

CHATGPT: PRECISIÓN Y LIMITACIONES EN OPERACIONES
MATEMÁTICAS
CHATGPT: ACCURACY AND LIMITATIONS IN MATHEMATICAL
OPERATIONS



TRABAJO FIN DE GRADO
CURSO 2023-2024

AUTORES
IGNACIO ÁLVAREZ CASO,
CÉSAR AMOR NÚÑEZ,
JULIÁN ENCINAS FUENTES,
EDUARDO RODELGO GUZMÁN

DIRECTOR
PABLO RABANAL BASALO

GRADO EN ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS-INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

CHATGPT: PRECISIÓN Y LIMITACIONES EN OPERACIONES
MATEMÁTICAS
CHATGPT: ACCURACY AND LIMITATIONS IN MATHEMATICAL
OPERATIONS

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTORES
IGNACIO ÁLVAREZ CASO,
CÉSAR AMOR NÚÑEZ,
JULIÁN ENCINAS FUENTES,
EDUARDO RODELGO GUZMÁN

DIRECTOR
PABLO RABANAL BASALO

CONVOCATORIA: JUNIO 2024

GRADO EN ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS-INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

27 DE MAYO DE 2024

Resumen

ChatGPT: Precisión y limitaciones en operaciones matemáticas

En este trabajo de fin de grado, se investiga la capacidad y los límites de ChatGPT para resolver operaciones matemáticas. La investigación se realiza, por un lado, a través de una pregunta directa a la inteligencia artificial y, por otro, mediante la petición de ejecución de unos casos de prueba en un código de programación que previamente ChatGPT ha proporcionado. De esta manera, al tratarse de un sistema de procesamiento del lenguaje natural, se determinan cuáles son las características de esta inteligencia que derivan en un resultado correcto o erróneo en el cálculo de una operación matemática. Para ello, se desarrolla una aplicación web con las interfaces necesarias para realizar pruebas, y con una sección estadística donde visualizar de forma gráfica los resultados. Esta se implementa bajo el *framework* de Django, elegido por su compatibilidad con el lenguaje Python, en el que se ejecuta la aplicación, por su modularidad y escalabilidad.

Palabras clave

ChatGPT, matemáticas, programación, operación, aplicación web, inteligencia artificial, limitaciones.

Abstract

ChatGPT: Accuracy and limitations in mathematical operations

In this Final Degree work, we investigate the capacity and limits of ChatGPT to solve mathematical operations. The research is done by direct question to the AI and through the request of execution of some test cases in a programming code that ChatGPT has previously provided. In this way, being a natural language processing system, the characteristics that lead ChatGPT to give the correct result or to make a mistake in the calculation of a mathematical operation are determined. For this purpose, a web application is developed with the necessary interfaces to perform tests, and with a statistical section where the results can be graphically displayed. The application is developed under the Django framework, chosen for its compatibility with the Python language, in which the application is developed, for its modularity and scalability.

Keywords

ChatGPT, mathematics, programming, operation, web application, artificial intelligence, limitations.

Índice de contenidos

| | |
|---|-----------|
| Resumen..... | 3 |
| Abstract..... | 4 |
| Índice de contenidos..... | 5 |
| Índice de imágenes..... | 7 |
| Índice de gráficos..... | 8 |
| Capítulo 1 - Introducción..... | 9 |
| 1.1. Motivación..... | 9 |
| 1.2. Objetivos..... | 9 |
| 1.3. Metodología..... | 10 |
| Capítulo 2 - Marco Teórico..... | 12 |
| Capítulo 3 - Tecnologías..... | 15 |
| 3.1. Tecnologías utilizadas..... | 15 |
| 3.1.1. Django..... | 15 |
| 3.1.2. ChatGPT..... | 17 |
| 3.1.3. SymPy..... | 19 |
| 3.1.4. Matplotlib y Seaborn..... | 19 |
| 3.1.5. JDoodle..... | 20 |
| 3.1.6. SQLite..... | 21 |
| 3.2. Tecnologías desechadas..... | 22 |
| Capítulo 4 - Arquitectura..... | 24 |
| 4.1. Estructura general..... | 24 |
| 4.1.1. El controlador..... | 35 |
| 4.2. Resolución de problemas matemáticos..... | 36 |
| 4.2.1. SymPy..... | 37 |
| 4.2.2. El comparador..... | 37 |
| 4.3. Generación de código..... | 38 |
| 4.3.1. JDoodle..... | 39 |
| 4.4. Gráficos..... | 40 |
| 4.4.1. Tipos de gráficos utilizados..... | 41 |
| 4.4.2. Funciones principales..... | 42 |

| | |
|---------------------------------------|-----------|
| 4.5. Base de datos..... | 44 |
| Capítulo 5 - Resultados..... | 45 |
| 5.1. La aplicación..... | 45 |
| 5.2. Principales hallazgos..... | 50 |
| 5.3. Limitaciones del estudio..... | 63 |
| Capítulo 6 - Conclusiones..... | 65 |
| 6.1. Trabajo a futuro..... | 67 |
| Chapter 6 - Conclusions..... | 68 |
| Contribuciones Personales..... | 70 |
| Ignacio Álvarez Caso..... | 70 |
| César Amor Núñez..... | 72 |
| Julián Encinas Fuentes..... | 74 |
| Eduardo José Rodelgo Guzmán..... | 76 |
| Bibliografía..... | 78 |

Índice de imágenes

| | |
|--|----|
| Imagen 1. Logo de OpenAI..... | 12 |
| Imagen 2. Model Template View. Características y funcionalidad del Modelo Vista Controlador MTV (Juan Díaz, s.f.)..... | 17 |
| Imagen 4. Ejemplos del uso de Matplotlib..... | 20 |
| Imagen 5. Logo de JDoodle..... | 21 |
| Imagen 6. Logo de Reflex..... | 22 |
| Imagen 7. Función manejo de formularios de las operaciones..... | 28 |
| Imagen 8. Función para manejar el formulario de código..... | 28 |
| Imagen 9. Función encargada de añadir nuevos datos a la tabla..... | 29 |
| Imagen 10. Función para ver los datos de la tabla..... | 29 |
| Imagen 11. Función estadística..... | 30 |
| Imagen 12. Formulario base común a todas las funciones..... | 32 |
| Imagen 13. Función formulario integral..... | 33 |
| Imagen 14. Función para limpiar y validar los límites..... | 33 |
| Imagen 15. Función para la petición matemática a ChatGPT..... | 36 |
| Imagen 16. Función para la petición de código a ChatGPT..... | 39 |
| Imagen 17. Función petición API JDoodle..... | 40 |
| Imagen 18. Función crear gráfico correlación de operaciones simples..... | 43 |
| Imagen 19. Función mostrar gráfico correlación operaciones simples..... | 43 |
| Imagen 20. Página de inicio de la aplicación..... | 45 |
| Imagen 21. Página de la operación matemática Límite..... | 46 |
| Imagen 22. Mensaje de error de una operación matemática..... | 47 |
| Imagen 23. Pantalla de resultados de una operación matemática..... | 47 |
| Imagen 24. Pantalla de petición de un código a ChatGPT..... | 48 |
| Imagen 25. Resultados de un programa solicitado a ChatGPT..... | 49 |
| Imagen 26. Acierto de ChatGPT en una operación..... | 60 |
| Imagen 27. Fallo de ChatGPT en una operación..... | 61 |
| Imagen 28. Petición de un programa con argumentos de entrada..... | 62 |
| Imagen 29. Solución a la petición de la imagen 28..... | 63 |

Índice de gráficos

| | |
|--|----|
| Gráfico 1. Correlación entre variables y acierto del conjunto de resultados..... | 51 |
| Gráfico 2. Correlación entre variables y acierto de las operaciones simples..... | 52 |
| Gráfico 3. Correlación entre variables y acierto de las ecuaciones..... | 53 |
| Gráfico 4. Correlación entre variables y acierto de integrales..... | 54 |
| Gráfico 5. Posibilidad de acierto de integrales definidas y no definidas..... | 54 |
| Gráfico 6. Correlación entre variables y acierto de derivadas..... | 55 |
| Gráfico 7. Posibilidad de acierto según el orden de la derivada..... | 56 |
| Gráfico 9. Histograma de tasa de acierto y fallo de las operaciones simples..... | 57 |
| Gráfico 10. Histograma de tasa de acierto y fallo de las ecuaciones..... | 58 |
| Gráfico 11. Histograma de tasa de acierto y fallo de las integrales..... | 58 |
| Gráfico 12. Histograma de tasa de acierto y fallo de las derivadas..... | 59 |
| Gráfico 13. Histograma de tasa de acierto y fallo de los límites..... | 59 |

Capítulo 1 - Introducción

1.1. Motivación

En la sociedad actual el término inteligencia artificial (IA) ya no es algo solo conocido por aquellas personas más cercanas al sector de la informática. Los sistemas de procesamiento del lenguaje natural (PLN) gozan de un nivel de sofisticación cada vez mayor y son capaces de ayudar en tareas más complejas. Dentro de este ámbito, ChatGPT emerge como una herramienta prominente que utiliza la tecnología de generación de lenguaje para interactuar con los usuarios de manera fluida y comprensiva (Sergio Parra, 2023) (Antonio Lorenzo, 2023).

La motivación detrás de este trabajo de fin de grado (TFG) se encuentra en definir límites, en descubrir posibilidades y en satisfacer nuestra curiosidad sobre lo que, actualmente, las inteligencias artificiales son capaces de hacer para entendernos, ayudarnos. De esta forma, este trabajo se centra en comprender, en profundidad, el rendimiento de ChatGPT en la resolución de operaciones matemáticas, y explorar de qué forma es capaz de corregir y compilar código en diversos lenguajes de programación.

1.2. Objetivos

El objetivo principal del presente trabajo es documentar aquellos cálculos matemáticos que la inteligencia artificial, ChatGPT, es o no capaz de realizar de manera correcta, analizando qué características influyen para pasar del acierto al error.

Además, se plantean una serie de objetivos en apoyo del primero. Por un lado, mostrar de forma estadística en qué cálculos falla más la herramienta y, por otro lado, comparar si se comporta de forma similar cuando programa un código en un cierto lenguaje para resolver dicha operación mostrando un resultado, en lugar de resolver una operación matemática.

Por último, este TFG tiene también como objetivo aplicar los principales conocimientos que se han ido adquiriendo a lo largo de la carrera de Ingeniería Informática, así como aprender nuevos lenguajes de programación, nociones de inteligencia artificial o el desarrollo de aplicaciones web.

1.3. Metodología

Para lograr la consecuencia de estos objetivos, se implementará una aplicación web sobre la que se pueda navegar de forma intuitiva y se permitan realizar las distintas operaciones donde se muestren, de forma fácilmente interpretable, los resultados.

Para interpretar los datos de los resultados de la investigación, se hará uso del análisis estadístico, tanto dentro como fuera de la aplicación. Por último, para establecer un marco teórico que sirva de soporte al desarrollo del trabajo, se consultarán fuentes de información, primarias y secundarias, con referente bibliográfico acerca de inteligencia artificial, sistemas de procesamiento del lenguaje natural y ChatGPT, tanto por separado como en relación a su desempeño en las operaciones matemáticas.

Para la elaboración del presente trabajo, se ha establecido una planificación del desarrollo comprendida en varias fases. En primer lugar, se realizará la búsqueda de tecnologías con el objetivo de identificar y seleccionar las más adecuadas para el desarrollo de la aplicación. A continuación, se procederá con la implementación de las operaciones matemáticas dentro de la misma, seguido de la parte de código de programación. Para finalizar la aplicación, se creará un módulo que sirva para mostrar los resultados estadísticos. Posteriormente, se creará un banco de pruebas, se ejecutará y se analizarán los resultados obtenidos. La planificación tendrá un seguimiento bisemanal donde se pondrán en común los avances obtenidos y se establecerán nuevos.

Capítulo 2 - Marco Teórico

OpenAI es una empresa centrada en investigación y desarrollo de inteligencia artificial y son los creadores de ChatGPT. Según ellos, su misión es que la inteligencia artificial beneficie a toda la humanidad (OpenAI, s.f. A).



Imagen 1. Logo de OpenAI.

Para definir qué es ChatGPT se ha preguntado a la propia herramienta que se defina a sí misma:

"Soy ChatGPT, un programa de inteligencia artificial desarrollado por OpenAI. No soy una persona real ni me he colado en tu ordenador. Soy un software diseñado para procesar y generar texto de manera similar a cómo lo haría un humano, utilizando modelos de lenguaje entrenados en grandes cantidades de datos. La capacidad para mantener una conversación con sentido se debe a este entrenamiento en datos lingüísticos diversos, lo que me permite comprender y generar respuestas coherentes basadas en el contexto y las preguntas que me haces."

Con el mismo método, el artículo "ChatGPT, una herramienta que no podemos desconocer en nuestra práctica docente", aborda el impacto de la inteligencia artificial en el ámbito educativo. Se destaca la evolución significativa de la inteligencia

artificial en los últimos años, especialmente con la aparición de modelos avanzados del lenguaje como GTP-3, que pueden generar texto coherente aprovechando el vasto conocimiento disponible en Internet. Se reconoce que esta herramienta tiene diversos usos, incluido el ámbito educativo, donde puede tener un impacto tanto positivo como negativo, lo que subraya la necesidad de una reflexión crítica sobre su uso. El artículo propone una reflexión sobre este tema, utilizando la técnica de entrevistar directamente a ChatGPT. Esta experiencia proporciona una visión sobre la avanzada y convincente capacidad de esta IA para generar respuestas, lo que lleva a la conclusión de que, desde la perspectiva docente, no se puede ignorar la existencia de esta herramienta (Leonardo Esnaola, 2023). Sin salir del alcance de la docencia, el estudio de (Tenhundfeld, 2023) presentó un artículo sobre la interacción humano-AI, destacando las fortalezas y limitaciones de ChatGPT en este contexto. Por otro lado, la investigación de (Bin-Hady, 2023) investigó el uso de ChatGPT en la enseñanza del idioma inglés, sugiriendo que puede mejorar las habilidades lingüísticas de los estudiantes.

¿Y si el campo sobre el que le preguntamos es una ciencia exacta, como puede ser el caso de las matemáticas? El estudio titulado "ChatGPT: una herramienta revolucionaria para enseñar y aprender matemáticas" se enfoca en analizar las percepciones de estudiantes y educadores con respecto al uso de inteligencia artificial en la enseñanza de las matemáticas. Para ello, se emplea un enfoque cualitativo que consta de dos etapas: análisis de contenido de entrevistas e investigación de la experiencia del usuario. En la primera etapa, se destaca que ChatGPT es reconocido por sus capacidades mejoradas en matemáticas y su potencial para aumentar el éxito educativo al proporcionar conocimientos básicos de matemáticas. La segunda etapa revela varios problemas, como la falta de comprensión profunda de geometría y la incapacidad para corregir conceptos erróneos de manera efectiva (Yousef Wardat,

2023). De manera similar, el estudio sobre las "Capacidades matemáticas de ChatGPT" (Luca Pinchetti, 2023) investiga el rendimiento en matemáticas de dos versiones de la versión GPT-3 (lanzadas el 9 de enero de 2023 y el 30 de enero de 2023), así como de GPT-4. Se llevaron a cabo pruebas en conjuntos de datos públicos y en conjuntos creados manualmente. Los resultados muestran que esta inteligencia puede ser utilizada con éxito como asistente matemático para consultar hechos, actuando ambas versiones como motores de búsqueda matemáticos e interfaces de bases de conocimiento. Sin embargo, se muestran dificultades en matemáticas de nivel universitario, a pesar de su capacidad para manejar matemáticas de educación secundaria y bachillerato. Se concluye que el rendimiento matemático general de ambos modelos está muy por debajo del nivel de un estudiante de grado.

