



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MÁSTER

Extensión de herramienta CASE para el desarrollo de sistemas IoT

Interpretación inteligente de casos de uso

Autor

Dúval Carvajal Suárez

Director/es

Miguel J. Hornos Barranco

Carlos Rodríguez Domínguez



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, septiembre 2024



Extensión de herramienta CASE para el desarrollo de sistemas IoT

Interpretación inteligente de casos de uso

Autor

Dúval Carvajal Suárez

Director/es

Miguel J. Hornos Barranco
Carlos Rodríguez Domínguez

Extensión de herramienta CASE para el desarrollo de sistemas IoT: Interpretación inteligente de casos de uso

Dúval Carvajal Suárez

Palabras clave: ajuste fino, entrenamiento, inteligencia artificial, diagramas de casos de uso, diagramas de clases, instrucciones, automatización, modelos, sistemas IoT.

Resumen

El desarrollo de Sistemas basados en Internet de las Cosas IoT (*Internet Of Things*) va aumentando conforme va pasando el tiempo y su trabajo tiene complicaciones al momento de concretar la idea central del proyecto. Como todo Sistema de software debe tener un análisis técnico previo al desarrollo físico del proyecto, los desarrolladores tienden a debatir sobre la creación de diagramas técnicos que permiten a todo el equipo entender el funcionamiento completo del sistema. Existe una herramienta CASE denominada TDDT4IoTS (*Test-Driven Development Tool for IoT-based Systems*), la cual fue mejorada extendiendo su funcionalidad dotándola de mayor inteligencia para desarrollar sistemas IoT. La entrada principal de información por parte de la herramienta son las descripciones del sistema a desarrollar especificadas por los desarrolladores en forma de casos de uso extendidos. Utilizando la API de OpenAI se implementaron las técnicas que brindan sus modelos para analizar estas descripciones e identificarán automáticamente los elementos clave (posibles clases, atributos, relaciones, etc.) que se deben considerar para generar, también de manera automática, un diagrama de clases conceptual, que luego podría refinarse para obtener un diagrama de clases del dominio del diseño de la solución. Además, se logró implementar el ajuste fino de forma los usuarios, mediante la herramienta, tenga la opción de poder indicarle las instrucciones necesarias al modelo que más les interese para mejorar el desarrollo del sistema IoT.

CASE tool extension for IoT systems development: Intelligent interpretation of use cases

Dúval Carvajal Suárez

Keywords: fine-tuning, training, artificial intelligence, use case diagrams, class diagrams, prompts, automation, models, IoT systems.

Abstract

The development of IoT systems has been increasing over time, and its implementation always presents challenges when trying to convey the central idea of the project. Like any software system, it requires a technical analysis prior to the physical development of the project. Developers tend to debate the creation of technical diagrams that allow the entire team to understand the system's full functionality. There is a CASE tool called TDDT4IoTS (Test-Driven Development Tool for IoT-based Systems), which has been enhanced by extending its functionality and equipping it with greater intelligence for developing IoT systems. The tool's primary input consists of system descriptions provided by developers in the form of extended use cases. By leveraging the OpenAI API, techniques from its models were implemented to analyze these descriptions and automatically identify key elements (such as potential classes, attributes, relationships, etc.) that should be considered in order to automatically generate a conceptual class diagram. This diagram could then be refined to produce a design-level class diagram for the solution domain. Additionally, fine-tuning has been implemented, allowing users to provide specific prompts to the model through the tool to improve the development of IoT systems in a manner that best suits their needs.

Yo, **Dúval Carvajal Suárez**, alumno del **MÁSTER DE DESARROLLO DEL SOFTWARE**, con NIE **Z1655535T**, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: *Dúval Carvajal Suárez*

Granada a 13 de septiembre de 2024.

D. **Miguel J. Hornos Barranco**, Profesor del Área de Lenguajes y Sistemas Informáticos del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

D. **Carlos Rodríguez Domínguez**, Profesor del Área de Lenguajes y Sistemas Informáticos del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Extensión de herramienta CASE para el desarrollo de sistemas IoT, Interpretación inteligente de casos de uso***, ha sido realizado bajo su supervisión por **Dúval Carvajal Suárez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 13 de septiembre de 2024.

Los directores:

Miguel J. Hornos Barranco Carlos Rodríguez Domínguez

Yo, **Dúval Carvajal Suárez**, alumno del **MÁSTER DE DESARROLLO DEL SOFTWARE** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI **Z1655535T**, declaro explícitamente que el trabajo presentado es original, entendido en el sentido que no he utilizado ninguna fuente sin citarla debidamente.

Fdo: **Dúval Carvajal Suárez**

Granada, 13 de septiembre de 2024.

Agradecimientos

Agradezco profundamente a mis padres por todo su apoyo, que ha sido fundamental para completar este nuevo nivel de estudios y mejorar tanto mis conocimientos profesionales como mi calidad de vida. También quiero expresar mi gratitud a mis tutores, Miguel Hornos y Carlos Rodríguez, por sus valiosas observaciones a lo largo de este trabajo, que me ayudaron a entregar un proyecto sólido. De igual manera, agradezco a Gleiston Guerrero, docente de la Universidad Técnica Estatal de Quevedo - Ecuador, por brindarme el ánimo necesario para alcanzar esta meta en mi trayecto profesional. Finalmente, extendiendo mi agradecimiento al grupo de estudiantes que dedicaron su tiempo a evaluar y documentar correctamente el trabajo realizado.

Índice general

1. Introducción	1
1.1. Motivación	2
1.2. Objetivos	2
1.2.1. Objetivo general	2
1.2.2. Objetivos específicos	3
1.3. Estructura del trabajo	3
2. Revisión y análisis de la literatura	5
2.1. Modelos de Lenguaje y LLM	5
2.1.1. Modelos base de OpenAI	6
2.1.2. Modelos para procesamiento de texto	7
2.1.3. Conclusiones	9
2.2. Ajustes finos de modelos	11
2.2.1. Ajuste fino específico para tareas	12
2.2.2. Ajuste fino general	12
2.2.3. Ajuste fino específico para dominios	13
2.2.4. Ajuste fino eficiente en parámetros	13
2.2.5. Ajuste fino semi-supervisado	14
2.2.6. Ajuste fino adversarial	14
2.2.7. Ajuste fino enfocado en la seguridad	14
2.2.8. Ajuste basado en instrucciones	15
2.2.9. Ajuste fino en dos etapas	15
2.2.10. Conclusiones	15
2.3. Generación Automática de UML	18
2.3.1. Conclusiones	19
2.4. IA en la ingeniería de software	22
2.4.1. Generación automática de código y prototipos	22
2.4.2. Simulación y búsqueda con IA	22
2.4.3. Optimización y portabilidad de modelos de IA/ML	23
2.4.4. Educación en ingeniería de software con IA generativa	23
2.4.5. Mapeo sistemático de técnicas de IA en ingeniería de software	24
2.4.6. Pruebas de calidad para software con IA	24

2.4.7. Análisis de estructura y distribución de fallos en redes de software	25
2.4.8. Desarrollo de nuevos productos con IA	25
2.5. Conclusiones	26
3. Desarrollo de la extensión de la herramienta	29
3.1. Análisis de requisitos	29
3.1.1. Requisitos funcionales	29
3.1.2. Requisitos No Funcionales	30
3.1.3. Restricciones	31
3.1.4. Recursos Utilizados	31
3.2. Arquitectura	32
3.2.1. Capa del cliente	33
3.2.2. Capa de conexión	33
3.2.3. Capa de negocio	34
3.2.4. Capa de la nube	34
3.2.5. Capa de datos	35
3.3. Desarrollo e Implementación	35
3.3.1. Fase 1: Análisis de requerimientos	36
3.3.2. Fase 2: Diseño de la extensión	38
3.3.3. Fase 3: Configuración de los componentes	42
3.3.4. Fase 4: Desarrollo de los componentes	44
3.4. Metodología para desarrollar la aplicación web	56
3.4.1. Interfaces de la aplicación web	56
3.4.2. Evaluación de la extensión dentro de la aplicación web	64
3.4.3. Ejemplo de proyecto usado para evaluar la extensión de la herramienta	65
4. Resultados y discusión	71
4.1. Distribución de la Usabilidad de la Interfaz	72
4.2. Distribución de la Precisión del Diagrama	73
4.3. Distribución del Tiempo de Generación del Diagrama	73
4.4. Recomendación de la Aplicación	73
4.5. Porcentaje de Modificación del Diagrama	73
4.6. Conclusiones	74
5. Conclusiones y trabajo futuro	75
5.1. Principales conclusiones del trabajo desarrollado	75
5.2. Trabajo futuro	76
Bibliografía	82

