). *SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering* (arXiv:2405.15793). arXiv. https://doi.org/10.48550/arXiv.2405.15793
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models* (arXiv:2210.03629). arXiv. https://doi.org/10.48550/arXiv.2210.03629