Por último, y aunque de forma más generalizada, el artículo sobre "Recursos tecnológicos en matemática: transparencia, visibilidad e invisibilidad en la resolución de un problema" (Nicolás Rosbaco, 2023) aborda el uso de herramientas tecnológicas para resolver problemas matemáticos. Se enfoca en analizar un problema desde una perspectiva matemática y evaluar cómo herramientas como wxMaxima, GeoGebra, Scratch y ChatGPT pueden contribuir a su resolución.

Capítulo 3 - Tecnologías

En este capítulo se explican las diversas tecnologías que han sido utilizadas para el desarrollo del trabajo, desde las inicialmente consideradas y luego descartadas hasta las finalmente integradas. En primer lugar, se desarrolla el *framework* en el que se basa toda la aplicación web, Django, seguido de las capacidades de procesamiento del lenguaje natural proporcionadas por ChatGPT. Además, se desglosan también otras tecnologías que han tenido una funcionalidad importante en el proyecto como SymPy para simulaciones, Matplotlib y Seaborn para visualizaciones, y JDoodle para entornos de ejecución de código. Finalmente, se detallan aquellas que se consideraron pero finalmente no se han utilizado.

3.1. Tecnologías utilizadas

3.1.1. Django

Previo a conocer qué es Django es necesario definir qué es un *framework*. Según (Vive, 2022) “un *framework* es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de *software*”.

El objetivo de los *framework* es facilitar a los desarrolladores su labor, obteniendo como resultado trabajos mejor organizados, implementados y evitar así problemas como la falta de consistencia y estandarización. Además, estas estructuras permiten una buena escalabilidad, lo que posibilita trabajar con proyectos más complejos, de diferentes tamaños y, en caso de necesitar expandir y evolucionar los mismos, están diseñados

para soportar actualizaciones según los avances del proyecto. Debido a su carácter modular, los posibles problemas de mantenimiento que se pudiesen producir, se reducen radicalmente al obtener sistemas separados entre sí. De esta forma, permite realizar cambios sobre diferentes partes sin que el resto se vea afectado. Por lo general, cada *framework* tiene su propio diseño y eso conlleva que se encuentren numerosos manuales de uso, documentación, foros y diversos tutoriales que facilitan su aprendizaje y la resolución de dudas o problemas.

Según (Django, s.f.) "Django es un *framework* web Python de alto nivel que fomenta un desarrollo rápido y un diseño limpio y pragmático. Creado por desarrolladores experimentados, se encarga de gran parte de las molestias del desarrollo web, para que puedas concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda. Es gratuito y de código abierto".

La herramienta es descrita como rápida a la hora de realizar desarrollos, segura, extremadamente escalable, versátil (debido a la gran cantidad de usos que se le puede dar) y equipada con una gran cantidad de herramientas a disposición del usuario (Django, s.f. B). Django sigue la estructura del patrón modelo-vista-controlador, derivando al modelo MTV que consigue dividir los proyectos en tres capas bien diferenciadas:

- *Model*: la capa de datos del modelo encargada de gestionar estos datos y obtener información sobre una base de datos.
- *Template*: capa que procesa las solicitudes del cliente mostrando además información al usuario.
- *View*: gestiona las comunicaciones entre la vista y el usuario o cliente.

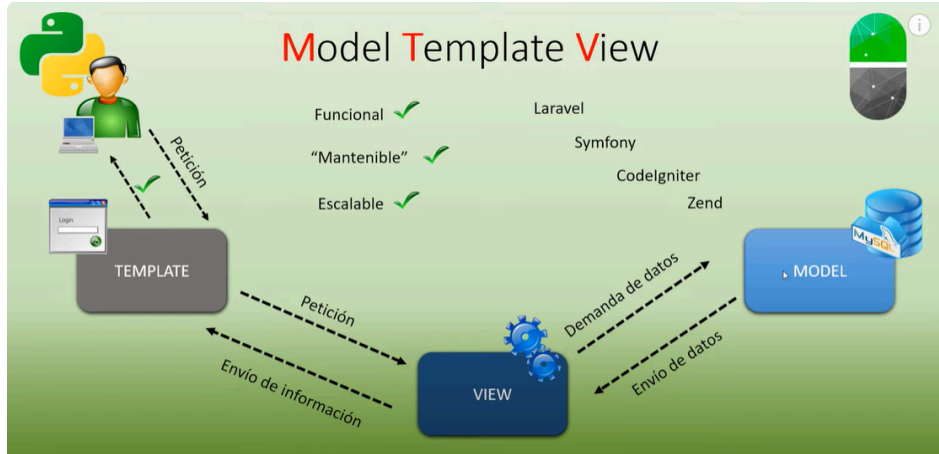


Imagen 2. Model Template View. Características y funcionalidad del Modelo Vista Controlador MTV (Juan Díaz, s.f.).

Esta tecnología soporta oficialmente diferentes bases de datos: SQLite3, PostgreSQL, MySQL, Oracle. También, se encuentran conectores ofrecidos por terceros como: SQL Server, SAP SQL, DB2 y Firebird, entre otros. Al ser un *framework* muy popular, se dispone de tutoriales en la propia página de Django, donde se enseña a crear proyectos desde cero y entender la estructura del modelo. Además, existen gran cantidad de tutoriales y documentos en internet, y disponen de un canal de Discord propio donde existe una comunidad de expertos y profesionales que brindan ayuda de forma desinteresada.

3.1.2. ChatGPT

Como se ha mencionado en el capítulo anterior, ChatGPT es un modelo de lenguaje desarrollado por OpenAI, empresa se dedica a la investigación e implementación de inteligencia artificial, que se define, según (RAE, s.f.) como la "disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico".

ChatGPT es una herramienta diseñada para mantener conversaciones con el usuario, capaz de dar respuestas razonadas e incluso de autocorregirse en caso de haber cometido errores. Al tratarse de una tecnología reciente, esta no es perfecta y podemos encontrar ciertas limitaciones que habrá que tener en cuenta a la hora de utilizarla. Entre estas barreras encontramos:

- Respuestas incorrectas aunque estén bien redactadas y sean plausibles.
- Sensibilidad a la redacción de entrada, es decir, es necesario formular adecuadamente la pregunta para que pueda "entenderla" y generar una respuesta satisfactoria.
- Sesgos a la hora de haber entrenado a la herramienta, lo que hace que en algunas respuestas haya tendencia hacia ciertas respuestas.
- Al no existir la opción de que ChatGPT te pregunte por detalles importantes o específicos antes de generar la respuesta, lleva a que el producto tenga que adivinar o deducir sobre qué está preguntando el usuario.
- Fallos a la hora de bloquear preguntas de carácter inapropiado, lo que puede llevar a respuestas sobre temas dañinos o sesgados.

En la aplicación, ChatGPT se integra como una biblioteca que aprovecha una interfaz de programación de aplicaciones (API)¹. En la que se usan sus capacidades para generar respuestas relacionadas con operaciones y resultados matemáticos y código de programación que resuelva los problemas especificados.

¹ Una API es una interfaz de programación de aplicaciones es un conjunto de definiciones y protocolos que se usa para diseñar e integrar el *software* de las aplicaciones (Red Hat, 2023).

3.1.3. SymPy

SymPy es una biblioteca de matemáticas simbólicas de Python. Se basa en el lenguaje de Python, no consume muchos recursos y, al ser una biblioteca, es fácilmente integrable en diferentes aplicaciones. Además, cuenta con la ventajas de ser gratuita.

En esta biblioteca, se pueden encontrar soluciones para gran variedad de operaciones matemáticas. Algunas de las operaciones que podemos explotar mediante SymPy son: simplificación de expresiones algebraicas, derivadas, integrales, ecuaciones algebraicas, sistemas de ecuaciones, funciones trigonométricas, exponenciales, matrices, vectores, polinomios, álgebra lineal... En su propia página web existen diferentes tutoriales y documentación suficiente sobre las distintas funciones y funcionalidades que se pueden utilizar (SymPy, s.f.).

3.1.4. Matplotlib y Seaborn

Matplotlib y Seaborn son bibliotecas de Python utilizadas para la visualización de datos. Con ellas, se pueden generar gran variedad de gráficos que facilitan la comprensión y análisis de datos (Matplotlib, s.f.) (Seaborn, s.f.).



Imagen 4. Ejemplos del uso de Matplotlib.

3.1.5. JDoodle

JDoodle es un compilador *online* capaz de optimizar código, explicarlo y añadir comentarios describiendo qué función tiene cada línea del código, entre otros. En su página web explican que su misión nació de la visión de revolucionar el desarrollo, hacer que la codificación sea accesible para todos y crear una industria de desarrollo de software más equitativa e inclusiva.

JDoodle dispone de una API, lo que supone que se puede enviar información a la herramienta y obtener respuestas, es decir, enviar entradas para obtener salidas de interés para el trabajo. De esta manera, se obtiene automatización de los procesos para agilizar y dar facilidad en la integración de la plataforma.



Imagen 5. Logo de JDoodle.

3.1.6. SQLite

SQLite es un sistema de gestión de bases de datos relacional que se caracteriza por ser ligero, de fácil implementación y sin necesidad de configuración extensa. Una de sus principales ventajas radica en su portabilidad, ya que su base de datos se almacena en un solo archivo, lo que facilita su transporte y distribución. Además, al no requerir un servidor independiente, es ideal para aplicaciones de tamaño pequeño a mediano y proyectos con recursos limitados.

En el contexto del proyecto, se ha utilizado SQLite debido a que es una de las bases de datos predeterminadas compatibles con Django. La integración nativa de Django con SQLite simplifica el proceso de desarrollo y despliegue, permitiendo centrarse en la lógica de la aplicación sin las preocupaciones por la gestión compleja de la base de datos.

3.2. Tecnologías desechadas

Reflex es un *framework* de código abierto cuya finalidad es la de crear rápidamente aplicaciones web interactivas usando únicamente Python. Su principal atractivo consiste en la necesidad de solo tener que dominar un único lenguaje de programación (Reflex, s.f.).

Se comenzó a denominar Reflex en junio de 2023 tras un cambio de marca de la aplicación Pynecone, creada en 2022 y de la que heredó todo su código. Se vende como un marco de trabajo para novatos y expertos, siendo fácil de usar pero permitiendo usos avanzados. Maneja la interfaz de usuario, la lógica del lado del servidor y el despliegue de la aplicación, lo que permite no seguir el tradicional modelo vista controlador (Franco Torres, 2024).



Imagen 6. Logo de Reflex.

Actualmente, el equipo de desarrollo trabaja en la incorporación de estilos más complejos y animados, y en ampliar su biblioteca de componentes básicos. Además, tienen como objetivo añadir más plantillas base y componentes de alto nivel para, según cuentan, que “los desarrolladores puedan pasar de una idea a una aplicación

funcional lo más rápido posible". Por último, señalan que el *framework* es solo la mitad de su visión y que en los próximos meses construirán y lanzarán una versión beta de su propio servicio de *hosting* para que los usuarios puedan desplegar, escalar y compartir aplicaciones rápidamente. Sus principales áreas son la seguridad, la fiabilidad y la optimización del rendimiento (Reflex, s.f.).

Tanto por la atracción por las premisas y características descritas, como por el objetivo de profundizar conocimientos en Python, al no ser un lenguaje muy usado en los años de carrera, se escogió Reflex como medio para visualizar el fin de nuestra investigación con la idea de aprovechar su modularidad, sus plantillas y su fácil despliegue. Tras investigar sobre el entorno de trabajo y desplegar una página de inicio sencilla, la tecnología se descarta. Un motivo fue la complejidad que adquiere la programación con Reflex cuando se intenta salir del código y herramientas base que se proporcionan y se quieren definir estilos y estructuras propios. El otro fue la poca documentación y foros o lugares de ayuda disponibles al tratarse de una tecnología muy reciente.

Capítulo 4 - Arquitectura

Como se ha explicado en el capítulo de tecnologías utilizadas, Django es un *framework* de desarrollo web que sigue el patrón modelo-vista-controlador (MVC), renombrado como el patrón Modelo-vista-Template (MVT). Este proporciona un conjunto muy completo de herramientas y características que facilita el rápido desarrollo del proyecto con una aplicación web segura y escalable.

4.1. Estructura general

El directorio principal contiene archivos de configuración esenciales, como:

- **Settings.py:** archivo que sirve para la configuración y personalización de la aplicación. Se va a analizar cada sección y su implicación en el sistema:
 - Configuración de rutas y directorios: en este apartado se encuentra la ruta base del proyecto (`BASE_DIR`). Por otro lado, `STATICFILES_DIRS` y `STATIC_ROOT` son las variables que establecen las rutas para los archivos estáticos de la aplicación, como imágenes.
 - Configuración de seguridad: `SECRET_KEY` es una cadena secreta que se usa para firmar las sesiones y otros datos sensibles. La variable `DEBUG` simplemente controla si dicho modo está habilitado o no. En este proyecto su valor es `TRUE`, lo que significa que está activado.
 - Configuración de aplicaciones: `INSTALLED_APPS` denomina las aplicaciones instaladas en el proyecto, algunas ya integradas por defecto, como la administración, la autenticación y las sesiones, y otras creadas desde cero, como la aplicación EMChatGPT.
 - Configuración de plantillas: `TEMPLATES` configura el motor de plantillas que usa la aplicación, lo que incluye la ubicación de las plantillas HTML y

los *context processors*, que proporcionan variables de contexto a las plantillas.

- Configuración de la base de datos: **DATABASES** define la configuración de la base de datos utilizada por la aplicación que, en este caso, es SQLite.
 - Configuración de validación de contraseñas, de internacionalización y zonas horarias.
-
- **Urls.py**: este archivo define como se mapean las URLs a las vistas de la aplicación. En primer lugar, se importan las funciones y las clases de vistas desde el modulo `views.py` de la aplicación EMChatGPT. Después, se configuran todas las URLs disponibles en la lista `urlpatterns`, donde cada una tiene su ruta correspondiente y un nombre identificativo. De esta manera, cada ruta URL está asociada a una función de vista específica, permitiendo a Django manejar las solicitudes entrantes y generar las respuestas correspondientes.

 - **Wsgi.py**: es el punto de entrada para servidores web compatibles WSGI para servir a la aplicación, es decir, es el encargado de configurar el entorno de ejecución y expone la aplicación WSGI de Django, lo que permite que la aplicación sea servida por un servidor web compatible con WSGI.

 - **Manage.py**: el archivo `manage.py` es una utilidad de línea de comandos que forma parte de todos los proyectos de Django y se utiliza para realizar las tareas administrativas del proyecto, que son:
 - Iniciar el servidor de desarrollo: mediante la consola de comandos, a través de la instrucción `python manage.py runserver`, se permite la visualización e interacción con la aplicación.

- Crear y aplicar las migraciones a la base de datos: usando el comando `manage.py makemigrations` para crear los archivos de migración que describen los cambios necesarios en el esquema de la base. Además, para aplicar las migraciones pendientes y actualizar la estructuras de las tablas `manage.py migrate`

- Plantillas: este directorio es una parte crucial de la estructura del proyecto, ya que contiene todos los archivos HTML que definen la interfaz de usuario, es decir, almacena las plantillas que se renderizan dinámicamente con datos proporcionados por las vistas de Django. En el contexto del patrón MVT, el directorio plantillas desempeña el papel de la vista o *template* junto al archivo *views.py*. Las vistas en Django son responsables de procesar las solicitudes del cliente y devolver una respuesta, que en la mayoría de casos es una plantilla renderizada. Por lo tanto, estas actúan como la capa de presentación de la aplicación web.