Índice de figuras

3.1. Arquitectura de la extensión.	32
3.2. Fases de Desarrollo.	36
3.3. Diagrama de casos de uso de la extensión.	37
3.4. Componentes de la extensión.	41
3.5. Paquetes desarrollados dentro del componente db-repository- tddt4iots.	44
3.6. Paquetes desarrollados dentro del componente ms-core-tddt4iots.	45
3.7. Paquetes desarrollados dentro del componente ms-core-tddt4iots- openai.	46
3.8. Diagrama de secuencia para explicar a detalle el proceso de entrenamiento del modelo mediante el archivo jsonl.	47
3.9. Diagrama de secuencia para explicar a detalle el proceso de crear los archivos jsonl.	49
3.10. Diagrama de secuencia para explicar a detalle el proceso de usar el modelo.	50
3.11. Paquetes desarrollados dentro del componente de python.	51
3.12. Entrenamiento de los modelos de OpenAI.	52
3.13. Uso de los modelos entrenados por OpenAI.	53
3.14. Estructura de los paquetes para el componente armadillo-api.	54
3.15. Interfaz principal de la aplicación.	57
3.16. Opción para agregar la clave secreta de la API de OpenAI.	57
3.17. Formulario para ingresar la clave secreta de OpenAi.	58
3.18. Opcion para configurar la API de OpenAI.	58
3.19. Formulario de configuración de los modelos de OpenAi. Mo- delos base y el uso que le dará el usuario.	58
3.20. Formulario de configuración de los modelos de OpenAi. En- trenamiento de los modelos base y su modelo seleccionado.	59
3.21. Opción para gestionar los modelos base de OpenAI dentro de la herramienta.	60
3.22. Opción para gestionar los entrenamientos de los modelos base de OpenAI dentro de la herramienta.	60
3.23. Estado de un entrenamiento realizado dentro de la herramien- ta y sus opciones disponibles.	61

3.24. Información detallada del entrenamiento.	61
3.25. Prueba del modelo de OpenAI entrenado.	62
3.26. Área para crear un diagrama de casos de uso e indicar si se usará el modelo de OpenAI.	62
3.27. Formulario para ingresar la descripción de cada caso de uso del sistema.	63
3.28. Estado del modelo al analizar los casos de uso.	63
3.29. Diagrama de clases generado por la herramienta y el modelo de OpenAI.	64
3.30. Diagrama de casos de uso para el Sistema de Monitoreo de Cultivos Inteligente con IoT.	69
4.1. Resultados de la encuesta.	72

Índice de tablas

2.1. Comparación de Modelos	10
2.2. Comparativa de Ajustes Finos	17
2.3. Comparación de Técnicas para UML	21
2.4. Comparativa de Técnicas de IA en Ingeniería de Software	27

Capítulo 1

Introducción

El desarrollo de sistemas IoT ha crecido exponencialmente en los últimos años, lo que ha generado una demanda creciente de herramientas que automaticen y optimicen el proceso de diseño y desarrollo de estos sistemas. A medida que las soluciones IoT se vuelven más complejas, la necesidad de minimizar errores humanos y acortar los tiempos de desarrollo es más crítica. Las herramientas CASE (*Computer-Aided Software Engineering*) juegan un papel clave en la mejora de la productividad y en la reducción de errores mediante la automatización de tareas rutinarias.

En este contexto, la inteligencia artificial (IA) emerge como una tecnología estratégica para optimizar el desarrollo de sistemas IoT. La capacidad de la IA para procesar grandes volúmenes de datos y aprender de ellos puede ser utilizada para automatizar tareas como el análisis de requisitos y la generación de diagramas UML a partir de descripciones textuales. Este enfoque no solo mejora la eficiencia del proceso de desarrollo, mitigando la posible comisión de errores humanos, sino que también ayuda a los desarrolladores a concentrarse en tareas más creativas y/o complejas.

El presente trabajo tiene como objetivo principal extender y mejorar la herramienta TDDT4IoTS, integrando técnicas de IA que permitan automatizar tareas clave en el desarrollo de sistemas IoT. A través de la implementación de modelos preentrenados de IA, se pretende mejorar la precisión en el análisis de especificaciones de sistemas y la generación automática de diagramas de clases a partir de dichas especificaciones.

A lo largo de este trabajo se describen los conceptos clave y avances relacionados con la integración de IA en herramientas CASE para el desarrollo de sistemas IoT. En primer lugar, se presenta un análisis del estado del arte de las tecnologías de IA, con especial énfasis en el uso de modelos preentrenados y técnicas de procesamiento de lenguaje natural o NLP (*Natural Language Processing*). Luego, se detalla el proceso de desarrollo e implementación de

la extensión de TDDT4IoTS, abordando los requisitos, la arquitectura del sistema y la integración de los modelos de IA. Finalmente, se exponen los resultados obtenidos a partir de pruebas realizadas con usuarios y se discuten las conclusiones más relevantes del proyecto.

1.1. Motivación

Durante el tiempo que llevo desarrollando software para diversos clientes, he identificado una creciente necesidad de herramientas que reduzcan el tiempo dedicado al análisis de requisitos de software. Esta tarea es una de las más complejas, ya que requiere comprender con precisión las necesidades del cliente para mejorar sus procesos. Aunque los desarrolladores tenemos un dominio claro de las tecnologías disponibles para crear diversos productos, cada cliente presenta necesidades únicas. Sería altamente beneficioso contar con herramientas que permitan automatizar el desarrollo de software a partir de requisitos redactados en lenguaje natural, proporcionando una base sólida sobre la cual construir el resto del proyecto.

Personalmente, también me interesa mucho desarrollar herramientas que agilicen la creación de otros productos. Es interesante observar cómo se pueden automatizar procesos que los programadores realizamos manualmente o de memoria. Considerando el impacto global actual de la IA, se propone aprovechar esta tecnología para mejorar la eficiencia y productividad de los programadores. Utilizar IA para optimizar nuestro trabajo no solo mejorará los resultados, sino que también permitirá a los desarrolladores enfocarse en tareas más estratégicas y creativas.

1.2. Objetivos

Los objetivos de este trabajo se formularon teniendo en cuenta el tiempo, los recursos y la tecnología disponibles.

1.2.1. Objetivo general

Extender y mejorar la herramienta TDDT4IoTS mediante la integración de tecnologías de IA, con el fin de automatizar ciertas tareas en el desarrollo de sistemas IoT, concretamente las relativas a modelado, para mejorar la eficiencia y minimizar errores humanos.

1.2.2. Objetivos específicos

Para lograr el objetivo general que se acaba de indicar, se establecieron los siguientes objetivos específicos:

1. Analizar técnicas de IA a aplicar: Investigar y evaluar las técnicas y tecnologías de IA más adecuadas para integrarlas en la herramienta TDDT4IoTS, enfocadas en el análisis de información textual referente a la especificación de sistemas IoT.
2. Automatizar el análisis de las especificaciones del sistema: Implementar modelos de IA que permitan analizar automáticamente descripciones de sistemas especificadas en forma de casos de uso extendidos, identificando elementos clave, como posibles clases, atributos y relaciones.
3. Generar automáticamente diagramas UML: Crear un módulo dentro de TDDT4IoTS que, a partir del análisis de casos extendidos de uso, genere automáticamente diagramas de clases conceptuales, que luego puedan refinarse para obtener diagramas de clases en UML del dominio del diseño de la solución.
4. Evaluar lo implementado con usuarios reales: Realizar una encuesta para saber la opinión de desarrolladores que hayan usado la extensión implementada e incorporada a la herramienta.

1.3. Estructura del trabajo

El resto de la memoria de este Trabajo Fin de Máster (TFM) está organizada de la siguiente manera:

Capítulo 2: Está compuesto por varias secciones que abordan diferentes aspectos de los modelos de lenguaje grandes o LLM (*Large Language Models*) y ajuste fino, la generación automática de UML, la integración de software con IA. Comienza con una discusión sobre los **modelos de lenguaje** y NLP, incluyendo modelos base de OpenAI y modelos para procesamiento de texto. Luego, profundiza en el ajuste fino de modelos, con subsecciones que detallan ajustes específicos para tareas, dominios, adversarial, entre otros. También se exploran las técnicas de generación automática de diagramas UML y la integración del software con IA, abordando temas como optimización y portabilidad de modelos y mapas sistemáticos de técnicas. Finalmente, se discute el desarrollo de nuevos productos con IA, proporcionando una visión amplia y detallada de las tecnologías y metodologías actuales en el campo de la IA.