- **Views.py:** este archivo se encarga de administrar las vistas que gestionan las solicitudes del cliente y generan las respuestas. También, define las funciones que representan las diferentes páginas web de la aplicación y se establecen las acciones a realizar cuando se accede a estas. Sus funciones son:
 - Manejo de solicitudes HTTP: dentro del archivo, se definen algunas funciones que son llamadas cuando se recibe una solicitud HTTP en el servidor. Estas, procesan la solicitud y generan la respuesta correspondiente.
 - Renderización de plantillas: la mayoría de funciones del archivo utilizan el método *render* para renderizar las plantillas HTML.

- Procesamiento de formularios: algunas de las clases de `views.py` manejan la lógica de procesamiento de los formularios implementados en `forms.py`. Por ejemplo, funciones como `manejar_formulario`, `codigo`, `analizar_expresion` procesan los datos enviados a través de los formularios, realizan operaciones de análisis de expresiones matemáticas, consultas de bases de datos y devuelven los resultados apropiados.
- Interacción con la base de datos: es nexo de unión entre la base de datos y la plantilla que la muestra. Se encarga tanto de la lectura cuando el usuario solicita ver la base de datos, como de la escritura a la hora de realizar una nueva operación matemática. El sistema automáticamente incluye esta nueva entrada a su tabla correspondiente. Esto se realiza a través de los métodos `manejar_estadisticas` y `ver_estadisticas`, ambos basándose en el archivo `models.py`.
- Generación de gráficos y visualizaciones: al acceder a la base de datos, encontramos las clases encargadas de la generación de los distintos gráficos que se pueden visualizar en la aplicación: `generar_grafico_correlacion`, `mostrar_correlacion`, `grafico_variables`, `grafico_posibilidad_acierto`, `obtener_grafico_probabilidad_integral`, además de las funciones de generar el gráfico de correlación por cada tipo de operación matemática.

De esta manera, `views.py` es una pieza clave de la aplicación, ya que contiene la lógica para manejar las solicitudes entrantes, procesar datos, interactuar con la base de datos y generar las respuestas adecuadas a las

solicitudes del usuario. Es responsable de la funcionalidad correspondiente del lado del servidor.

```
def manejar_formulario(request, opcion, template):
    # Obtenemos la clase de formulario correspondiente según la opción seleccionada.
    FormClass = FORMS[opcion]
    # Verificamos si se ha enviado un formulario mediante POST.
    if request.method == "POST":
        # Creamos una instancia del formulario con los datos recibidos.
        form = FormClass(request.POST)
        # Verificamos si el formulario es válido.
        if form.is_valid():
            # Construimos la consulta a partir de los datos del formulario.
            consulta = construir_consulta(form, opcion)
            # Iniciamos el controlador para procesar la consulta.
            resultado = inicioControlador(consulta)
            # Si la opción no es 7, realizamos análisis de estadísticas y manejamos las estadísticas.
            if opcion != 7:
                estadisticas_dicc = analizar_expresion(consulta['funcion'], opcion, consulta)
                manejar_estadisticas(estadisticas_dicc, resultado)
            # Redirigimos a la vista de solución de operaciones con el resultado obtenido.
            return solucionOperaciones(request, resultado)
        else:
            # Si no se ha enviado un formulario, creamos una instancia del formulario vacío.
            form = FormClass()
            # Renderizamos la plantilla con el formulario correspondiente.
            return render(request, template, {"form": form})
# Función para manejar el formulario de Código
```

Imagen 7. Función manejo de formularios de las operaciones.

```
# Función para manejar el formulario de Código.
def codigo(request):
    # Definimos la clase de formulario como FormularioCodigo.
    FormClass = FormularioCodigo
    # Verificamos si se ha enviado un formulario mediante POST.
    if request.method == "POST":
        # Creamos una instancia del formulario con los datos recibidos.
        form = FormClass(request.POST)
        # Verificamos si el formulario es válido.
        if form.is_valid():
            # Obtenemos el lenguaje, la descripción y los casos del formulario.
            lenguaje = form.cleaned_data['lenguaje']
            descripcion = form.cleaned_data['area']
            casos = form.cleaned_data['casos']
            # Iniciamos el controlador para procesar el código.
            resultado = inicioControladorCodigo(lenguaje, descripcion, casos)
            # Redirigimos a la vista de solución de código con el resultado obtenido.
            return solucionCodigo(request, resultado)
        else:
            # Si no se ha enviado un formulario, creamos una instancia del formulario vacío.
            form = FormClass()
            # Renderizamos la plantilla de Código con el formulario correspondiente.
            return render(request, "Codigo.html", {"form": form})
```

Imagen 8. Función para manejar el formulario de código.

```

def manejar_estadisticas(estadisticas_dicc,resultado):
    tipo_operacion = estadisticas_dicc["Opcion"]

    try:
        # Obtener el último registro
        ultima_estadistica = Estadistica.objects.latest('id')
        # Obtener el siguiente ID
        nuevo_id = ultima_estadistica.id + 1
    except Estadistica.DoesNotExist:
        # Si no hay registros, comenzar con ID 1
        nuevo_id = 1

    estadistica = Estadistica(
        id=nuevo_id,
        opcion=tipo_operacion,
        grado_maximo=estadisticas_dicc["Grado maximo"],
        longitud_maxima=estadisticas_dicc["LongitudMax"],
        decimal_mas_grande_largo=estadisticas_dicc["Decimal mas grande largo"],
        suma=estadisticas_dicc["suma"],
        resta=estadisticas_dicc["resta"],
        multiplicacion=estadisticas_dicc["multiplicacion"],
        exponente=estadisticas_dicc["exponente"],
        division=estadisticas_dicc["division"],
        sqrt=estadisticas_dicc["sqrt()"],
        factorial=estadisticas_dicc["factorial"],
        numeros_irracionales= estadisticas_dicc["numeros irracionales"],
        orden_derivada = estadisticas_dicc["orden derivada"],
        seno=estadisticas_dicc["seno"],
        coseno=estadisticas_dicc["coseno"],
        tangente=estadisticas_dicc["tangente"],
        infinito_positivo=estadisticas_dicc["-oo"],
        infinito_negativo=estadisticas_dicc["oo"],
        integral_definida=estadisticas_dicc["integral definida"],
        acierto=resultado['resultado']
    )

    estadistica.save()

```

Imagen 9. Función encargada de añadir nuevos datos a la tabla.

```

def ver_datos(request):
    estadisticas = Estadistica.objects.all()
    return render(request, 'Datos.html', {'estadisticas': estadisticas})

```

Imagen 10. Función para ver los datos de la tabla.

- **Models.py:** es el archivo encargado de la definición de la estructura y comportamiento de los modelos de datos de la aplicación. La clase *Estadísticas* mapea los datos a una tabla en la base de datos y define los campos y métodos asociados a esa entidad. Se encarga de:
 - Modelado de datos: define las entidades y relaciones que componen la base de datos de la aplicación.
 - Integridad de los datos: aplica las reglas de validación para garantizar que los datos almacenados cumplan con ciertos criterios.
 - Interacción con la base de datos: maneja la interacción con la base de datos subyacente, lo que facilita la gestión y manipulación de los mismos.

```

from django.db import models

class Estadistica(models.Model):
    id = models.AutoField(primary_key=True)
    opcion = models.CharField(max_length=255)
    grado_maximo = models.IntegerField(default=0)
    longitud_maxima = models.IntegerField(default=0)
    decimal_mas_grande_largo = models.IntegerField(default=0)
    suma = models.IntegerField(default=0)
    resta = models.IntegerField(default=0)
    multiplicacion = models.IntegerField(default=0)
    exponente = models.IntegerField(default=0)
    division = models.IntegerField(default=0)
    sqrt = models.IntegerField(default=0)
    factorial = models.IntegerField(default=0)
    numeros_irracionales = models.IntegerField(default=0)
    orden_derivada = models.IntegerField(default=0)
    seno = models.IntegerField(default=0)
    coseno = models.IntegerField(default=0)
    tangente = models.IntegerField(default=0)
    infinito_positivo = models.IntegerField(default=0)
    infinito_negativo = models.IntegerField(default=0)
    acierto = models.BooleanField(default=False)
    integral_definida = models.BooleanField(default=False)
    def __str__(self):
        return f"{self.opcion} - ID: {self.id}"

```

Imagen 11. Función estadística.

- **Forms.py**: es el encargado principal de la gestión y validación de los formularios de la aplicación web. Se encarga de:
 - Definición de los formularios: define las clases de formularios que especifican qué campos se deben incluir en cada uno de los distintos formularios HTTP de cada operación matemática y cómo se deben validar esos campos.
 - Interacción con el usuario: proporciona una interfaz para que los usuarios ingresen datos de manera sencilla a través de formularios HTML. Dichos formularios incluyen campos de texto, selección, casillas de verificación, botones, etc.
 - Validación de los datos: define las reglas de validación para cada campo del formulario, por ejemplo, si se puede introducir letras o no, o solo algunas porque representan expresiones matemáticas como 'cos()', 'tan()', infinito, etc.
 - Procesamiento de datos: después de que el usuario envíe un formulario, el archivo maneja la validación de los datos ingresados, y si estos son válidos, procesa esos datos a través de la función correspondiente de **views.py**.
 - Seguridad: para evitar una mayor carga de trabajo a las validaciones, los caracteres de tipo letra se transforman a mayúscula directamente con el fin de facilitar al usuario el uso de la aplicación.

```

class BaseMathForm(forms.Form):
    # Definimos un método para limpiar la expresión matemática ingresada.
    def clean_math_expression(self, expression, allowed_chars, must_chars):
        # Convertimos la expresión a minúsculas para hacerla uniforme.
        expression = expression.lower()

        # Verificamos si la expresión contiene solo caracteres permitidos.
        if not re.match(r'^[' + re.escape('').join(allowed_chars) + '()]+$ ', expression):
            raise forms.ValidationError("La expresión contiene caracteres no permitidos.")

        # Verificamos si la operación contiene letras justo antes o después de un número.
        if re.search(r'\d+[a-zA-Z][a-zA-Z]+\d+', expression):
            raise forms.ValidationError("La operación no puede contener letras justo después o antes de un número.")

        # Verificamos si la expresión contiene el caracter obligatorio indicado.
        if must_chars not in expression:
            raise forms.ValidationError("La operación debe contener un signo igual (=)")
        elif must_chars and must_chars not in expression:
            # Dividimos la expresión en dos partes alrededor del signo igual.
            partes = expression.split('=')
            # Verificamos si hay datos a ambos lados del signo igual.
            if len(partes) != 2 or not partes[0].strip() or not partes[1].strip():
                raise forms.ValidationError("Debe haber datos a la izquierda y a la derecha del signo igual (=).")

        # Creamos una pila para verificar la correspondencia de paréntesis.
        stack = []
        for char in expression:
            if char == '(':
                stack.append('(')
            elif char == ')':
                # Verificamos si hay un paréntesis cerrado sin su correspondiente paréntesis abierto.
                if not stack:
                    raise forms.ValidationError("Paréntesis cerrado sin paréntesis abierto correspondiente.")
                stack.pop()

        # Verificamos si hay paréntesis abiertos sin su correspondiente paréntesis cerrado.
        if stack:
            raise forms.ValidationError("Paréntesis abierto sin paréntesis cerrado correspondiente.")

        # Retornamos la expresión validada.
        return expression

```

Imagen 12. Formulario base común a todas las funciones.

```

class FormularioIntegral(BaseMathForm):
    # Definimos un campo para ingresar la función a integrar.
    funcion = forms.CharField(label = any, widget=forms.TextInput(attrs={'class': 'form-control form-control-lg', 'p
    # Definimos campos ocultos para los límites de integración y la indicación de si la integral es definida.
    limite_inferior = forms.DecimalField(widget=forms.HiddenInput(), required=False)
    limite_superior = forms.DecimalField(widget=forms.HiddenInput(), required=False)
    integral_definida = forms.BooleanField(required=False, widget=forms.HiddenInput())

    # Método para limpiar y validar el campo de la función.
    def clean_funcion(self):
        # Obtenemos la función ingresada.
        funcion = self.cleaned_data.get('funcion')
        # Definimos los caracteres permitidos en la función.
        allowed_chars = "0123456789abcdefghijklmnopqrstuvwxyz,*/+^-^"
        # No necesitamos un carácter obligatorio en este caso.
        must_chars = ""
        # Llamamos al método de limpieza y validación general de expresiones matemáticas.
        return self.clean_math_expression(funcion, allowed_chars, must_chars)

    # Método para limpiar y validar el formulario en su conjunto.
    def clean(self):
        # Llamamos al método de limpieza y validación general del formulario.
        cleaned_data = super().clean()
        # Llamamos al método de limpieza y validación específico para los límites de integración.
        return self.clean_limites()

```

Imagen 13. Función formulario integral.

```

# Método para limpiar y validar los límites de integración.
def clean_limites(self):
    # Obtenemos los datos limpios del formulario.
    cleaned_data = super().clean()
    # Obtenemos los límites de integración.
    limite_inferior = cleaned_data.get('limite_inferior')
    limite_superior = cleaned_data.get('limite_superior')

    # Si ambos límites están definidos, verificamos que el límite inferior sea menor que el límite superior.
    if limite_inferior is not None and limite_superior is not None:
        cleaned_data['integral_definida'] = True
        if limite_inferior >= limite_superior:
            raise forms.ValidationError("El límite inferior debe ser menor que el límite superior si ambos están definidos.")

    # Si solo uno de los límites está definido, lanzamos una excepción.
    elif limite_inferior is None and limite_superior is not None:
        raise forms.ValidationError("No se puede dejar solo uno de los límites sin definir")

    elif limite_inferior is not None and limite_superior is None:
        raise forms.ValidationError("No se puede dejar solo uno de los límites sin definir")

    return cleaned_data

```

Imagen 14. Función para limpiar y validar los límites.

La estructura y las clases de este archivo están diseñadas y programadas para proporcionar al proyecto:

- Abstracción de formularios HTML: los formularios están definidos de una manera legible y mantenible en Python en lugar de puro HTML, lo que facilita la gestión y personalización de estos.
- Reutilización de código: los formularios están compuestos por una parte de validación común a todas las operaciones matemáticas. Con esto se logra que los formularios puedan ser reutilizados en múltiples vistas y plantillas, ayudando a tener un código limpio y modular.
- Validación centralizada: se centraliza la lógica de validación en un solo lugar, lo que facilita la modificación y actualización de las reglas de validación sin necesidad de cambiar múltiples vistas, archivos y plantillas.
- Integración con modelos de datos: puede vincularse directamente con los modelos de datos definidos en *models.py*, generando una mayor fluidez en la creación, actualización y validación de objetos de modelo a partir de los datos ingresados por el usuario.
- Facilidad de personalización: permite personalizar la apariencia y comportamiento de los formularios utilizando *widgets*, botones, y opciones de configuración proporcionadas por Django. De esta forma, ayudan al usuario a poder manejar y navegar por la aplicación.
- La estructura que siguen los formularios, como se ha descrito anteriormente, tiene una función llamada **BaseMathForm**. Esta contiene las validaciones comunes a todos, donde se restringen aquellos caracteres que no están presentes en ninguno de los casos, la apertura de paréntesis que están abiertos pero no cerrados y transforma todo el contenido al mismo formato para que SymPy pueda trabajar sin excepciones.

Posteriormente, cada tipo de operación matemática tiene su propia clase donde se crean los campos necesarios para poder ejecutar la operación. Por ejemplo: en el caso de las integrales, encontramos un campo de texto donde introducir la función y otros dos campos, esta vez numéricos de tipo decimal, para indicar tanto el límite superior e inferior en caso de que el usuario quiera que la integral sea definida. Es por ello que estos campos no son obligatorios. Después, se declaran la/s función/es *cLean* donde en caso de no necesitar más validación que la común solo se creará esta. En caso contrario, se crearán más funciones de este tipo para proceder a la validación de los elementos específicos de cada caso. Siguiendo el ejemplo anterior de las integrales, se cuenta con una función llamada *cLean_Limites* para verificar que el límite inferior no es mayor al superior, que existen ambos límites y no solo uno, y por lo tanto para determinar si la integral es o no definida.