Capítulo 3: Abarca el desarrollo del trabajo, comenzando con un análisis de requisitos, así como restricciones y recursos utilizados. Luego, aborda la arquitectura del sistema, cubriendo las capas del cliente, conexión, negocio, nube y datos. A continuación, se describe el proceso de desarrollo e implementación, estructurado en cuatro fases: análisis de requisitos, diseño, configuración y desarrollo de componentes. Finalmente, se presenta una metodología para el desarrollo de la aplicación web, destacando las interfaces y la evaluación de la extensión dentro de la aplicación.

Capítulo 4: Presenta los resultados obtenidos a partir de una encuesta realizada a 15 estudiantes que utilizaron la aplicación mejorada. Los resultados se analizaron y se visualizaron mediante gráficos, abordando temas como la usabilidad de la interfaz, la precisión de los diagramas generados, el tiempo de generación de los mismos y el porcentaje de modificaciones necesarias. A partir de estos datos, se ofrecen recomendaciones para mejorar la aplicación, evaluando su desempeño en el contexto del desarrollo de sistemas IoT.

Capítulo 5: Este capítulo concluye que los objetivos planteados al inicio del proyecto fueron cumplidos satisfactoriamente. Se destaca que la aplicación ayudó en el desarrollo de sistemas IoT y permitió una mejor comprensión y generación de diagramas de clases a partir de descripciones de casos de uso. Además, se discuten posibles líneas de mejora para futuras versiones del sistema y su potencial aplicabilidad en otros contextos de IoT.

Capítulo 2

Revisión y análisis de la literatura

En este capítulo se presenta una revisión de los trabajos relacionados que constituyen el fundamento teórico y práctico de esta investigación. Se examinan los avances en modelos de lenguaje y NLP, el uso de técnicas de *fine-tuning* para adaptar modelos pre-entrenados a tareas específicas, y las metodologías para la generación de diagramas UML a partir de texto natural. También se analizan las aplicaciones de IA en la ingeniería de software y se discuten las técnicas de evaluación y validación de modelos generativos. Esta revisión proporciona el contexto necesario para entender la relevancia y el impacto del uso de la IA en la generación automática de un diagrama de clases a partir de descripciones de casos de uso extendidos.

2.1. Modelos de Lenguaje y LLM

En esta sección se mencionarán algunos modelos de lenguaje entrenados previamente para comprender, generar y manipular el lenguaje humano. Los modelos desarrollados por OpenAI han revolucionado el campo del procesamiento del lenguaje natural, permitiendo entender y generar texto con gran precisión y coherencia. Los trabajos seleccionados mencionan múltiples aplicaciones y mejoras de estos modelos en diversas áreas. Desde la comparación entre respuestas humanas y generadas por IA, hasta la identificación de microservicios y la detección de información sensible en documentos textuales, los estudios destacan la versatilidad y el impacto transformador de estos avances tecnológicos.

2.1.1. Modelos base de OpenAI

OpenAI ofrece una gama de modelos avanzados diseñados para tareas específicas en el procesamiento del lenguaje natural (NLP). Estos modelos tienen capacidades únicas y se utilizan en diversas aplicaciones, desde la generación de texto hasta la identificación de patrones complejos en datos. A continuación, se describen algunos de los modelos base más destacados y se presentan casos de estudio que demuestran su uso y eficacia.

GPT-4 es el modelo más reciente y avanzado de OpenAI, conocido por su capacidad mejorada para comprender y generar tanto texto como código. Es particularmente útil en aplicaciones que requieren alta precisión y comprensión contextual, como chatbots avanzados, asistentes virtuales y generación de contenido complejo. Una variante optimizada, GPT-4 Turbo, ofrece un rendimiento mejorado y es más rentable porque disminuye el consumo de recursos de los servidores que ejecutan el modelo y costos más bajos por cada token procesado, ideal para aplicaciones que necesitan respuestas rápidas y eficientes sin sacrificar la calidad [1]. Chen et al. [2], examinan el uso de GPT-4 para la creación de modelos de objetivos que permiten definir y representar visualmente los objetivos de los usuarios o del sistema en el contexto de la ingeniería de requisitos. A través de una serie de experimentos, se demuestra que GPT-4 retiene un conocimiento considerable sobre el modelado de objetivos, aunque con ciertas limitaciones en precisión y especificidad. La investigación resalta la necesidad de múltiples iteraciones y retroalimentación interactiva para mejorar la calidad de los modelos generados.

GPT-3.5 es una versión mejorada del popular modelo GPT-3, con capacidades avanzadas para entender y generar lenguaje natural y código. GPT-3.5 Turbo es el modelo más capaz y rentable dentro de esta categoría, optimizado para tareas de chat y generación de respuestas precisas y coherentes. También está disponible el modelo GPT-3.5 Turbo Instruct, utilizado principalmente para tareas de completado de texto, similar al modelo text-davinci-003, pero con mejoras en la precisión y capacidad de respuesta [1]. Lucke y Frank [3] examinan las diferencias y mejoras entre GPT-3.5 y GPT-4. Los autores destacan cómo GPT-3.5 es capaz de manejar tareas de generación de texto con alta coherencia y precisión, pero GPT-4 ofrece mejoras significativas en términos de comprensión contextual y generación de código. Este estudio proporciona una visión detallada de las capacidades y aplicaciones de ambos modelos en diversos contextos de NLP.

GPT-3 Este modelo fue entrenado con enormes cantidades de datos textuales para generar respuestas y realizar tareas complejas de NLP. GPT-3 puede llevar a cabo una amplia variedad de tareas, como la traducción de idiomas, la redacción de textos, el resumen de información, entre otros, gracias a su capacidad para entender y generar lenguaje humano de manera

coherente [4]. Además, Scheschenja [4] indica que aunque es poderoso, tiene limitaciones, como la posibilidad de generar información incorrecta o “alucinaciones”, cuando presenta respuestas que no están basadas en hechos reales. Sin embargo, sigue siendo un avance significativo en el campo de la inteligencia artificial, particularmente útil en áreas como la educación, el servicio al cliente, y el análisis de texto.

Codex es un modelo derivado de GPT-3, diseñado específicamente para la generación de código y asistencia en tareas de programación. Puede interpretar y generar código en múltiples lenguajes de programación, lo que lo hace ideal para desarrollar aplicaciones, depurar código y automatizar tareas de programación. Codex es la base de herramientas como GitHub Copilot, que asisten a los desarrolladores sugiriendo líneas de código y completando funciones de manera autónoma [1]. Ságođi et al. [5], exploran cómo GPT-4 y Codex pueden identificar y corregir vulnerabilidades en software real. Los resultados indican que Codex, al estar especializado en tareas de programación, muestra una alta precisión en la identificación y corrección de errores, mejorando significativamente la seguridad y eficiencia del desarrollo de software.

2.1.2. Modelos para procesamiento de texto

Gokcimen y Das [6] exploran la similitud entre respuestas generadas por humanos y las producidas por modelos de lenguaje, específicamente ChatGPT. Utilizan una serie de preguntas para comparar las respuestas de varios modelos basados en transformadores, incluyendo *RoBERTa Base OpenAI Detector*, que mostró el mejor rendimiento en términos de generar respuestas similares a las humanas. Los autores destacan la capacidad de ChatGPT para producir respuestas coherentes y relevantes, acercándose mucho a la capacidad humana en la generación de lenguaje natural.

Este estudio no solo proporciona una comparación detallada entre los modelos, sino que también ofrece una perspectiva sobre cómo los avances en los modelos de lenguaje pueden ser aprovechados en aplicaciones prácticas. La investigación se basa en el uso del conjunto de datos HC3, diseñado para comparar las respuestas de ChatGPT con las de expertos humanos, abarcando una amplia gama de temas. Los resultados sugieren que, aunque existen diferencias entre las respuestas humanas y las generadas por IA, los modelos de lenguaje como ChatGPT están cada vez más cerca de replicar la complejidad y la sutileza de la comunicación humana. Esto subraya la importancia de seguir mejorando estos modelos para aplicaciones en diversas áreas, desde el servicio al cliente hasta la educación y la salud.

Muralitharan y Arumugam [7] presentan un algoritmo innovador que combina las capacidades del modelo BERT y LSTM para la detección de in-

formación sensible en documentos textuales. Este enfoque híbrido utiliza la robustez de BERT para el entendimiento contextual y la secuencialidad de LSTM para manejar largas secuencias de texto, logrando así una alta precisión en la identificación de datos sensibles. Los autores aplicaron su modelo en diversos conjuntos de datos, demostrando que su algoritmo supera a los métodos tradicionales en términos de precisión y capacidad de detección.