4.1.1. El controlador

El controlador es la clase principal del proyecto, encargada de procesar y realizar todas las operaciones recibidas desde las diferentes vistas. En esta clase se incluyen las funciones principales para las distintas funcionalidades que realiza la aplicación, como la resolución de problemas matemáticos o la generación de código que resuelva estos anteriores.

```

def inicioControlador(pregunta):
    # Inicializamos un diccionario para almacenar el resultado.
    resultado = {}

    # Verificamos y corregimos la pregunta proporcionada.
    correction = tipo(pregunta)
    if isinstance(correction, str) and "Error" in correction:
        # En caso de error, lanzamos una excepción con un mensaje explicativo.
        raise ValueError(f"Revise si ha elegido la opción correcta, si la ha escrito correctamente y vuelva a intentarlo. {correction}")

    # Construimos la consulta para ChatGPT.
    gpt = construir_consulta_gpt(pregunta)

    # Decidimos la introducción adecuada según la opción seleccionada.
    introduccion = decision(pregunta["opcion"])

    # Generamos la completación de la conversación con ChatGPT.
    chat_completion = client.chat.completions.create(model="gpt-3.5-turbo-1106", messages=[introduccion, {"role": "user", "content": gpt}])
    # Obtenemos la respuesta generada por ChatGPT.
    respuesta = chat_completion.choices[0].message.content

    # Comparamos la respuesta de ChatGPT con la corrección obtenida.
    if comparar(respuesta, correction):
        acierto = True
    else:
        acierto = False

    # Almacenamos los datos relevantes en el diccionario de resultado.
    resultado["opcion"] = pregunta["opcion"]
    resultado["pregunta"] = gpt
    resultado["resultado"] = acierto
    resultado["sympy"] = correction
    resultado["chatgpt"] = respuesta

    # Retornamos el resultado.
    return resultado

```

Imagen 15. Función para la petición matemática a ChatGPT.

4.2. Resolución de problemas matemáticos.

Para esta funcionalidad, el proyecto se apoya en la librería *OpenAI*, importada de la clase *OpenAI*. Esta será la encargada de conectarse con la API, únicamente utilizando un *ID* generado desde el panel de administración de la propia compañía. Debido al gran abanico de formatos que se pueden utilizar, se debe de mandar en una única petición dos mensajes, diferenciados según la clase con el rol que se le asigne. Son:

- **System:** aquí ha de ser incluido un *prompt* con todo el contexto sobre la petición y el formato que debe de seguir la respuesta. En este caso se incluye el tipo de operación a resolver y una gran explicación de cómo debe ser el formato del

output que devuelva. Esto es crucial dado que las respuestas deben utilizar siempre un formato homogéneo para su comparación con las generadas con las otras tecnologías utilizadas.

- User: aquí únicamente se incluye la operación a resolver respetando el formato que le indicamos en *system*.

Después de realizar la petición a través de la librería, se recibe un fichero JSON del cual se puede extraer información. En esta función también se incluyen las llamadas a las operaciones con SymPy y la comparación de resultados.

4.2.1. SymPy

SymPy es la librería encargada de verificar las soluciones que ChatGPT proporciona a los problemas matemáticos. Antes de realizar cualquier petición a ChatGPT, el controlador será el encargado de llamar a la función y a la clase necesaria para resolver la operación seleccionada. Debido a las validaciones realizadas a nivel de vista, se asegura que las operaciones reciban un diccionario con la estructura necesaria según el tipo. Por ejemplo, una integral definida recibirá las claves `limite_superior` y `limite_inferior` para poder calcular su solución. Esta solución es la que permitirá comparar el resultado con el aportado por ChatGPT y, en caso de que el usuario plantee una consulta no matemática o errónea, el error de SymPy permitiría gasto en *tokens* de la API.

4.2.2. El comparador

El comparador es la parte final del controlador. Es llamado tras las funciones principales de este y se encuentra en su propia clase. Su objetivo es reconvertir todos los tipos de

estructuras de datos recibidas a unas iguales, pero con expresiones que SymPy pueda leer y comparar. Se diferencian tres posibles estructuras:

- El primero de ellos son diccionarios utilizados para sistemas, dado que el resultado debe de coincidir, tanto en clave como en valor. Por lo tanto, no podemos considerar que $x=3, y=-5$ es igual a $x=-5, y=3$.
- La segunda estructura es *set* y es utilizada cuando se tienen varias soluciones para una misma incógnita. Por ejemplo, las ecuaciones polinómicas de un grado alto devuelven varios posibles valores para una misma incógnita.
- Por último, un *string* para representar resultados únicos.

4.3. Generación de código

Para la generación de código se utiliza, de nuevo, la librería *OpenAI* interaccionando con la API. La función encargada se divide en dos partes:

- En la primera, el controlador envía una petición con el *prompt* que manda el usuario para la generación del código en el lenguaje solicitado; aún no se incluyen los *inputs*. En este punto, ChatGPT devuelve únicamente el código como se le ha indicado.
- En la segunda, se usa ese mismo código para mandarle una nueva petición con la respuesta anteriormente generada y el *input* que introdujo el usuario. De esta forma, se asegura obtener una mejor respuesta y un formato más sencillo y claro de procesar.

La función retorna un diccionario con el código y la solución para mostrarlo por pantalla al usuario.

```

def inicioControladorCodigo(lenguaje, programa, casos):

    # Introduccion del mensaje
    introduccion = f"Debes programar el programa descrito a continuación en el lenguaje {lenguaje}. Solo escribe el código,
    if casos is not None:
        introduccion+=f"Te voy a dar los casos de prueba estos casos de prueba debes leerlos como un input, no incluyas ning
    # Concatena la introducción con la descripción del programa
    descripcion_programa = introduccion + programa

    # Definir los parámetros de la consulta para la generación de código
    parametros = {
        "model": "gpt-3.5-turbo-1106",
        "messages": [{"role": "user", "content": descripcion_programa}]
    }

    try:
        # Envía la consulta a la API de ChatGPT
        chat_completion = client.chat.completions.create(**parametros)

        # Extrae el programa generado de la respuesta
        programa_generado = chat_completion.choices[0].message.content

        # Hacer una segunda llamada a ChatGPT con el programa generado y los casos
        introduccionCaso = f"Debes indicar el resultado de aplicar unos casos de prueba al programa {programa_generado}. Sol

        # Concatena la introducción con los casos de prueba
        decripcion_casos = introduccionCaso + casos

        # Definir los parámetros de la consulta para la generación de resultados
        parametros_evaluacion = {
            "model": "gpt-3.5-turbo-1106",
            "messages": [{"role": "user", "content": decripcion_casos}]
        }

        # Obtener la respuesta de la segunda llamada
        chat_completion_evaluacion = client.chat.completions.create(**parametros_evaluacion)

```

Imagen 16. Función para la petición de código a ChatGPT.

4.3.1. JDoodle

JDoodle es la herramienta que utiliza el controlador para comprobar si el código generado por ChatGPT es compilable, ejecutable y la solución es correcta. Para ello, se deben realizar peticiones *POST* a través de la librería *requests* de Python. Para comprobar que las conexiones sean válidas, las peticiones deben incluir: ID del usuario, la clave de dicho usuario, el código a probar, el lenguaje, la versión del lenguaje y opcionalmente el *input*. Para la versión del lenguaje, el controlador tiene un diccionario que lo traduce a su última versión disponible.

La *request* recibe muchos datos interesantes, entre ellos el estado de la petición, la respuesta de la ejecución, el *output*, el tiempo de ejecución y la carga de la ejecución. La función utiliza el `.status_code` para saber si la *request* ha sido exitosa (`status_code==200`) o si ha habido un error tipo 400 o 500; de la petición o del servidor, respectivamente. Al final, la función devuelve *True* en función de los casos mencionados y el *output* de la solución. En caso de error en la compilación o en la ejecución, JDoodle devuelve un valor numérico entre 1-7 que permite, a través de un diccionario, traducirlo e informar al usuario.

```
def jdoodle(solucionChat, lenguaje, casos):  
    # URL del endpoint de la API de JDoodle  
    url = "https://api.jdoodle.com/v1/execute"  
  
    # Datos de la solicitud  
    data = {  
        "clientId": jdoodleID,  
        "clientSecret": jdoodleSecret,  
        "script": solucionChat,  
        "language": lenguaje,  
        "versionIndex": versiones_lenguajes[lenguaje]  
    }  
  
    if casos is not None:  
        casos+="\n"  
        data["stdin"]=casos  
  
    # Headers de la solicitud  
    headers = {'Content-type': 'application/json'}  
  
    # Realizar la solicitud POST  
    response = requests.post(url, data=json.dumps(data), headers=headers)  
    respuesta_json=response.json()  
    if response.status_code == 200:  
        return {"ejecucion": codigos_estado[respuesta_json.get('statusCode', ' ')], "resultadoJdoodle": respuesta_json.get('output', ' '), "error": None}  
    else:  
        if response.status_code == 429 or response.status_code == 401:  
            return {"ejecucion": codigos_estado[respuesta_json.get('statusCode', ' ')], "resultadoJdoodle": None, "error": respuesta_json.get('error')}  
        else:  
            return {"ejecucion": codigos_estado[respuesta_json.get('statusCode', ' ')], "resultadoJdoodle": None, "error": respuesta_json.get('error')}
```

Imagen 17. Función petición API JDoodle.

4.4. Gráficos

La lógica de la creación de los gráficos, sin tener en cuenta las plantillas HTML para su visualización, se encuentra en el archivo *views.py*. Dentro de este, se utilizan una serie

de librerías esenciales para poder crearlos. Entre ellas encontramos Pandas, Matplotlib y Seaborn; junto a otros para la manipulación de imágenes y codificación base64. Los gráficos que se pueden visualizar en la aplicación son gráficos de barras y matrices.

4.4.1. Tipos de gráficos utilizados

- Gráfico de barras: se utilizan gráficos de barras para mostrar la relación entre una variable categórica y una variable numérica. La altura de cada barra representa la frecuencia o el valor de la variable en esa categoría.
- Matriz de correlación: es una representación tabular de las correlaciones entre variables. Cada celda en la tabla muestra el coeficiente de correlación entre dos variables. La matriz de correlación, entre las variables de cada fila y columna, proporciona información sobre cómo las variables están relacionadas entre sí. Cada celda de la matriz representa el coeficiente de correlación entre dos variables. Los valores pueden variar entre -1 y 1:
 - 1: indica una correlación positiva perfecta, lo que significa que las variables aumentan juntas en la misma dirección.
 - -1: indica una correlación negativa perfecta, lo que significa que las variables cambian en direcciones opuestas.
 - 0: indica que no hay correlación entre las variables.

La matriz de correlación es útil para identificar relaciones lineales entre diferentes variables en un conjunto de datos. Además, la matriz forma un mapa de calor con el que se diferencian de mejor manera los valores que toma cada variable. Esto ayudará a la hora de realizar las interpretaciones y conclusiones del proyecto.

4.4.2. Funciones principales

- `generar_grafico_correlacion()`: calcula la correlación entre cada variable numérica y la variable de salida *acierto* y genera un gráfico de barras para visualizar esta correlación. Se utiliza en múltiples vistas para diferentes opciones.
- `generar_grafico_correlacion_opcion_X()`: al igual que la anterior, obtiene la correlación entre cada variable y el *acierto* pero en este caso teniendo en cuenta solo las de la opción *x*, es decir, desde la opción 2 a la 7 representando todas las operaciones matemáticas tratadas.
- `mostrar_correlacion(request)`: esta vista se encarga de renderizar el gráfico de correlación en formato base64 en una plantilla HTML.
- `correlacion_variables(request)`: maneja la solicitud *POST* del formulario para calcular y mostrar la matriz de correlación entre las variables seleccionadas por el usuario.
- `grafico_posibilidad_acierto(request)`: genera un gráfico de barras que muestra la probabilidad de *acierto* para cada orden de derivada.
- `obtener_matriz_correlacion(request)`: esta vista maneja la solicitud para generar la matriz de correlación para diferentes opciones y devuelve las imágenes en formato base64 para mostrarlas en la página web.
- `obtener_grafico_probabilidad_integral(request)`: en esta función se genera un gráfico de barras que permite al usuario ver la probabilidad de *acierto* que tiene ChatGPT cuando la integral es definida y cuando no lo es.

```

def generar_grafico_correlacion_opcion_2():
    # Filtrar los datos de la base de datos para obtener solo la opción 2
    estadisticas_opcion_2 = Estadistica.objects.filter(opcion='2')
    # Crear un DataFrame de pandas con los datos de la opción 2
    df_opcion_2 = pd.DataFrame(list(estadisticas_opcion_2.values()))

    # Calcular el porcentaje de aciertos y fallos
    total_registros = len(df_opcion_2)
    aciertos = df_opcion_2['acierto'].sum()
    porcentaje_aciertos = (aciertos / total_registros) * 100
    porcentaje_fallos = 100 - porcentaje_aciertos

    # Calcular la correlación entre cada variable numérica y la variable de salida 'acierto'
    correlaciones = df_opcion_2.drop(columns=['id', 'opcion', 'acierto']).corrwith(df_opcion_2['acierto'])

    # Crear los gráficos
    fig, axs = plt.subplots(2, figsize=(12, 12))

    # Gráfico de porcentaje de aciertos y fallos
    axs[0].bar(['Aciertos', 'Fallos'], [porcentaje_aciertos, porcentaje_fallos], color=['green', 'red'])
    axs[0].set_title('Porcentaje de Aciertos y Fallos - Opción 2')
    axs[0].set_ylabel('Porcentaje')
    axs[0].set_ylim(0, 100)

    # Agregar porcentaje numérico en el gráfico de barras
    for i, porcentaje in enumerate([porcentaje_aciertos, porcentaje_fallos]):
        axs[0].text(i, porcentaje + 1, f'{porcentaje:.2f}%', ha='center', va='bottom', fontsize=10)

    # Gráfico de correlación
    axs[1].bar(correlaciones.index, correlaciones.values, color='skyblue')
    axs[1].set_title('Correlación entre variables y la variable de salida (Acierto) - Opción 2')
    axs[1].set_xlabel('Variables')
    axs[1].set_ylabel('Correlación')
    plt.xticks(rotation=45, ha='right')

    # Ajustar el espaciado entre los gráficos
    plt.tight_layout()

    # Guardar la figura como una imagen
    buffer = BytesIO()
    plt.savefig(buffer, format='png')
    buffer.seek(0)
    imagen_base64 = base64.b64encode(buffer.getvalue()).decode('utf-8')
    plt.close()
    # Retornar la imagen base64
    return imagen_base64

```

Imagen 18. Función crear gráfico correlación de operaciones simples.

```

def mostrar_correlacion_opcion_2(request):
    # Obtener la imagen base64 de los gráficos para la opción 2
    imagen_base64 = generar_grafico_correlacion_opcion_2()

    # Aquí puedes retornar la imagen base64 para mostrarla en tu aplicación web
    return render(request, 'mostrar_correlacion_2.html', {'imagen_base64': imagen_base64})

```

Imagen 19. Función mostrar gráfico correlación operaciones simples.

Cabe destacar que todos los gráficos son dinámicos, lo que significa que son imágenes no estáticas. A medida que el usuario va realizando nuevas operaciones estas se verán reflejadas en todos los gráficos correspondientes. Esto es gracias a que cuando el cliente solicita una página que contiene un gráfico dinámico, el servidor procesa esa solicitud, genera el gráfico utilizando datos disponibles en la base de datos u otra fuente, codifica el gráfico en un formato adecuado y envía el resultado al cliente para su visualización.

4.5. Base de datos

La base de datos usada en este proyecto es de tipo SQLite3 y está basada en el archivo `models.py`. Dicho archivo se construye teniendo como base la biblioteca `django.db.models`, que proporciona las herramientas necesarias para definir y trabajar con modelos de datos en una base de datos relacional.

Django ORM (object-Relational Mapping) se utiliza para interactuar con la base de datos, consiguiendo que, en lugar de escribir consultas SQL directamente, se puedan definir modelos de Python que representan las tablas y luego utilizar el ORM para realizar operaciones de lectura y escritura sobre la misma. Esto simplifica el proceso de manejar la base de datos y hace que el código sea más legible y mantenible. La elección de SQLite3 como motor de base datos se debe a que es una excelente opción para proyectos más pequeños o en desarrollo.

Capítulo 5 - Resultados

En este capítulo se muestra cómo funciona la aplicación desarrollada y se plantea la respuesta al objetivo principal del trabajo, exponiendo los resultados de investigar las operaciones matemáticas que ChatGPT es capaz de realizar de manera correcta y qué características o límites son los que pueden llevar a recibir una respuesta errónea.

5.1. La aplicación

La aplicación contiene tres bloques diferenciados. El primero está dedicado a solicitar a ChatGPT las distintas operaciones matemáticas, mientras que en el segundo se puede pedir a esta inteligencia artificial que realice un programa en un código de programación a elección del usuario. Por último, en el tercer apartado se recogen las estadísticas de las pruebas y la muestra del banco de datos.



Imagen 20. Página de inicio de la aplicación.

El primer paso para solicitar a ChatGPT que resuelva una operación matemática es seleccionar el tipo de operación que se desee. Para ello, se podrá escoger desde el menú de la aplicación y completar el formulario con los datos correspondientes.

Límite

La idea intuitiva de límite forma parte del conocimiento popular. Tender a un límite significa aproximarse a una meta, que no siempre se logra alcanzar. En el ámbito matemático, esta idea se ha plasmado en una definición que combina los conceptos de lo infinitamente pequeño y lo infinitamente grande, en otras palabras, el infinito.

log(x)

+ - * / ^ ! √ log x e π sin cos tan

Elige la variable sobre la que aplicar el límite:

x

Elige el punto de evaluación:

∞

∞

Calcular

Imagen 21. Página de la operación matemática Límite.

Posteriormente, al pulsar en el botón "calcular", la aplicación validará los datos ingresados y, si son correctos, se envían al controlador para su procesado. En caso de no serlo, se muestra un mensaje de error y se solicita al usuario que introduzca datos válidos.



Imagen 22. Mensaje de error de una operación matemática.

Una vez el controlador ha procesado los datos ingresados, se muestra el resultado matemático que la inteligencia artificial proporciona. Asimismo, también se evidencia, mediante un mensaje en pantalla, si ChatGPT ha realizado correctamente la operación matemática.

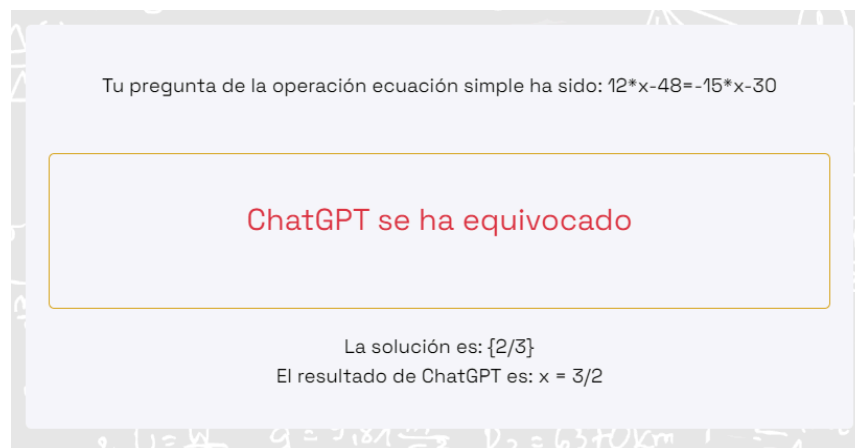


Imagen 23. Pantalla de resultados de una operación matemática.

Respecto al segundo bloque anteriormente mencionado para la petición y ejecución de un código de programación. En primer lugar, se debe seleccionar la opción "Código" en el menú principal de la aplicación, donde se deberá completar el formulario con la información del programa, incluyendo el lenguaje de programación, la descripción del programa y, de forma opcional, los casos de prueba. Para finalizar, el usuario pulsará en "generar".



The image shows a web application interface titled "Generador de código". The interface is set against a background of mathematical formulas and diagrams. It features a light blue form with the following elements:

- A header section with the text "Elige el lenguaje de programación:" followed by a text input field containing the word "Java".
- A section titled "Describe el programa:" followed by a larger text area containing the instruction "Haz un programa que divida dos números pasados como parámetro."
- A section titled "Incluye casos de prueba (Opcional):" followed by a text input field containing the text "126 y 2".
- A yellow button labeled "Generar" at the bottom of the form.

Imagen 24. Pantalla de petición de un código a ChatGPT.

De esta forma, los datos pasan por el controlador para posteriormente mostrar el programa generado por ChatGPT y los resultados de los casos de prueba, si hubiera, junto con la validación del compilador.

Programa generado:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        float num1 = input.nextFloat();
        float num2 = input.nextFloat();