La investigación destaca cómo la combinación de diferentes arquitecturas de modelos de lenguaje puede mejorar significativamente las tareas de NLP, especialmente en contextos donde la privacidad y la seguridad de la información son críticas. Este trabajo no solo proporciona una herramienta efectiva para la detección de datos sensibles, sino que también abre nuevas vías para el desarrollo de algoritmos más sofisticados que puedan manejar tareas complejas en NLP con mayor eficiencia y precisión. La relevancia de este estudio se magnifica al considerar aplicaciones en industrias que manejan grandes volúmenes de datos personales y sensibles, como la salud y las finanzas, donde la precisión en la detección de información crítica es de suma importancia.

Xiubin Zhu et al. [8] presentan un método innovador para explicar modelos de caja negra mediante el uso de modelos de sustitutos locales basados en reglas difusas. Los modelos de sustitutos son aproximaciones más simples que imitan el comportamiento de modelos complejos, permitiendo su interpretación. Esta investigación tiene como objetivo proporcionar interpretaciones claras y comprensibles del comportamiento de modelos complejos, permitiendo a los usuarios entender mejor las razones detrás de ciertas decisiones. El enfoque desarrollado se aplica en varios escenarios, demostrando que los modelos de sustitutos pueden generar explicaciones precisas y útiles sin comprometer la precisión del modelo original.

Este estudio se enmarca dentro del creciente interés por la interpretabilidad en el aprendizaje automático, abordando uno de los mayores desafíos en el campo de la IA. Al ofrecer una metodología para generar explicaciones comprensibles, esta investigación no solo mejora la transparencia de los modelos de caja negra, sino que también facilita su adopción en sectores donde la interpretabilidad es crucial, como la medicina y las finanzas. Además, se relaciona con otros trabajos que buscan mejorar la precisión y comprensión de los modelos de lenguaje y NLP, destacando la importancia de desarrollar herramientas que permitan a los usuarios confiar y entender las decisiones automatizadas.

Hay una serie de trabajos [9][10][11], que exploran diferentes aplicaciones de la IA generativa y herramientas de modelado en la ingeniería de software. El primer estudio [9] investiga el uso de ChatGPT en tareas de modelado de software, específicamente en la creación de diagramas UML. Los autores identifican limitaciones significativas en la capacidad de ChatGPT para

generar modelos precisos y consistentes, destacando problemas de sintaxis, semántica y escalabilidad. A pesar de estas limitaciones, el estudio subraya el potencial de ChatGPT y la necesidad de desarrollar metodologías que aseguren la precisión y consistencia en modelos más grandes y complejos.

Babaj et al. [10] propone un enfoque para la identificación de microservicios utilizando un modelo de transformador profundo de NLP en desarrollos greenfield. Los modelos transformadores usan un mecanismo de atención para procesar secuencias de datos de manera más eficiente. Este enfoque mejora la identificación y arquitectura de microservicios, facilitando el desarrollo de aplicaciones más robustas y escalables. Este estudio resalta el potencial de los modelos de transformadores en la ingeniería de software, particularmente en el diseño y desarrollo de microservicios, enfatizando la importancia de estas tecnologías para mejorar la eficiencia y precisión en el desarrollo de software.

Mohd et al. [11] propone un método que combina UML y un marco *non-functional requirements* (NFR) o en su traducción al español requisitos no funcionales para analizar los requisitos de un sistema de información. Este método sistemático utiliza diagramas UML para representar visualmente los elementos del sistema y sus interacciones, mejorando la claridad y precisión en la definición de requisitos. La investigación destaca la importancia de combinar enfoques basados en IA y metodologías tradicionales de ingeniería de software para ofrecer soluciones más robustas y efectivas.

2.1.3. Conclusiones

Al relacionar los estudios [9][10][11], se evidencia una tendencia hacia el uso de herramientas de modelado avanzado para enfrentar los desafíos en la ingeniería de software, optimizando tanto la precisión como la eficiencia en el diseño de sistemas complejos. La combinación de IA generativa y técnicas de modelado tradicionales promete mejorar significativamente la calidad y escalabilidad del desarrollo de software, destacando la necesidad de una colaboración continua entre la comunidad de modelado de software y los desarrolladores de IA.

Para comprender mejor el uso de los modelos utilizados en los trabajos mencionados, la tabla 2.1 muestra una comparación de todos los modelos evaluando algunas de sus capacidades:

- **Texto:** Referencia a la capacidad que tiene el modelo para generar texto.
- **Código:** Referencia para la capacidad de generar código.
- **NLP:** Referencia sobre la capacidad de precisión en el procesamiento

NLP.

- **Velocidad:** Referencia hacia la velocidad de respuesta del modelo
- **Sensibilidad:** Referencia a la capacidad del modelo en detectar datos de entrada sensibles.

Ademas, para simplificar la visualización de la tabla en la memoria de este trabajo, se utilizaron los siguientes símbolos para identificar las capacidades que forman parte de cada modelo:

- ✓: Representa que el modelo es muy adecuado para esta capacidad.
- ✗: Representa que el modelo no aplica para esta capacidad.
- —: Representa que el modelo tiene una capacidad moerada.

Tabla 2.1: Comparación de modelos en base a sus capacidades clave

Modelo	Texto	Código	NLP	Velocidad	Sensibilidad
GPT-4	✓	✓	✓	—	✗
GPT-3.5	✓	✓	✓	✓	✗
GPT-3	✓	—	—	✓	✗
Codex	—	✓	—	✓	✗
Robert	—	✗	✓	—	✓
BERT/LSTM	—	✗	✓	—	✓

En conclusión, teniendo en cuenta los datos analizados. El modelo que mas se ajusto para que se logren cumplir los objetivos de este trabajo fue el modelo GPT-3.5 dado que cumple con la mayor cantidad de capacidades mencionadas. Su capacidad para procesar texto natural permita generar representaciones estructuradas como el archivo JSON que necesita la herramienta TDDT4IoTS y se encargue de graficar el diagrama de clases. En comparación con otros modelos como GPT-4, que aunque es más potente, presenta una mayor complejidad y puede ser más lento, GPT-3.5 ofrece un rendimiento robusto en tareas de generación de texto y manejo de información contextual con una velocidad más adecuada para entornos donde se necesitan respuestas rápidas y eficientes.

Se decidió que GPT-3.5 era más preciso y consistente que GPT-3 en la generación de texto, lo cual resultaba crucial para el proyecto, que dependía de la correcta interpretación de las especificaciones escritas en lenguaje natural del sistema a desarrollar en forma de casos de uso extendidos para transformarlos en diagramas de clases. Aunque GPT-3 ofrecía un rendimiento aceptable, su capacidad para procesar lenguaje con alta precisión era inferior,

lo que podría haber afectado la calidad de los diagramas generados. Otros modelos, como Codex, estaban más orientados a la generación de código, pero no tenían la misma capacidad en el procesamiento de lenguaje natural que GPT-3.5. Además, modelos como Robert y BERT/LSTM, aunque fuertes en la detección de datos sensibles, no eran adecuados para la creación de diagramas a partir de texto. Por estas razones, GPT-3.5 fue elegido, ya que ofrecía una mejor combinación de precisión, rapidez y capacidad de interpretación para satisfacer las necesidades del TFM.

2.2. Ajustes finos de modelos

El *fine-tuning*, o ajuste fino, es una técnica esencial en el desarrollo y optimización de modelos de lenguaje pre-entrenados. Esta técnica permite adaptar los modelos a tareas específicas o mejorar su rendimiento general utilizando conjuntos de datos adicionales y específicos. Existen varios tipos de ajuste fino que se han desarrollado para maximizar la eficiencia y la efectividad de estos modelos en diversas aplicaciones. Entre los tipos más destacados se encuentran los siguientes:

- Específico para tareas
- General
- Específico para dominios
- Eficiente en parámetros
- Semi-supervisado
- Adversarial
- Enfocado en la seguridad
- Basado en instrucciones
- Dos etapas

Cada uno de estos enfoques tiene sus propias ventajas y se aplica en contextos diferentes para lograr resultados óptimos. A continuación, se analizan estudios que han implementado estos tipos de ajuste fino, destacando cómo cada uno ha contribuido a mejorar el rendimiento de los modelos de lenguaje en diversas tareas.

2.2.1. Ajuste fino específico para tareas

El ajuste fino específico para tareas se centra en adaptar modelos pre-entrenados para realizar de manera óptima tareas particulares. En el trabajo realizado por Ságodi et al. [5], los autores utilizan esta técnica para ajustar GPT-4 a la tarea de detección y corrección de vulnerabilidades en software real. El proceso de ajuste fino involucró el uso de un conjunto de datos extenso de código con vulnerabilidades etiquetadas. A través de iteraciones de entrenamiento, lograron que el modelo identificara y sugiriera correcciones precisas, mejorando la seguridad y eficiencia del software. Este enfoque específico permite que el modelo se enfoque en las particularidades de la tarea, mejorando significativamente su eficacia y reduciendo el tiempo necesario para la corrección de errores.

Un enfoque similar se observa en el trabajo de Zhang et al. [12] donde los autores aplican el ajuste fino para mejorar la predicción de la corrección de parches de software. Utilizando datos históricos de parches y sus resultados, adaptan el modelo pre-entrenado para predecir si un parche corregirá correctamente un error. Este ajuste fino específico para tareas permitió mejorar la precisión en la predicción, lo que es crucial para la estabilidad y seguridad del software. Los resultados demostraron una mejora significativa en la tasa de éxito de los parches aplicados, evidenciando la eficacia del ajuste fino en tareas críticas de mantenimiento de software.

Por su parte, Li et al. [13] han revisado diversas técnicas de ajuste fino aplicadas en la generación de texto. Destacan el uso del ajuste fino específico para tareas para mejorar la calidad de los textos generados en diferentes contextos, reafirmando la utilidad de este enfoque para optimizar el rendimiento en tareas específicas de generación de texto. Analizaron modelos ajustados para tareas como redacción creativa, generación de respuestas en chatbots, y creación de contenido técnico, mostrando mejoras sustanciales en la coherencia y relevancia de los textos generados.

2.2.2. Ajuste fino general

El ajuste fino general se enfoca en mejorar el rendimiento de los modelos pre-entrenados en una variedad de tareas. Lucke y Frank [3] comparan GPT-3.5 y GPT-4, destacando cómo ambos modelos han sido ajustados finamente para mejorar su capacidad de generación de texto y comprensión contextual. Este ajuste general permite que los modelos sean aplicables a múltiples tareas de NLP sin la necesidad de ajustes adicionales específicos para cada tarea. Los autores encontraron que el ajuste fino general mejoró significativamente la fluidez y coherencia de las respuestas generadas, haciéndolas más cercanas a la producción humana en una amplia gama de contextos.

Cheng et al. [14] mencionan diferentes dominios y aplicaciones del ajuste fino general. Los autores destacan cómo el ajuste fino general ha permitido a los modelos pre-entrenados alcanzar un rendimiento superior en una amplia gama de tareas, demostrando su versatilidad y eficiencia. Este enfoque se ha utilizado para mejorar modelos en aplicaciones como la traducción automática, la clasificación de textos y la generación de resúmenes, mostrando mejoras consistentes en precisión y relevancia.

2.2.3. Ajuste fino específico para dominios

El ajuste fino específico para dominios se aplica para adaptar los modelos a dominios específicos, mejorando su rendimiento en tareas relacionadas con dicho dominio. Chen et al. [2] muestran que se puede utilizar GPT-4 para la creación de modelos de objetivos en la ingeniería de requisitos. Al ajustar finamente GPT-4 con datos específicos del dominio de ingeniería de requisitos, el modelo logra comprender mejor las estructuras y terminologías específicas utilizadas, mejorando su rendimiento en esta tarea particular. El estudio muestra que el modelo podía generar modelos de objetivos con una precisión y coherencia mejoradas, facilitando el trabajo de los ingenieros de requisitos.

Cheng et al. [14] también abordan el ajuste fino específico para dominios, destacando cómo estas técnicas permiten a los modelos pre-entrenados adaptarse eficazmente a las necesidades de diferentes industrias y aplicaciones. Este enfoque facilita la integración de modelos de IA en contextos específicos, mejorando su relevancia y utilidad. Por ejemplo, se menciona la adaptación de modelos para el sector de la salud, donde se lograron mejoras en la precisión del diagnóstico y la generación de informes médicos mediante el ajuste fino con datos clínicos específicos.

2.2.4. Ajuste fino eficiente en parámetros

El ajuste fino eficiente en parámetros busca optimizar el proceso de ajuste fino minimizando los recursos necesarios mientras se mantiene o mejora el rendimiento del modelo. Liu et al. [15] presentan métodos para ajustar finamente los modelos sin necesidad de ajustar todos los parámetros. Utilizan técnicas como la congelación de capas y la adaptación de solo ciertas partes del modelo, ahorrando recursos computacionales y tiempo. Este enfoque es especialmente útil en entornos con recursos limitados, permitiendo que los modelos logren un alto rendimiento sin incurrir en altos costos computacionales. Los resultados del estudio mostraron que los métodos eficientes en parámetros podían mantener un alto nivel de precisión y relevancia, demostrando la eficacia de estos enfoques en aplicaciones prácticas.

2.2.5. Ajuste fino semi-supervisado

El ajuste fino semi-supervisado combina datos etiquetados y no etiquetados para mejorar el rendimiento del modelo. Se ha propuesto una técnica [16] que aprovecha tanto datos supervisados como no supervisados para el ajuste fino. Este enfoque permite que el modelo aprenda de manera más eficiente, mejorando su capacidad de generalización y rendimiento en tareas de NLP. Los experimentos realizados demostraron que el uso de datos no etiquetados en combinación con datos etiquetados permitía al modelo mejorar su precisión en tareas como la clasificación de textos y la detección de entidades nombradas, destacando la utilidad del enfoque semi-supervisado. Los autores también resaltan cómo esta técnica puede reducir significativamente los costos y el tiempo asociados con la obtención de grandes cantidades de datos etiquetados, haciendo el ajuste fino más accesible y eficiente.

2.2.6. Ajuste fino adversarial

El ajuste fino adversarial mejora la robustez del modelo mediante la exposición a ejemplos adversariales durante el entrenamiento. Lee y Jang [17] utilizan esta técnica para mejorar la detección de texto generado por máquinas. El modelo se ajusta finamente utilizando técnicas adversariales, lo que ayuda a que el modelo se vuelva más robusto ante intentos de generar texto que engañe a los sistemas de detección. Los autores encontraron que el modelo ajustado adversarialmente podía detectar con mayor precisión textos generados por IA, reduciendo las tasas de falsos positivos y negativos, lo cual es crucial para aplicaciones en seguridad y vigilancia digital. Este enfoque no solo mejora la precisión, sino que también incrementa la capacidad del modelo para adaptarse a nuevos tipos de ataques adversariales, haciendo que los sistemas de detección sean más resilientes frente a las evoluciones en las tácticas de generación de texto adversarial.

2.2.7. Ajuste fino enfocado en la seguridad

El ajuste fino enfocado en la seguridad se enfoca en adaptar el modelo para generar código seguro, minimizando vulnerabilidades comunes. Li et al. [18] ajustan finamente un modelo de lenguaje grande para la generación segura de código, utilizando un conjunto de datos de código seguro. Este enfoque prioriza prácticas de codificación seguras, mejorando la seguridad del código. Los resultados mostraron que el modelo ajustado podía generar código que cumplía con altos estándares de seguridad, reduciendo significativamente las vulnerabilidades comunes en el desarrollo de software. Este enfoque fomenta mejores prácticas de codificación, proponiendo soluciones basadas en estándares de seguridad que crean un entorno de desarrollo seguro.

2.2.8. Ajuste basado en instrucciones

El ajuste basado en instrucciones [19] ajusta el modelo para seguir instrucciones específicas de manera más efectiva. Este ajuste mejora la coherencia y precisión del modelo, al seguir instrucciones dadas por el usuario. El estudio mostró que el ajuste basado en instrucciones ayudó a reducir la ambigüedad y a mejorar la precisión de las respuestas generadas, haciendo que el modelo fuera más fiable y útil para tareas específicas. Además, este tipo de ajuste fino puede facilitar la personalización del modelo para distintos usuarios y aplicaciones, permitiendo que el modelo responda de manera más adecuada a las necesidades y expectativas particulares de los usuarios.

2.2.9. Ajuste fino en dos etapas

Zhang y Hu [20] proponen esta técnica permite una mayor flexibilidad y eficiencia en el ajuste fino, utilizando tanto la estructura original del modelo como redes especializadas en tareas. Este enfoque mejora el rendimiento del modelo en tareas de clasificación de texto al combinar diferentes estrategias de ajuste. Los resultados mostraron que el ajuste fino en dos etapas permitía al modelo adaptarse mejor a las características específicas de las tareas, logrando una mayor precisión y eficiencia. Este método también facilita la incorporación de nuevas características o datos en el modelo sin necesidad de un reentrenamiento completo, optimizando el tiempo y los recursos necesarios para mantener y mejorar el rendimiento del modelo.