        float result = divide(num1, num2);
        System.out.println(result);
    }

    public static float divide(float a, float b) {
        return a / b;
    }
}
```

Resultado del caso de prueba:

63.0

Corrección del código:

El código es correcto

Imagen 25. Resultados de un programa solicitado a ChatGPT.

Por último, en el tercer bloque, la aplicación cuenta con un apartado donde se pueden visualizar los distintos gráficos generados en la realización de pruebas y solicitudes a la página. De esta forma, la principal función de este apartado es la recopilación de todos los datos usados con distintos casos de prueba para desarrollar el análisis estadístico.

5.2. Principales hallazgos

Con los resultados obtenidos a través del uso de un banco de casos de pruebas, el cual consta de 110 operaciones simples, 106 ecuaciones, 188 integrales, 164 derivadas, 143 límites, se han obtenido los gráficos, que a continuación se mostrarán, que demuestran la correlación entre una serie de características, como el grado de la derivada calculada, y la posibilidad de acierto de ChatGPT.

Observando el primer gráfico, donde se tienen en consideración todas las operaciones y variables analizadas en este estudio, se muestra que gran parte de las variables influyen negativamente en la posibilidad de acierto. Es decir, sin importar el tipo de variable objeto de estudio, existe una tendencia clara hacia el error; a excepción de la longitud máxima y del seno. Asimismo, se puede vislumbrar que algunas influyen positivamente, como es el caso de las raíces cuadradas y los infinitos positivos, pero con valores cercanos a cero. Ello indica que no existe una correlación concluyente como para poder considerarlo un factor determinante.

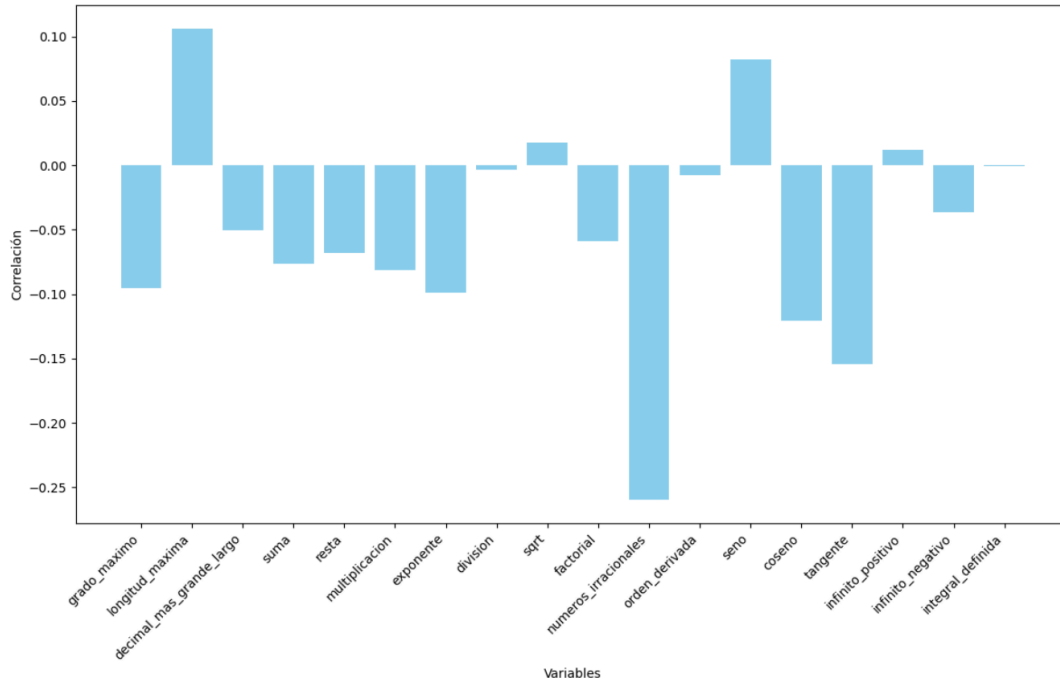


Gráfico 1. Correlación entre variables y acierto del conjunto de resultados.

Para explicar los resultados expuestos por el gráfico 1, se debe observar la información proporcionada por las figuras que se enfocan en un tipo de operación concreta y que se muestran a continuación.

- Operación simple: en este gráfico se observa que la longitud máxima de los enteros es una característica positiva en los resultados que la inteligencia artificial proporciona. Esto se traduce en una menor tasa de error cuanto mayor sea la variable (ver Gráfico 2).

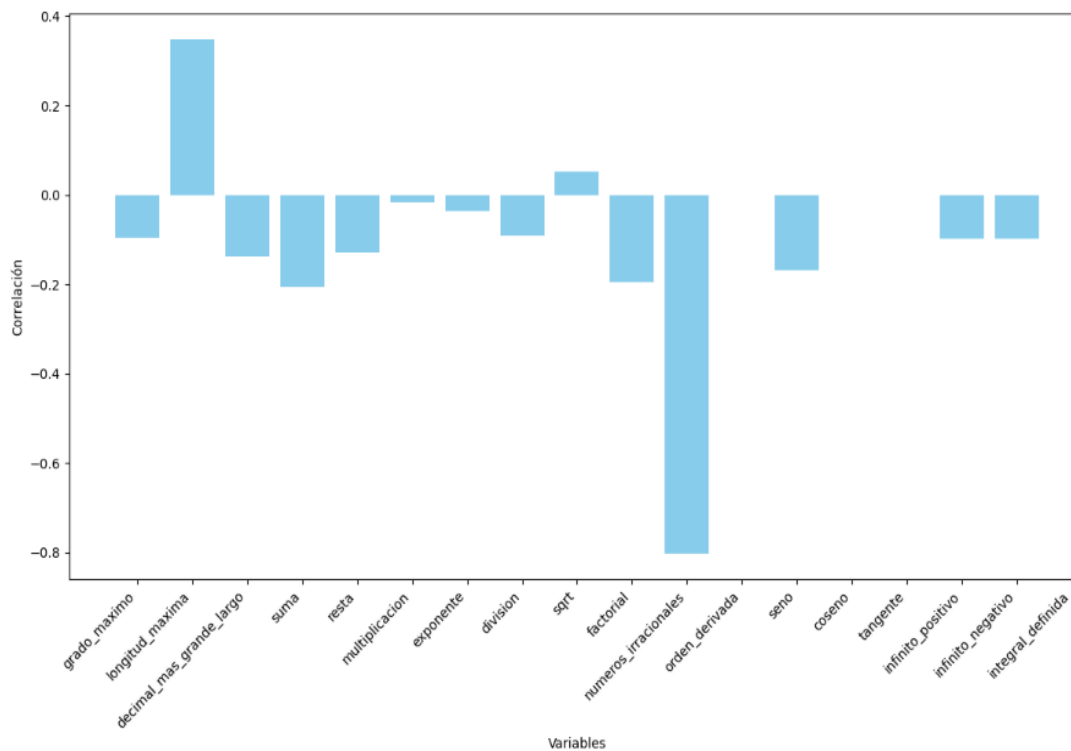


Gráfico 2. Correlación entre variables y acierto de las operaciones simples.

- Ecuaciones: tras la recopilación de los datos, se puede observar que los factores que influyen positivamente en el acierto de ChatGPT son la longitud máxima, resta, multiplicación y la potencia (ver gráfico 3).

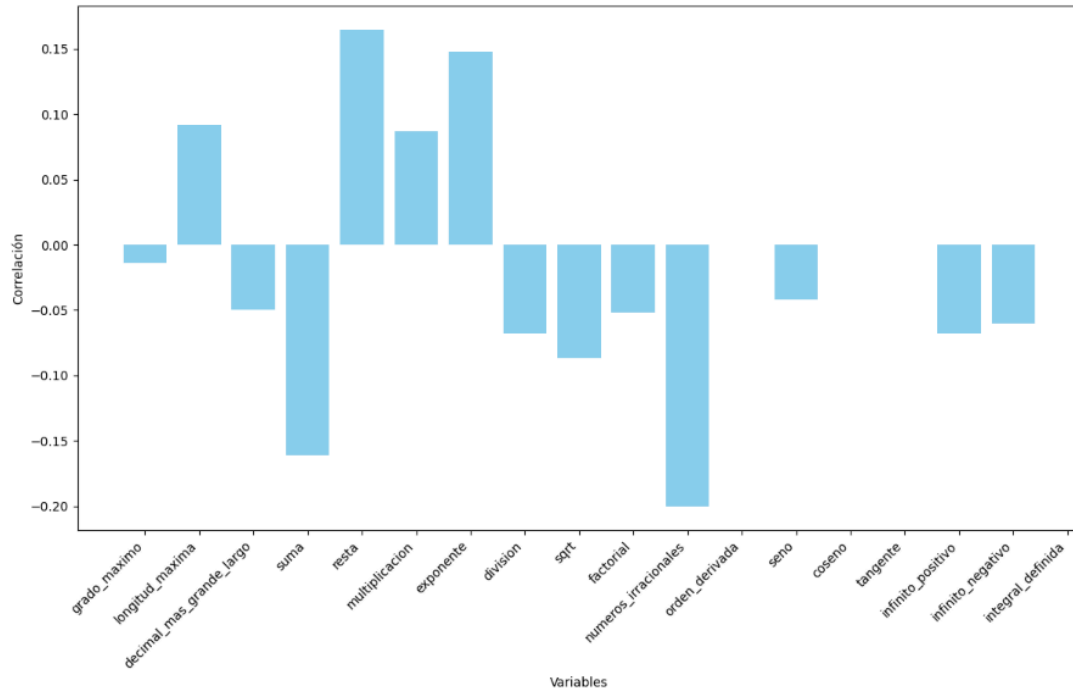


Gráfico 3. Correlación entre variables y acierto de las ecuaciones.

- Integrales: en esta operación las variables que favorecen el porcentaje de aciertos son la raíz cuadrada, el seno, el coseno, infinito y la integral definida como se puede observar en el gráfico 4. Asimismo, la figura 5 revela que la integral definida presenta una posibilidad de acierto de, aproximadamente, un 8 % superior respecto a la indefinida.

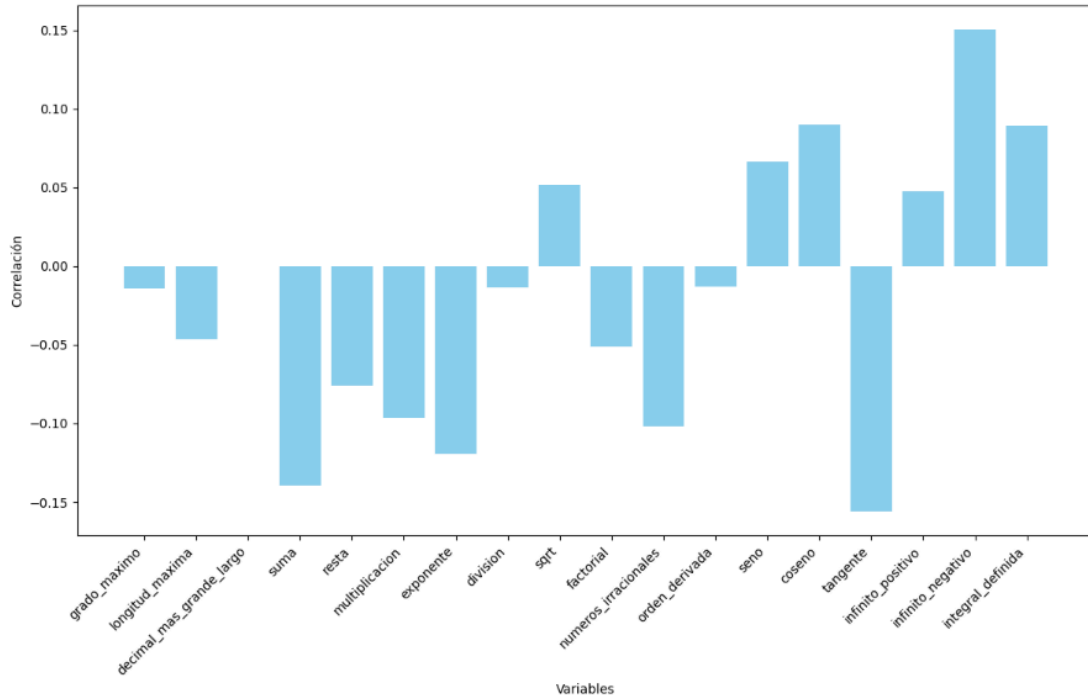


Gráfico 4. Correlación entre variables y acierto de integrales.

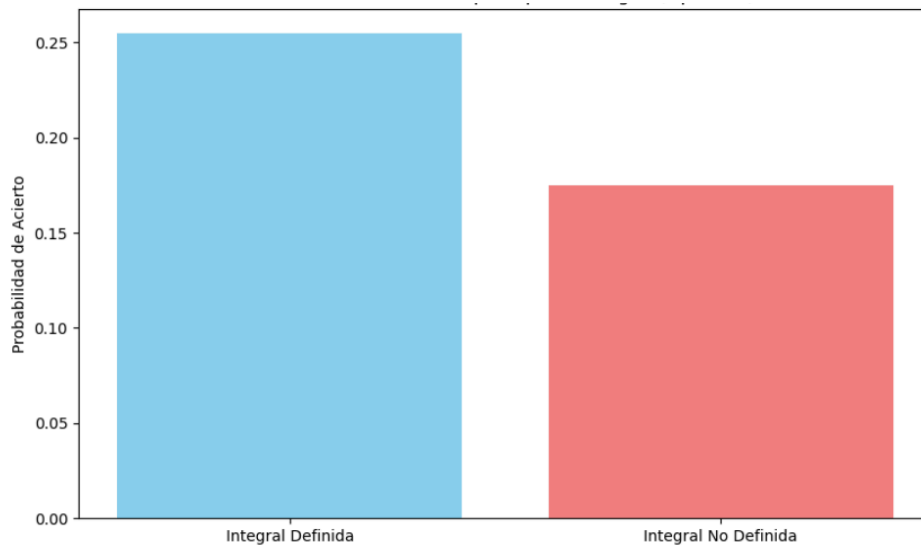


Gráfico 5. Posibilidad de acierto de integrales definidas y no definidas.

- Derivadas: respecto a esta operación matemática. La suma, la resta, la división y el seno son las variables que propician que ChatGPT muestre el resultado correcto (ver gráfico 6). Del mismo modo, la probabilidad de acierto según el orden de la derivada es un factor influyente en el resultado que la inteligencia artificial proporciona. En el gráfico 7 se registra que a partir de la primera derivada la probabilidad cae drásticamente, siendo en la derivada cuarta completamente nula.

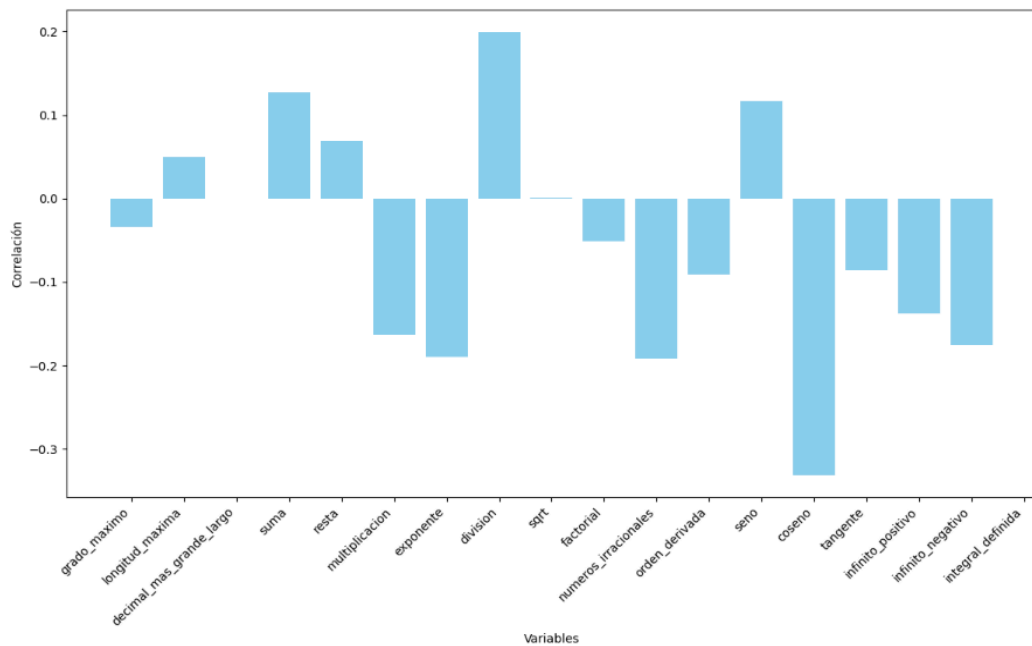


Gráfico 6. Correlación entre variables y acierto de derivadas.

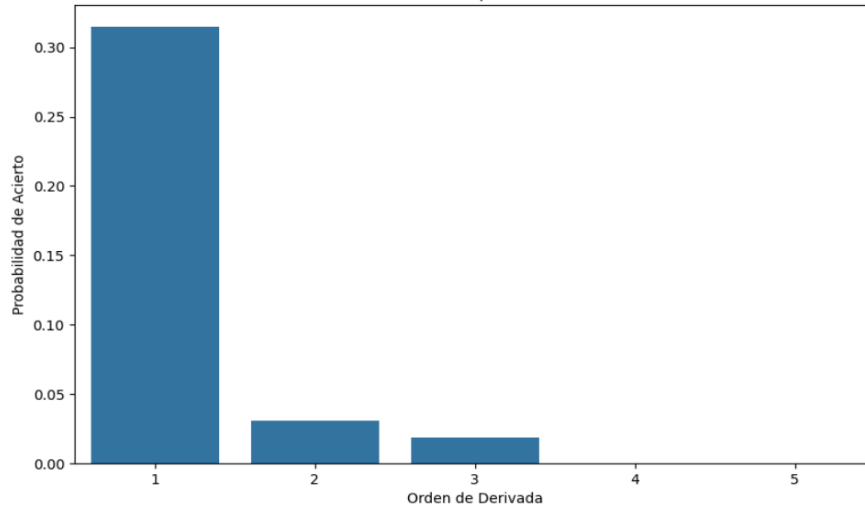


Gráfico 7. Posibilidad de acierto según el orden de la derivada.

- Límites: en este sentido y como se puede observar en el gráfico que se muestra a continuación, las características que influyen positivamente al solicitar a ChatGPT que resuelva un límite son la multiplicación, la división, el seno y el infinito positivo (ver gráfico 8).

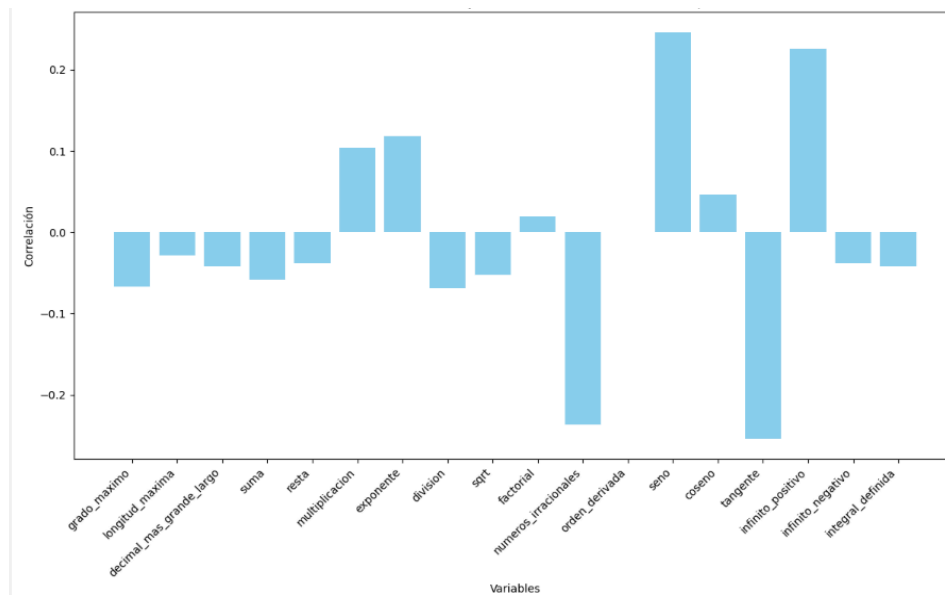


Gráfico 8. Correlación entre variables y acierto de los límites.

Para profundizar en el presente análisis, se han realizado una serie de histogramas que muestran la capacidad de acierto según el tipo de operación matemática. Los resultados que estas figuras arrojan apoyan el resultado del primer gráfico comentado, dado que cuanto mayor tasa de acierto tenga la operación, más se parecerá su gráfico al global.

Los histogramas muestran que la única operación que tiene un mayor porcentaje de acierto respecto a error es la operación simple con 50,91 % frente a 49,09 % respectivamente (ver gráfico 9). Asimismo, el porcentaje de acierto disminuye de manera paulatina conforme se analizan el resto de operaciones matemáticas planteadas en el presente trabajo. De esta forma, el tanto por cien de resoluciones correctas es de un 29,68 % en el caso de las derivadas (ver gráfico 12), un 20,28 % para los límites (ver gráfico 13) y un 19,68 % si se observan las integrales (ver gráfico 11). Así, el menor porcentaje de acierto de ChatGPT, siendo menos de un 10 %, es en las ecuaciones con un 8,49 % (ver gráfico 10).

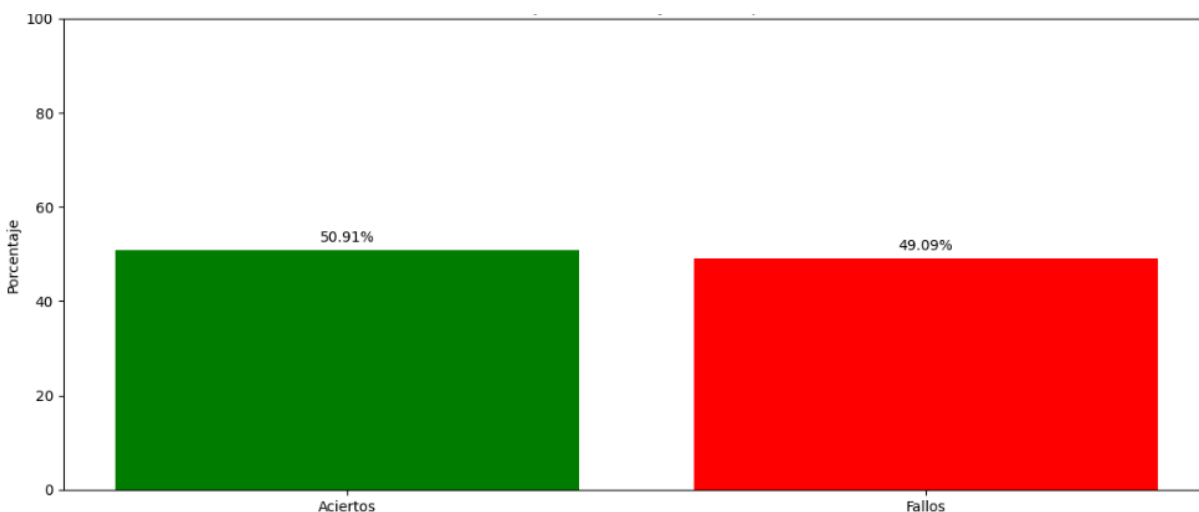


Gráfico 9. Histograma de tasa de acierto y fallo de las operaciones simples.

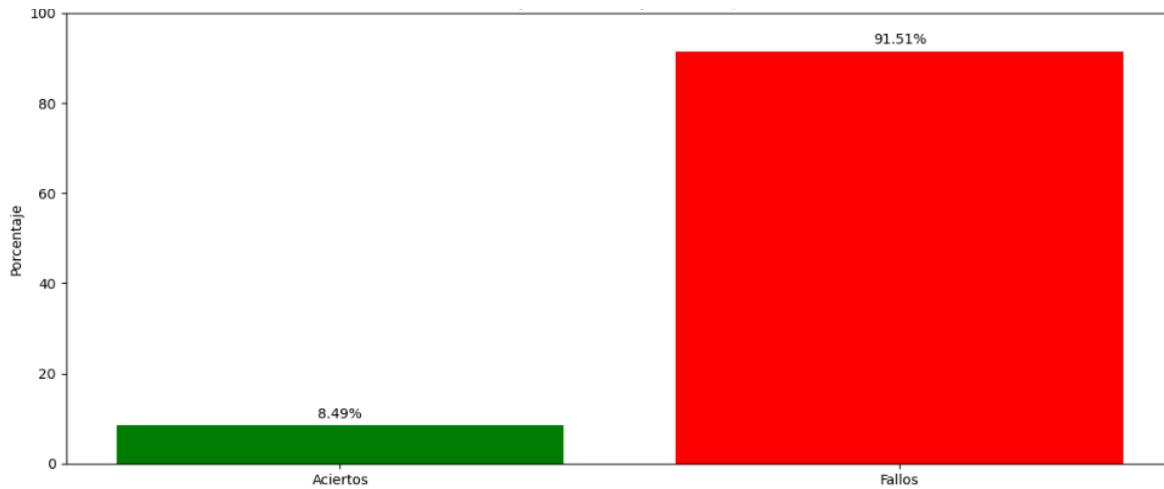


Gráfico 10. Histograma de tasa de acierto y fallo de las ecuaciones.

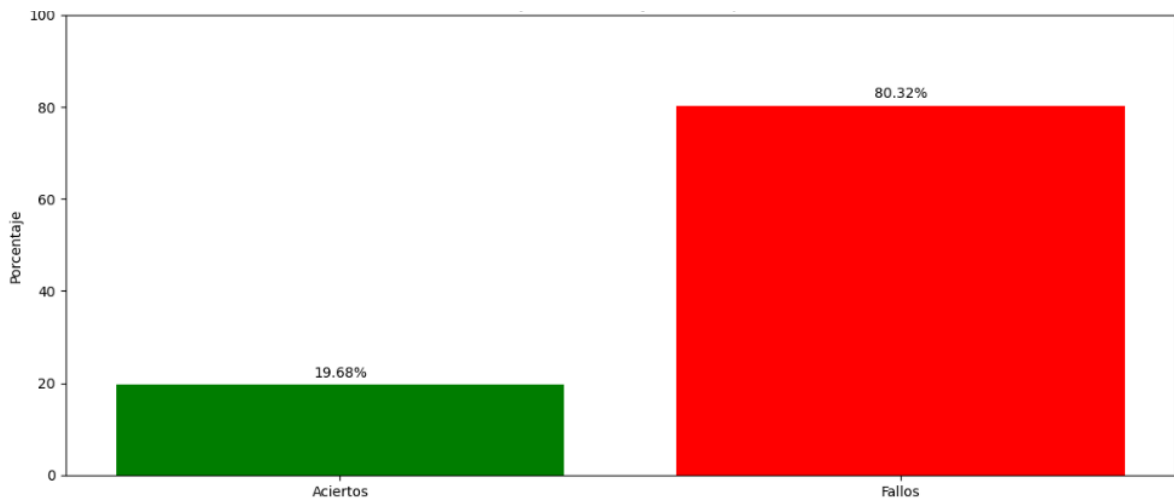


Gráfico 11. Histograma de tasa de acierto y fallo de las integrales.

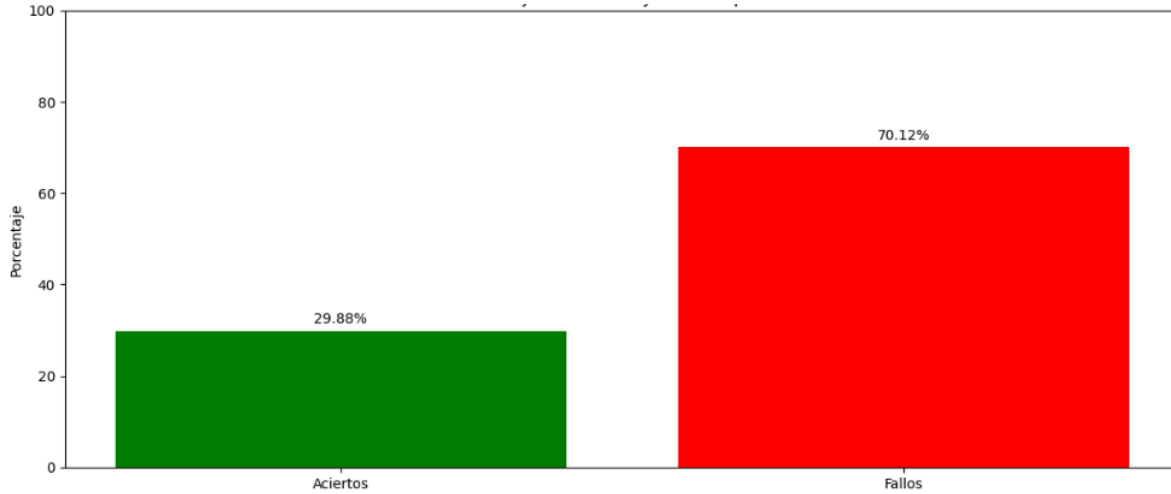


Gráfico 12. Histograma de tasa de acierto y fallo de las derivadas.

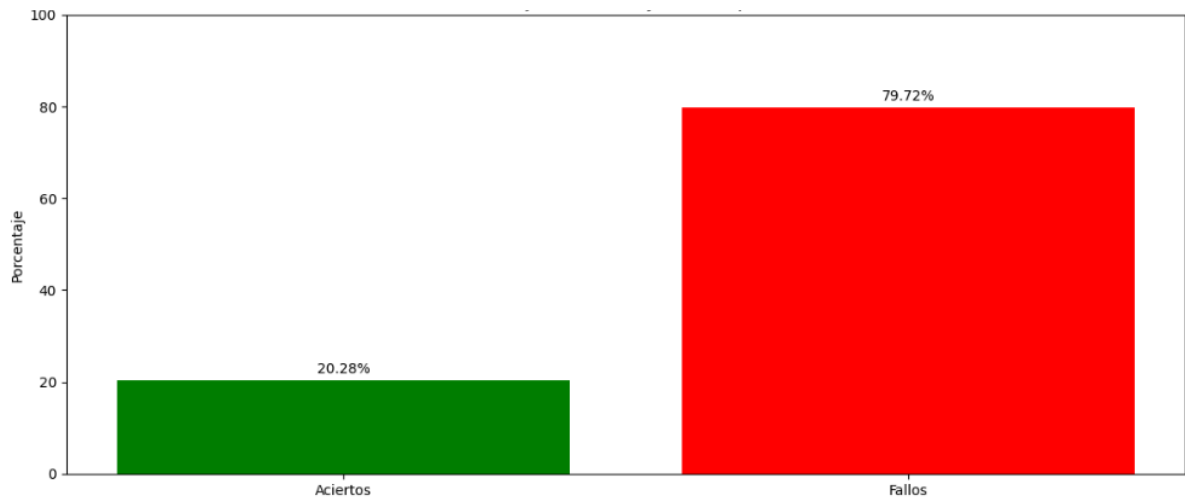


Gráfico 13. Histograma de tasa de acierto y fallo de los límites.

Además, atendiendo a los resultados obtenidos en la presente investigación, se observa una correlación significativa entre las operaciones más sencillas, es decir, aquellas que involucran menos operaciones elementales, y las operaciones que, aún

siendo más complejas, forman parte usualmente de ejemplos en problemas matemáticos, como las tablas de derivadas o integrales que se estudian en el cálculo infinitesimal y su probabilidad de resolución exitosa. Por ejemplo, existe una mayor tasa de acierto en operaciones técnicamente más complejas, como una integral de la función $\frac{1}{x}$ (ver imagen 20), en comparación con una operación simple como $192.34 * 3 + 5$ (ver imagen 21).



Imagen 26. Acierto de ChatGPT en una operación.



Imagen 27. Fallo de ChatGPT en una operación.

Esto se debe a la gran cantidad de ejemplos con los que ha podido ser entrenado el modelo de lenguaje. Se observa que no existe una relación directa entre la dificultad y el fallo en el resultado. En cambio, sí está vinculado con la frecuencia de entrenamiento con operaciones similares, es decir, la cantidad de veces que el lenguaje ha tenido que resolver dicha operación. Cuantas más ocasiones se haya entrenado con ejemplos del mismo estilo, habrá mayor probabilidad de acierto.

Esto último también se produce a la hora de solicitar un código para resolver problemas matemáticos, puesto que manteniendo un nivel de detalle similar, los resultados varían, en gran medida, cuando se le pide un código para resolver problemas matemáticos famosos como los primeros cinco números de Fibonacci respecto una simple operación matemática como la mostrada anteriormente.

Por último, como podemos observar en los ejemplos de las imágenes 22 y 23, ChatGPT es capaz de generar un código adecuado a la petición mostrada, sin embargo, no proporciona resultados correctos en el cálculo de la operación, mostrando la herramienta de inteligencia artificial una solución errónea al igual que en la solicitud

de operaciones. Así, se puede determinar que ChatGPT es capaz de utilizar adecuadamente los recursos del lenguaje solicitado, pero no de la ejecución del mismo.

Generador de código

Elige el lenguaje de programación:

C++ 17

Describe el programa:

Haz un programa llamado que haga sumas dados dos números pasados por parámetro.

Incluye casos de prueba (Opcional):

4234 723211

Generar

Imagen 28. Petición de un programa con argumentos de entrada.

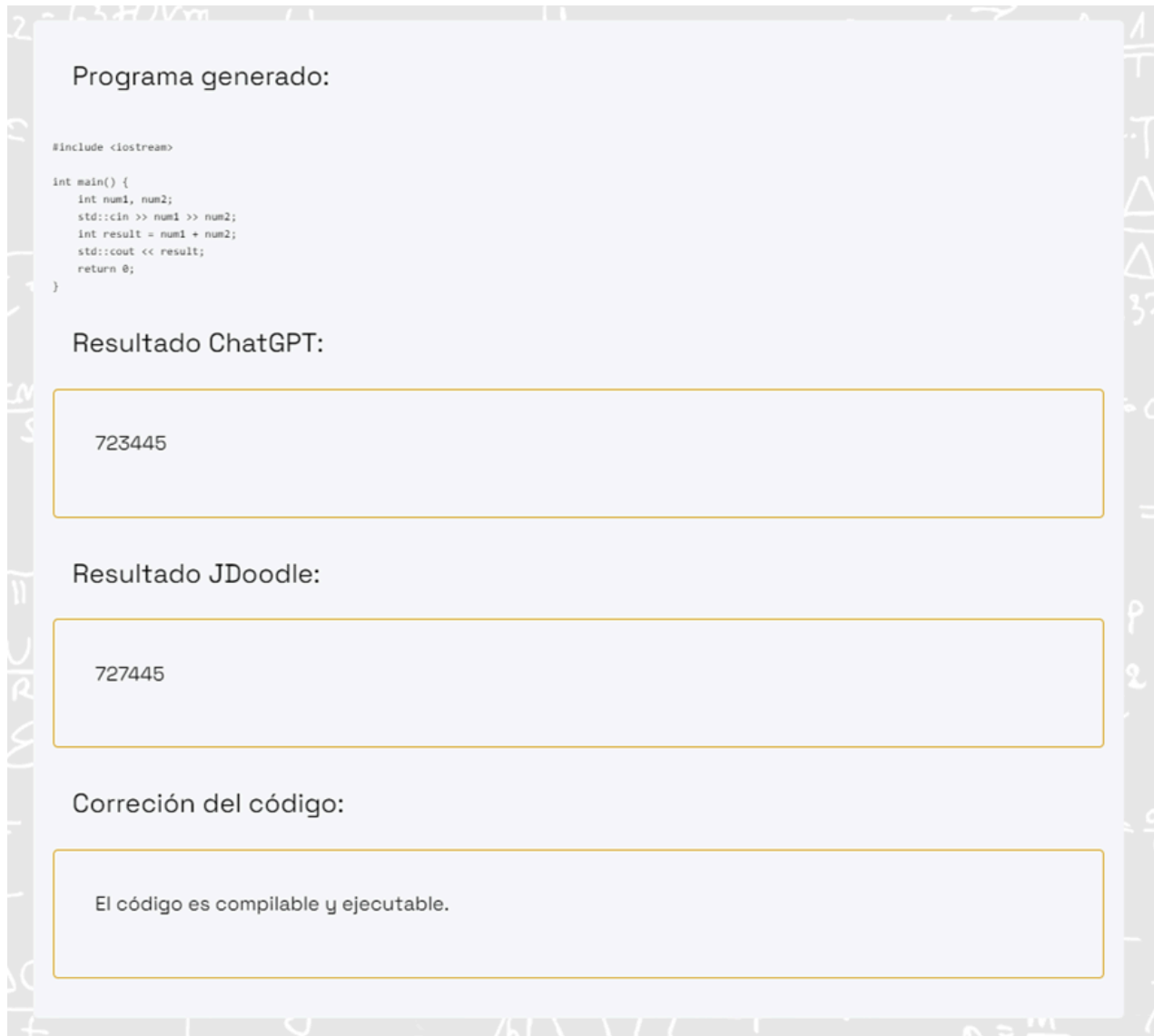


Imagen 29. Solución a la petición de la imagen 28.

5.3. Limitaciones del estudio

Todos los casos de prueba utilizados para la comprobación del trabajo y el análisis posterior han sido generados y supervisados por el equipo investigador. Con una cantidad de datos mayor, se podrían conseguir una mayor precisión en los resultados del análisis.

Asimismo, respecto a las peticiones de licencias, ninguna de las dos empresas encargadas de las APIs utilizadas en este trabajo, OpenAI y JDoodle, han dado acceso a una cuenta sin limitaciones, por lo que se han usado las comerciales abiertas al público. La limitación de peticiones por hora establecidas por las empresas no ha sido posible evitar habiendo sido coste de las mismas asumido por el equipo. Además, cabe resaltar que, para el mantenimiento de datos homogéneos y evitar discrepancias, se ha mantenido siempre una misma versión de ChatGPT la 3.5.

Capítulo 6 - Conclusiones

Como se definió en el apartado de objetivos del proyecto, se buscaba documentar aquellos cálculos matemáticos que la inteligencia artificial ChatGPT es o no capaz de realizar de manera correcta, analizando qué características influyen para pasar del acierto al error. Podemos concluir que, ante los escenarios planteados en el trabajo, se ha podido observar el dispar comportamiento de la inteligencia en las diferentes operaciones matemáticas con respecto a las distintas variables analizadas.

De esta forma, se pone de manifiesto que los modelos de lenguaje de aprendizaje profundo están principalmente configurados para procesar y predecir datos lingüísticos, no para resolver problemas matemáticos. Estos modelos dependen en gran medida de los datos con los que fueron entrenados, haciendo que el éxito en la solución de la operación dependa del entrenamiento del *bot* en vez de la complejidad de la misma. Además, la eficacia de la respuesta mejora significativamente cuando se le proporcionan detalles precisos y específicos en las solicitudes.

Sin embargo, resulta notable que una tecnología como ChatGPT, que no está específicamente diseñada para tareas matemáticas, pueda manejar con éxito una cantidad considerable de operaciones básicas, como sumas, restas, raíces y funciones trigonométricas. Esto contrasta con modelos más especializados, como los de regresión lineal, que están explícitamente diseñados para generar resultados matemáticos precisos, por ejemplo, en la estimación del precio de propiedades inmobiliarias basándose en características específicas de una localidad.

Respecto al código, se concluye que es capaz de proporcionar un programa que compila, pero que al operar los casos de prueba falla con operaciones más sencillas que si se le pregunta directamente. No se ha encontrado ningún caso en el acierto proporcionando un resultado a partir de un código en el que previamente haya fallado mediante operación directa.

A pesar de no estar diseñado para ejecutar operaciones matemáticas complejas, ChatGPT puede potencializar su capacidad mediante la integración de una capa intermedia que maneje estos cálculos de manera independiente, como lo hace la biblioteca SymPy en nuestro estudio. Esta integración permite que ChatGPT no solo comprenda y procese el problema planteado por el usuario, sino que también lo traduzca a un lenguaje de programación como Python. Este enfoque abre la posibilidad a que usuarios sin experiencia previa en programación, puedan resolver una amplia gama de problemas matemáticos de forma rápida y eficiente a través de ChatGPT, representando un avance significativo en términos de productividad y educación, siempre que se aplique correctamente.

Actualmente, la versión 4 de ChatGPT es capaz de ejecutar un programa, para ello se envía el código y este se encarga de ejecutarlo y mostrar la salida. Además, se puede solicitar que resuelva cualquier problema, ya sea matemático o de cualquier otro tipo.

Por ejemplo, solicitando un programa que se encargue de extraer datos de un archivo, enviado por parámetro, la aplicación de ChatGPT proporcionará, a través de un desplegable, el código correspondiente para poder ejecutar dicha funcionalidad.

Esta nueva versión resuelve eficazmente las limitaciones analizadas en el trabajo, dado que si un usuario con unos conocimientos mínimos sobre programación proporciona un

programa para resolver problemas matemáticos puede solicitar a ChatGPT que lo resuelva usando python y la librería SymPy.

6.1. Trabajo a futuro

Viendo como evoluciona el propio ChatGPT en las resoluciones de problemas matemáticos, este estudio marca un camino para poder realizar nuevas investigaciones. Puesto que ante la gran popularidad de los *chatbots* basados en LLM, se podrían plantear nuevos estudios que comparen las diferencias entre estos; como podrían ser LLama2, Gemini, PaLM, etcétera.

Chapter 6 - Conclusions

As defined in the project's objectives section, the aim was to document the mathematical calculations that the artificial intelligence ChatGPT is or is not capable of performing correctly, analyzing which characteristics influence the transition from success to error. We can conclude that, given the scenarios presented in this work, the disparate behavior of the intelligence in various mathematical operations with respect to the different variables analyzed has been observed.

This highlights that deep learning language models are primarily configured to process and predict linguistic data, not to solve mathematical problems. These models heavily depend on the data they were trained with, making the success in solving the operation dependent on the bot's training rather than the complexity of the operation itself. Additionally, the effectiveness of the response significantly improves when precise and specific details are provided in the requests.

However, it is noteworthy that a technology like ChatGPT, which is not specifically designed for mathematical tasks, can successfully handle a considerable number of basic operations, such as addition, subtraction, roots, and trigonometric functions. This contrasts with more specialized models, such as linear regression models, which are explicitly designed to generate precise mathematical results, for example, in estimating real estate prices based on specific characteristics of a location.

Regarding code, it is concluded that it is capable of providing a program that compiles, but when operating the test cases, it fails with simpler operations compared

to when directly asked. No case was found where it provided a correct result from a code where it previously failed via direct operation.

Despite not being designed to execute complex mathematical operations, ChatGPT can enhance its capability through the integration of an intermediate layer that handles these calculations independently, as the SymPy library does in our study. This integration allows ChatGPT not only to understand and process the problem posed by the user but also to translate it into a programming language like Python. This approach opens the possibility for users without prior programming experience to solve a wide range of mathematical problems quickly and efficiently through ChatGPT, representing a significant advance in terms of productivity and education when applied correctly.

Currently, version 4 of ChatGPT can execute a program; for this, the code is sent, and it executes it and displays the output. Additionally, it can be requested to solve any problem, whether mathematical or otherwise. For example, by requesting a program to extract data from a file sent as a parameter, ChatGPT's application will provide, through a dropdown menu, the corresponding code to execute this functionality. This new version effectively resolves the limitations analyzed in the work, given that if a user with minimal programming knowledge provides a program to solve mathematical problems, they can request ChatGPT to solve it using Python and the SymPy library.

Contribuciones Personales

Ignacio Álvarez Caso.

Mis contribuciones al proyecto abarcan diversas áreas, centradas principalmente en el desarrollo de la aplicación, la gestión de la base de datos, la creación de gráficos y la documentación del proyecto. A continuación, se detallan mis aportes específicos:

Desarrollo de la Aplicación

- Creación de Formularios y Validaciones
 - Diseñé y desarrollé los formularios utilizados en la aplicación, asegurando que incluyeran validaciones para facilitar el uso y minimizar errores al ingresar casos de prueba.
 - Implementé mecanismos de validación de datos que garantizan la integridad y precisión de la información ingresada por los usuarios, mejorando así la fiabilidad y facilidad de uso de la aplicación.
- Gestión de la Base de Datos
 - Migraciones y Manejo de Datos: me encargué de la configuración y gestión de la base de datos utilizando SQLite 3, una de las bases de datos predeterminadas en Django.
 - Realicé migraciones de archivos para añadir bancos de pruebas a la base de datos, asegurando que todos los datos necesarios estuvieran disponibles y correctamente estructurados para su uso en la aplicación.
 - Creación de Gráficos y Plantillas.
- Generación de Gráficos: Desarrollé todos los gráficos que se muestran en la aplicación, utilizando datos extraídos de la base de datos. Esto incluyó la utilización de bibliotecas como Matplotlib y Seaborn para visualizar los resultados de manera clara y comprensible.

- Además, creé las plantillas correspondientes para la visualización de estos gráficos y para la interacción con la base de datos, proporcionando una interfaz de usuario intuitiva y funcional.

Documentación del Proyecto

- Redacción del capítulo de Tecnologías Usadas: escribí la sección sobre SQLite 3 en el capítulo de tecnologías usadas, detallando sus ventajas y justificación de su uso en nuestro proyecto.
- Desarrollo del capítulo 4: me encargué de la redacción del capítulo 4, que aborda la estructura general del proyecto. En este capítulo, describí la integración de Django, la implementación de gráficos y la gestión de la base de datos, incluyendo capturas de pantalla de las partes de código más relevantes.
- Incorporación de resultados en el capítulo 5: dedicado a los resultados, añadí las imágenes de los gráficos generados para facilitar la discusión y análisis de los resultados obtenidos.

César Amor Núñez

Mis contribuciones al trabajo se dividen en distintas temáticas, que van desde el desarrollo de la aplicación web hasta la documentación de la memoria y la búsqueda de bibliografía. A continuación, se detallan los aportes concretos:

Desarrollo de la Aplicación

Búsqueda y análisis de la documentación de los *framework* Reflex y Django y su implementación:

Al inicio del desarrollo del TFG busqué documentación e implementé la primera versión de nuestra aplicación mediante el uso de Reflex. Posteriormente, se descartó e hice lo mismo para el *framework* Django mediante el estudio de un curso sobre la plataforma con el fin de hacer la mejor configuración desde el inicio y sentar las bases de las actualizaciones posteriores.

Tras aprender las nociones básicas, implementé la versión inicial de la aplicación, asegurando una buena estructura y aprovechando de forma eficiente los recursos con los que cuenta Django, como la herencia de plantillas HTML.

Planteamiento e implementación del *front-end*:

Me encargué de plantear el diseño de la aplicación y de la implementación de la herencia de plantillas HTML, que parte de una base de la que van heredando las plantillas específicas. A partir de ahí, fui adaptando los diseños a las distintas funcionalidades, utilizando JavaScript en algunas de las páginas. Comenzando con las operaciones matemáticas, el diseño concreto del formulario de relleno de datos de cada operación, el formulario de la petición de un código de programación a ChatGPT y la visualización de los datos del banco de pruebas y de los gráficos de resultados. Además, colaboré en la implementación de los formularios Django y en el desarrollo de las validaciones para cada tipo de operación.

Desarrollo del controlador de código:

Desarrollé la función encargada de manejar la lógica de petición de un código a la inteligencia, implementando una doble llamada a la API de ChatGPT para recibir el código en la primera y usarlo como entrada en la segunda con los casos de prueba para obtener resultados más precisos. Además, colaboré en la función encargada de implementar la API JDoodle para corregir y contrastar el proporcionado por ChatGPT.

Documentación del Proyecto:

Resumen, introducción y marco teórico:

Escribí el resumen del trabajo y el capítulo 1, centrado en la definición de los objetivos principales y secundarios y de exponer la metodología y la motivación por el tema. Me centré en la búsqueda de bibliografía para conocer el estado de la literatura en esta materia y mostrar los principales estudios que relacionan ChatGPT y las matemáticas.

Tecnologías y resultados:

Fui el encargado de escribir las tecnologías desechadas, explicando los motivos y redacté el apartado de "La aplicación" en los resultados, mostrando de forma visual a través de imágenes los principales casos de uso.

Correcciones generales y formato:

Implementé todo el formato de la memoria, la estructura por párrafos y el índice de contenidos, de imágenes y de gráficos y la bibliografía. Además, revisé todos los apartados con el fin de corregir errores, colocar las imágenes y tener una mejor redacción.

Julián Encinas Fuentes

Desarrollo de la Aplicación

Estructuración del Proyecto Django:

Creé la estructura básica del proyecto Django, definiendo los módulos y aplicaciones necesarios para el desarrollo de nuestro sistema. Además, configuré las rutas principales y las vistas iniciales para establecer una base sólida sobre la cual construir las funcionalidades adicionales.

Integración del Código Python Existente:

Analiqué y refactoricé el código Python previamente desarrollado para asegurar su compatibilidad e integración eficiente dentro del framework Django. Implementé módulos específicos que permiten la interacción entre el código existente y las vistas y modelos de Django, facilitando una transición suave y sin fricciones. Además de una base sólida sobre la que implementar evolutivos del proyecto, he trabajado en el *prompt* con el que nos hemos comunicado con la herramienta ChatGPT, lo que nos ha permitido obtener respuestas homogéneas facilitando la comunicación con otras tecnologías como SymPy.

Otra de mis aportaciones ha sido la creación de una capa intermedia de validaciones junto al compañero Ignacio Álvarez. Se trata de una comprobación simple de los argumentos introducidos en nuestras funciones operacionales con el fin de facilitarnos el trabajo y evitar errores que normalmente cometíamos, por ejemplo, la comprobación de paréntesis haciendo que sea obligatorio tenerlos correctamente abiertos y cerrados. En ningún caso el fin de esta capa ha sido la validación enfocada

al uso de la aplicación a nivel de usuario, por lo que solo hemos tenido en cuenta comprobaciones que hemos visto necesarias para nosotros.

Documentación del Proyecto

Mi aportación en la creación de la memoria se centra en:

- La creación inicial del índice con el que pudimos ir completando las diferentes partes que componen el trabajo realizado.
- Objetivos que hemos querido conseguir con este TFG.
- Marco teórico para poder entrar en contexto e introducir a los lectores de la memoria en nuestra investigación.
- Tecnologías utilizadas, donde definimos las herramientas principales utilizadas en este proyecto. De esta forma aportamos información relevante sobre el por qué hemos usado estas tecnologías y dar contexto sobre la forma de desarrollar este trabajo.
- Resultados y conclusiones: mi trabajo en estos puntos se ha centrado en añadir ejemplos y explicaciones sobre los resultados y conclusiones obtenidas.
- Conclusions and Abstract: traducción de las conclusiones y resumen del trabajo.

Además de las contribuciones anteriormente descritas, he realizado tareas de formatear los textos e imágenes y corregir gramaticalmente todas las secciones del proyecto.

Eduardo José Rodelgo Guzmán

Mis contribuciones al proyecto abarcan diversas áreas, centradas principalmente en el desarrollo de la aplicación a nivel de lógica. A continuación, se detallan mis aportes concretos:

Desarrollo de la Aplicación

Análisis de la documentación de la API de ChatGPT y su implementación:

- Analicé la documentación de la API de ChatGPT para entender sus capacidades y limitaciones, lo que permitió una integración eficaz en nuestra aplicación.
- Implementé la API de ChatGPT en la aplicación, asegurando que las solicitudes y respuestas se manejan de manera eficiente y segura.

Planteamiento e implementación de operaciones con SymPy:

- Planteé las operaciones matemáticas que nuestra aplicación web debe realizar utilizando la biblioteca SymPy, asegurándome de cubrir las necesidades específicas de cada operación.
- Implementé todas estas operaciones, ajustando cada una a sus particularidades para garantizar su correcto funcionamiento.

Desarrollo de un comparador de operaciones:

- Creé un comparador para las diferentes operaciones matemáticas que nuestra aplicación debe manejar, permitiendo la comparación de resultados independientemente de la operación enviada.

Implementación de la API Jdoodle para laValidación de Código:

- Implementé la API de JDoodle para la validación del código, teniendo en cuenta las particularidades tanto de las solicitudes como las respuestas, asegurando la correcta interpretación de los resultados.

Desarrollo de Conjunto de Datos y su procesamiento

- Creé un extenso conjunto de datos y desarrollé un código para procesar estos datos utilizando la consola de comandos, estableciendo pautas para su posterior análisis.

Documentación del Proyecto:

Arquitectura:

- En el capítulo 4, describí cómo hemos implementado el controlador utilizando la librería SymPy, el comparador de operaciones y la generación y corrección de código con JDoodle. Detallé la estructura general del proyecto y la interacción entre estos componentes.

Resultados:

- En el apartado 5.2 del capítulo de Resultados, interpreté y analicé todos los gráficos presentados.
- En el 5.3, identifiqué y analicé las principales limitaciones de nuestro estudio.

Conclusiones:

- Redacté las conclusiones finales del proyecto, ofreciendo una visión amplia sobre los diferentes modelos de machine learning y la evolución de ChatGPT hasta su última versión.
- Propuse soluciones para los problemas identificados y sugerí objetivos y trabajos futuros para continuar mejorando el proyecto.

Bibliografía

Bin-Hady, A., 2023. *Exploring the Dimensions of ChatGPT in English Language Learning: A Global Perspective*.

Díaz, J., s.f. *Curso Django*. Píldoras Informáticas, [online]. Disponible en:

<<https://www.pildorasinformaticas.es/>>

Django, s.f. A. *Django Project*. Django, [online]. Disponible en:

<<https://www.djangoproject.com/>>

Django, s.f. B. *Django documentation*. Django, [online]. Disponible en:

<<https://docs.djangoproject.com/en/5.0/>>

Eснаоla, L., 2023. *ChatGPT, una herramienta que no podemos desconocer en nuestra práctica docente*.

Frieder, S., 2023. *Mathematical Capabilities of ChatGPT*.

JDoodle, s.f. *Start coding without the complex setup*. JDoodle, [online]. Disponible en:

<<https://www.idoodle.com/>>

Lorenzo, A., 2023. *ChatGPT, el término más buscado en Google en lo que va de 2023*. El Economista, [online]. Disponible en:

<<https://www.eleconomista.es/tecnologia/noticias/12577619/12/23/chatgpt-el-termino-mas-buscado-en-google-en-lo-que-va-de-2023.html>>

Matplotlib, s.f. *Matplotlib: Visualization with Python*. Matplotlib, [online]. Disponible en:

<<https://matplotlib.org/>>

OpenAI, s.f. *About. OpenAI*, [online]. Disponible en:

<<https://openai.com/about/>>

Parra, S., 2023. *Los avances y controversias de la Inteligencia Artificial en 2023*, National Geographic [online]. Disponible en:

<https://www.nationalgeographic.com.es/ciencia/avances-y-controversias-inteligencia-artificial-2023_21268>

RAE, s.f. *Inteligencia Artificial*. Diccionario de la lengua española, RAE, [online]. Disponible en:

<<https://dle.rae.es/inteligencia#2DxmhCT>>

Red Hat, 2023. *¿Qué es una API y cómo funciona?*. RedHat, [online]. Disponible en:

<<https://www.redhat.com/es/topics/api/what-are-application-programming-interface>>

Reflex, s.f. *Reflex blog*. Reflex, [online]. Disponible en:

<<https://reflex.dev/blog/>>

Rosbaco, N., 2023. *Recursos tecnológicos en matemática: transparencia, visibilidad e invisibilidad en la resolución de un problema*.

Seaborn. s.f. *Seaborn: statistical data visualization*. Seaborn, [online]. Disponible en:

<<https://seaborn.pydata.org/>>

SymPy, s.f. *SymPy*. SymPy, [online]. Disponible en:

<<https://www.sympy.org/en/index.html>>

Tenhundfeld, N., 2023. *ChatGPT as a Tool for Studying Human-AI Interaction in the Wild*.

Torres, F., 2024. *Reflex: La Mejor Alternativa Secreta del Desarrollo Web con Python, Guía Definitiva en Español*. Medium, [online]. Disponible en:

<<https://medium.com/@fnracotorres/reflex-la-mejor-alternativa-secreta-del-desarrollo-web-con-python-gu%C3%ADa-definitiva-en-esp%C3%A1ol-4e982b0d7a41>>

Vive, 2022. *Framework: qué es, para qué sirve y algunos ejemplos*. UNIR, [online]. Disponible en:

<<https://unirfp.unir.net/revista/ingenieria-y-tecnologia/framework/>>

Wardat, Y., 2023. *ChatGPT: A revolutionary tool for teaching and learning mathematics.*