2.2.10. Conclusiones

Como conclusión de esta sección podemos resaltar que estos estudios demuestran cómo diferentes tipos de ajuste fino pueden ser aplicados para optimizar modelos de lenguaje pre-entrenados en diversas tareas y dominios. Desde ajustes específicos para tareas particulares hasta enfoques generales y eficientes en parámetros, cada técnica de ajuste fino ofrece ventajas únicas que mejoran el rendimiento y la aplicabilidad de los modelos de lenguaje en múltiples contextos. Estas investigaciones no solo subrayan la importancia del ajuste fino en el desarrollo de modelos de lenguaje más precisos y eficientes, sino que también destacan las diversas estrategias que pueden ser empleadas para maximizar el potencial de estos modelos en aplicaciones prácticas. La implementación de estas técnicas de ajuste fino permite a los desarrolladores y científicos de datos adaptar modelos avanzados a necesidades específicas, mejorando la eficacia y relevancia de las soluciones basadas en inteligencia artificial en distintos sectores.

La tabla 2.2 muestra una evaluación de los tipos de ajuste fino aplicadas a modelos de lenguaje mencionadas anteriormente, destacando característi-

cas clave como la precisión, flexibilidad, velocidad, uso de recursos, y las principales aplicaciones para las cuales son más adecuadas.

Cada columna en la tabla 2.2 de los tipos de ajuste fino tiene un significado específico:

- **Precisión:** Indica el nivel de exactitud que cada tipo de ajuste ofrece en sus resultados.
- **Flexibilidad:** Muestra la capacidad del ajuste fino de adaptarse a diferentes tareas o requisitos.
- **Velocidad:** Representa el tiempo o la eficiencia con la que se implementa el ajuste fino.
- **Recursos:** Refleja los recursos computacionales necesarios para ejecutar el ajuste fino.
- **Aplicación principal:** Describe las áreas donde cada técnica de ajuste fino se desempeña mejor

A continuación se explica cada símbolo utilizado en la tabla 2.2:

- **Precisión:** El símbolo de estrella (★) indica el nivel de precisión de cada ajuste fino. Cinco estrellas (★★★★★) representan la máxima precisión.
- **Flexibilidad:**
 - La flecha hacia abajo (↓) indica baja flexibilidad.
 - La flecha hacia arriba (↑) representa alta flexibilidad.
 - La flecha de doble sentido (↔) indica flexibilidad media.
- **Velocidad:**
- El rayo (⚡) simboliza la velocidad del ajuste fino. Mas rayos (⚡⚡⚡⚡) indican una velocidad mas rápida.
- **Recursos:** El símbolo de tuerca (⚙) indica el uso de recursos computacionales. Más tuercas (⚙⚙⚙⚙) indican un mayor uso de recursos.

Finalmente, tras el análisis realizado a los tipos de ajustes finos. El mas adecuado para implementar en este trabajo fue el Ajuste fino general. Se destaca por su equilibrio entre precisión y flexibilidad, lo que lo convierte en una opción ideal para proyectos que requieren manejar múltiples tipos de datos para entrenar un modelo en específico. El ajuste fino general cuenta

Tabla 2.2: Tabla comparativa de diferentes tipos de ajuste fino

Tipo de ajuste fino	Precisión	Flexibilidad	Velocidad	Recursos	Aplicación principal
Específico para tareas	★★★★★	↓	↓↓	★★★★★	Seguridad, vulnerabilidades
General	★★★★★	↑	↓↓↓↓	★★★★	Generación de texto
Específico para dominios	★★★★★	↓	↓↓	★★★★★	Medicina, ingeniería
Eficiente en parámetros	★★★★	↑	↓↓↓↓↓	★★	Aplicaciones limitadas
Semi-supervisado	★★★★★	↔	↓↓↓	★★★★	Procesamiento de lenguaje
Adversarial	★★★★★	↓	↓↓↓	★★★★★	Detección de ataques
Enfocado en seguridad	★★★★★	↓	↓↓↓	★★★★★	Desarrollo seguro
Basado en instrucciones	★★★★★	↑	↓↓↓↓	★★★★	Asistentes virtuales
En dos etapas	★★★★★	↑	↓↓↓↓	★★★★	Clasificación de texto

con un buen rendimiento en términos de recursos computacionales. Aunque no es el más rápido de los métodos evaluados, su capacidad para gestionar un amplio rango de aplicaciones sin requerir ajustes adicionales lo hizo particularmente adecuado para el TFM, en el cual era necesario abordar una variedad de entradas textuales con diferentes niveles de complejidad.

2.3. Generación Automática de UML

La generación automática de diagramas de clases UML ha avanzado significativamente gracias a la integración de técnicas de aprendizaje automático y NLP. Un enfoque innovador basado en métricas de forma-topología, que dividen el proceso en tres fases principales: planarización, ortogonalización y compactación, permite asegurar que los diagramas sean estéticamente agradables y legibles al minimizar cruces y optimizar la disposición de los elementos [21]. Este método ha mejorado la disposición automática de diagramas, permitiendo manejar eficazmente tanto diagramas estructurados como aquellos con mínima estructura. Este método, implementado en herramientas como JarInspector, asegura una mejor estética y legibilidad, cruciales para la documentación y la ingeniería inversa del software. Además, se ha abordado el problema del marco en la verificación de diagramas UML mediante un enfoque automatizado que define propiedades invariantes, garantizando la integridad del sistema durante transiciones o cambios de estado. Esta integración de métodos formales y análisis estático es esencial para la fiabilidad y corrección de los diagramas de clases UML utilizados en el desarrollo de software.

El uso de redes neuronales convolucionales o CNNs (*Convolutional Neural Networks*), pueden ser modeladas y especificadas utilizando UML como parte de un flujo de diseño automatizado. Este enfoque [22] permite la representación gráfica de la arquitectura de la red, incluyendo sus capas y conexiones, facilitando la implementación en plataformas como CPUs, GPUs y la eficiencia en tareas de aprendizaje profundo. Asimismo, la extracción automática de diagramas de clases propios del dominio del análisis a partir de descripciones de casos de uso mediante técnicas de NLP facilita la transición de los requisitos a los modelos del dominio del diseño, asegurando una representación precisa y completa de las funcionalidades del sistema [23].

El sistema NEURAL-UML [24], diseñado para reconocer elementos estructurales en diagramas de clases UML mediante redes neuronales, mejora la automatización del análisis e interpretación de diagramas. Este sistema identifica clases, atributos, métodos y relaciones, aumentando la precisión y eficiencia en el reconocimiento de patrones estructurales complejos. La identificación precisa de relaciones de composición en diagramas UML es otro avance signi-

ficativo, contribuyendo a la fiabilidad y precisión de los diagramas utilizados en el diseño detallado de software y arquitectura [25].

La combinación de aprendizaje automático y técnicas basadas en patrones ha permitido desarrollar un enfoque totalmente automatizado para generar diagramas de clases UML a partir de especificaciones en lenguaje natural [26]. Este método, que incluye la creación de un conjunto de datos y el entrenamiento de un clasificador, reduce el esfuerzo manual y mejora la consistencia en la modelización del software. Además, un modelo de clasificación de diagramas UML basado en CNNs [27] demuestra el potencial del aprendizaje profundo para automatizar y mejorar el procesamiento de artefactos de ingeniería de software, facilitando la gestión y análisis de diagramas en proyectos a gran escala.

En el ámbito educativo, se ha explorado la conversión de ejercicios textuales de ingeniería de software en diagramas de clases UML, utilizando técnicas de NLP y generación de diagramas [28]. Este método automatiza la creación de materiales educativos y ejercicios, facilitando a los instructores la generación y validación de diagramas a partir de descripciones textuales. Por último, un *survey* de enfoques y técnicas para generar diagramas de clases UML a partir de requisitos en lenguaje natural [29] destaca los desafíos y avances en la automatización de este proceso. La revisión de métodos que van desde heurísticas hasta aprendizaje automático proporciona una visión integral del estado del arte y guía para futuros esfuerzos de investigación en este campo.

2.3.1. Conclusiones

En conclusión, los avances en la generación automática de diagramas de clases UML, impulsados por técnicas de aprendizaje automático, NLP y métodos formales, han mejorado significativamente la precisión, eficiencia y utilidad práctica de estos diagramas en el desarrollo y diseño de software. Estos desarrollos no solo reducen el esfuerzo manual sino que también aseguran una representación más coherente y precisa de los sistemas, facilitando tanto la ingeniería de requisitos como el diseño detallado de software.

La tabla 2.3 muestra una comparativa de las técnicas analizadas para la generación automática de diagramas UML. Se evaluaron diversas características clave, como el nivel de automatización, el uso de NLP, la aplicación de redes neuronales, la optimización visual de los diagramas y la facilidad de implementación de cada técnica. Esta comparación permite identificar qué métodos son más adecuados según las necesidades específicas de un proyecto, destacando aquellos que ofrecen un mayor grado de automatización y facilidad de uso. A continuación se detalla que significa cada columna de la tabla:

- **Automatización:** Indica si la técnica es completamente automática o si requiere intervención manual para completar el proceso.
- **NLP :** Muestra si la técnica utiliza NLP para extraer información de descripciones textuales y generar los diagramas UML automáticamente.
- **Redes N.:** Señala si la técnica emplea redes neuronales para el análisis o generación de diagramas UML.
- **Óptimo:** Indica si la técnica incluye alguna forma de optimización visual para mejorar la disposición de los elementos en el diagrama, minimizando cruces y mejorando la estética general.
- **Implementación:** Evalúa la complejidad o facilidad con la que se puede implementar la técnica en un entorno de desarrollo. Se mide en función de qué tan fácil es su integración en herramientas o proyectos.

Para comprender los símbolos usados en la tabla se describen cada uno de ellos y que significado conlleva encontrarlos en cada una de las columnas:

- **✓:** Indica que la técnica incluye esa característica (por ejemplo, es completamente automática o utiliza NLP).
- **✗:** Indica que la técnica no incluye esa característica (por ejemplo, no utiliza NLP o no tiene redes neuronales).
- **—:** Señala que la técnica requiere intervención manual o que el proceso no está completamente automatizado.
- **★:** Representa la facilidad de implementación, con más estrellas indicando una mayor facilidad de implementación. Se utiliza de una a cinco estrellas para reflejar la complejidad relativa.

La técnica que más se relaciona con este TFM es la **extracción de diagramas a partir de texto utilizando NLP**, ya que es automática y emplea el NLP para generar diagramas UML directamente desde descripciones textuales, como los casos de uso extendidos. Este enfoque coincide con el objetivo de este trabajo, que utiliza técnicas de NLP para analizar las descripciones del sistema y crear automáticamente diagramas de clases, facilitando así el desarrollo de sistemas IoT.

Tabla 2.3: Comparación de Técnicas para la Generación Automática de UML

Técnica	Automatización	NLP	Redes N.	Óptimo	Implementación
Métricas de forma-topología	✓	✗	✗	✓	★★★★
Extracción de diagramas a partir de texto (NLP)	✓	✓	✗	✗	★★★★★
Redes Neuronales Convolucionales (CNN)	—	✗	✓	✗	★★
NEURAL-UML	✓	✗	✓	✗	★★★★★
Clasificación mediante CNN	—	✗	✓	✗	★★
Enseñanza automatizada (NLP)	✓	✓	✗	✗	★★★★★

2.4. IA en la ingeniería de software

La integración de la IA en la ingeniería de software ha revolucionado múltiples aspectos del desarrollo y gestión del software. Desde la generación automática de código hasta la optimización de modelos y la validación de calidad, la IA ofrece soluciones avanzadas que mejoran la eficiencia, precisión y capacidad de respuesta en proyectos de software. Los artículos revisados en esta sección proporcionan un amplio panorama de cómo la IA está transformando la ingeniería de software en diversas áreas clave.

2.4.1. Generación automática de código y prototipos

Se ha explorado cómo ChatGPT puede acelerar el desarrollo de software mediante la generación automática de código y prototipos. En un caos de estudio, se asignaron tareas de programación a estudiantes de ingeniería de software [30], quienes utilizaron ChatGPT para resolver problemas previamente completados. Los resultados demostraron que ChatGPT puede reducir significativamente el tiempo de desarrollo, al proporcionar esqueletos de programas funcionales que requieren poca modificación, especialmente en lenguajes comunes, como Java. Sin embargo, los programadores experimentados que trabajaron con lenguajes menos comunes, como Dart, encontraron que el código generado por ChatGPT a menudo necesitaba ajustes significativos.

La utilidad de la IA generativa como herramienta de apoyo en el desarrollo de software se pone de manifiesto al aliviar la carga de trabajo de los desarrolladores y acelerar el proceso de codificación inicial. Además, se destaca la necesidad de equilibrar el uso de herramientas de IA para maximizar el aprendizaje y la comprensión de los estudiantes sin depender excesivamente de ellas. Los empleadores y estudiantes encuestados expresaron actitudes positivas hacia la integración de ChatGPT en el entorno educativo, siempre que se utilice como complemento y no como sustituto del aprendizaje profundo de los principios de programación [30].

2.4.2. Simulación y búsqueda con IA

Un estudio utiliza un modelo de simulación y un algoritmo de búsqueda AI para comparar los méritos relativos de los métodos de desarrollo de software ágiles y tradicionales [31]. Utilizando el modelo POM2, se llevaron a cabo numerosas simulaciones para identificar factores clave que influyen en el éxito de ambos enfoques. Los resultados indicaron que, en ningún caso, los métodos ágiles fueron inferiores a los tradicionales, y en muchos casos, los métodos ágiles superaron significativamente a los métodos planificados.

Este hallazgo sugiere que las organizaciones deberían adoptar metodologías ágiles como práctica estándar para maximizar la eficiencia y adaptabilidad en proyectos de software.

La combinación de simulación y algoritmos de búsqueda IA puede proporcionar evidencia empírica sólida para respaldar decisiones estratégicas en la gestión de proyectos de software. La capacidad de modelar y simular diferentes políticas de priorización de requisitos y sus efectos en el desempeño del proyecto permite a los gerentes de proyectos tomar decisiones más informadas y basadas en datos. Este enfoque también pone de relieve la importancia de utilizar herramientas automatizadas para explorar y optimizar el espacio de diseño del proceso de desarrollo de software, asegurando que las mejores prácticas sean identificadas y adoptadas de manera sistemática [31].

2.4.3. Optimización y portabilidad de modelos de IA/ML

Se abordan los desafíos y técnicas para portar modelos de IA y aprendizaje automático (*ML - Machine Learning*) a unidades de procesamiento de inteligencia (IPUs). La optimización de modelos IA/ML para hardware especializado, como las IPU, puede mejorar significativamente la eficiencia y velocidad de procesamiento. El trabajo presenta un análisis detallado de las adaptaciones necesarias para aprovechar al máximo las capacidades de las IPU, que son cruciales para aplicaciones que requieren procesamiento intensivo de datos y en tiempo real [32].

La importancia de ajustar y optimizar los modelos IA/ML para diferentes arquitecturas de hardware para maximizar el rendimiento y la eficiencia en diversas aplicaciones industriales y de investigación es esencial. Esto implica no solo la reimplementación de algoritmos en un nuevo entorno de hardware, sino también la reconsideración de los parámetros y configuraciones del modelo para alinearse con las capacidades específicas de las IPU. Este enfoque es crucial para mantener la competitividad en un campo en rápida evolución como el de la IA y el aprendizaje automático o ML [32].

2.4.4. Educación en ingeniería de software con IA generativa

Se presenta un estudio sobre la incorporación de ChatGPT en programas de aprendizaje de ingeniería de software. Se asignaron tareas a estudiantes de diferentes niveles de experiencia para utilizar ChatGPT en la resolución de problemas de programación. Los resultados mostraron que los programadores novatos encontraron muy útil la generación de código por parte de ChatGPT, ya que les proporcionaba soluciones rápidas y funcionales. Sin embargo, los programadores más experimentados, aunque reconocieron la utilidad de la herramienta, señalaron que el código generado a menudo re-

quería modificaciones para cumplir con estándares de calidad y prácticas comunes en programación [30].

Las encuestas realizadas a los estudiantes y sus empleadores revelaron una actitud positiva hacia la integración de IA generativa en el currículo educativo, siempre que se utilice como una herramienta de apoyo y no como un sustituto del aprendizaje profundo de los principios de programación. Este estudio destaca la necesidad de una integración equilibrada de herramientas de IA en la educación, asegurando que los estudiantes desarrollen una comprensión sólida de los conceptos fundamentales mientras se benefician de las ventajas que ofrece la tecnología AI [30].

2.4.5. Mapeo sistemático de técnicas de IA en ingeniería de software

Un mapeo sistemático de las técnicas de IA aplicadas en la ingeniería de software identifica áreas clave de impacto y desarrollo. Se examina cómo la IA ha influido en la gestión de proyectos, la automatización de pruebas y la mejora de la calidad del software. El estudio proporciona una visión general comprensiva del estado del arte, destacando las tendencias actuales y los desafíos futuros en la implementación de tecnologías de IA en el ciclo de vida del software [33].

El mapeo sistemático revela que la integración de la IA no solo mejora la eficiencia y precisión en el desarrollo de software, sino que también abre nuevas oportunidades para innovar y optimizar procesos tradicionales. Las técnicas de IA, como el aprendizaje automático y el procesamiento de lenguaje natural, están siendo cada vez más adoptadas para resolver problemas complejos en la ingeniería de software, proporcionando soluciones avanzadas y eficaces que transforman la manera en que se diseñan, desarrollan y mantienen los sistemas de software [33].

2.4.6. Pruebas de calidad para software con IA

Se exploran los desafíos y prácticas en la validación y pruebas de software que incorpora componentes de IA. Las dificultades en asegurar la calidad y confiabilidad de software impulsado por IA se deben a la naturaleza dinámica e impredecible de los modelos de IA. Se proponen metodologías y prácticas para mejorar las pruebas y la validación, asegurando que los sistemas de software con IA sean robustos y confiables antes de su despliegue [34].

Este trabajo es esencial para desarrollar marcos de prueba que puedan manejar la complejidad y variabilidad inherentes a los sistemas de IA. Se abordan estrategias para identificar y mitigar riesgos potenciales, asegurando que los modelos de IA funcionen correctamente bajo diversas condiciones operati-

vas. Este enfoque es crucial para garantizar que los sistemas de software con IA cumplan con los estándares de calidad necesarios para aplicaciones críticas, reduciendo el riesgo de fallos y aumentando la confianza en el uso de tecnologías de IA en entornos sensibles [34].

2.4.7. Análisis de estructura y distribución de fallos en redes de software

Un modelo de red de software se presenta para analizar la estructura del software y la distribución de fallos. Utilizando técnicas de modelado de redes, se proporcionan ideas o *insights* sobre cómo se distribuyen los fallos dentro de la estructura del software, identificando patrones y áreas críticas que requieren atención. Este enfoque permite mejorar la calidad y mantenibilidad del software al facilitar una gestión más eficaz de los recursos de desarrollo y mantenimiento [35].

La investigación subraya la importancia de entender la topología del software para identificar y mitigar proactivamente las áreas propensas a fallos. Al analizar la red de software, se pueden detectar componentes vulnerables y optimizar el diseño para aumentar la robustez y confiabilidad del software. Este enfoque es esencial para desarrollar sistemas de software más resilientes y reducir el riesgo de fallos, mejorando así la calidad general y la eficiencia del desarrollo de software [35].

2.4.8. Desarrollo de nuevos productos con IA

La IA está transformando el desarrollo de nuevos productos, destacando su uso para acelerar el ciclo de desarrollo, mejorar la toma de decisiones y personalizar productos según las preferencias del cliente. La IA permite a las empresas responder más rápidamente a las demandas del mercado, innovar con mayor eficiencia y reducir los costos asociados con el desarrollo de productos [36].

La capacidad de la IA para revolucionar el proceso de desarrollo de productos la convierte en un motor clave para la innovación y la competitividad en el mercado global. La implementación de técnicas de IA en el desarrollo de productos permite una mayor agilidad y adaptabilidad, lo que es crucial para mantener la relevancia en un entorno empresarial en rápida evolución. Este estudio subraya la importancia de integrar la IA en las estrategias de desarrollo de productos para aprovechar sus ventajas y mantenerse a la vanguardia en la industria [36].

2.5. Conclusiones

La tabla 2.4 las principales técnicas de IA en ingeniería de software previamente explicadas. Esta tabla tiene como objetivo comparar las técnicas en función de su nivel de automatización, su impacto en la calidad del software y su idoneidad para su uso en proyectos de gran escala.

Para comprender el concepto de cada columna se define en la siguiente lista, además del significado de cada símbolo en las columnas respectivas:

- **Automatización:** Evalúa si la técnica es completamente automática o si requiere intervención manual en alguna parte del proceso.
- **Impacto en Calidad:** Indica si la técnica mejora significativamente la calidad del software.
- **Uso en Proyectos Grandes:** Refleja la adecuación de la técnica para su aplicación en proyectos de gran escala, donde es necesario manejar un mayor volumen de datos.

Significado de los símbolos:

- ✓: Indica que la técnica incluye completamente esa característica (por ejemplo, automatización completa o impacto positivo en calidad).
- —: Señala que la técnica es parcialmente automática o tiene un impacto parcial, requiriendo intervención manual o siendo menos efectiva en proyectos de gran escala.
- ✗: Indica que la técnica no incluye esa característica o no es adecuada para esa evaluación.
- ★: Estrella que indica el grado de idoneidad para su uso en proyectos grandes. Más estrellas significan una mayor idoneidad para proyectos de mayor escala y complejidad (de 1 a 5 estrellas).

Aunque la técnica de **generación automática de código** se asocia principalmente con la creación de código fuente, su aplicación se extiende al ámbito de la generación automática de diagramas UML. En el contexto de este TFM, esta técnica resulta especialmente útil porque puede interpretar descripciones textuales de los requisitos del sistema y generar estructuras que luego pueden interpretarse en diagramas UML de manera. Esta capacidad para automatizar la creación de modelos de clases a partir de descripciones de alto nivel es crucial para sistemas IoT, donde se necesita una representación visual de los componentes y sus interacciones, facilitando así el desarrollo del sistema.

Tabla 2.4: Comparativa de Técnicas de IA en Ingeniería de Software

Técnica	Automatización	Impacto en Calidad	Uso en Proyectos Grandes
Generación automática de código	✓	✓	★★★
Simulación y búsqueda con IA	✓	✓	★★★★★
Optimización de modelos IA/ML	✓	✓	★★★★★
IA generativa en educación	✓	—	★★★
Mapeo sistemático de IA	—	✓	★★★★★
Pruebas de calidad con IA	—	✓	★★★★★
Análisis de estructura de fallos	✓	✓	★★★★★
Desarrollo de nuevos productos con IA	✓	✓	★★★★★

Capítulo 3

Desarrollo de la extensión de la herramienta

3.1. Análisis de requisitos

En esta sección se presenta un análisis detallado de los requisitos del proyecto cuyo objetivo es ayudar a los ingenieros de software a desarrollar sistemas IoT, automatizando parte de sus tareas mediante la extensión de la herramienta TDDT4IoTS. Se han identificado y categorizado los requisitos funcionales y no funcionales, así como las restricciones y los recursos utilizados. La solución propuesta utiliza modelos entrenados de OpenAI, específicamente el modelo base GPT-3.5-Turbo, para analizar las descripciones de los casos de uso y generar automáticamente estructuras JSON que representan diagramas de clases.

El propósito de esta sección es proporcionar una visión clara de los diferentes tipos de requisitos necesarios para el desarrollo del sistema, incluyendo los funcionales y no funcionales, así como las restricciones y los recursos utilizados. Esta comprensión detallada asegurará que el proyecto se desarrolle de manera efectiva, cumpliendo con las expectativas de los usuarios y los objetivos del proyecto.

3.1.1. Requisitos funcionales

Los requisitos funcionales se utiliza para para definir las capacidades y características específicas que debe tener el sistema para cumplir con los objetivos del proyecto. Un aspecto clave es la identificación automática de elementos clave a partir de descripciones textuales proporcionadas por los desarrolladores. Esto incluye la capacidad de detectar clases, atributos y relaciones dentro del sistema descrito, facilitando así la generación inicial de los dia-

gramas de clases conceptuales.

Otra funcionalidad es la capacidad de generar el gráfico correspondiente al diagrama de clases a partir de la estructura JSON generada a partir del análisis de las descripciones de los casos de uso. Este diagrama será el esquema inicial del proyecto, proporcionando una base que los ingenieros pueden refinar posteriormente para obtener diagramas de clases del dominio de la solución. Además, la herramienta extendida debe integrarse sin problemas con TDDT4IoT, permitiendo a los usuarios (ingenieros en software) aprovechar las nuevas funcionalidades para facilitar su labor y aumentar su productividad.

3.1.2. Requisitos No Funcionales

Los requisitos no funcionales se refieren a los criterios que deben cumplir los sistemas en términos de:

- **Rendimiento y eficiencia:** El sistema debe ser capaz de procesar descripciones textuales y generar diagramas de clases de manera rápida y eficiente, para minimizar el tiempo de espera de los usuarios. La integración de los modelos de IA debe estar optimizada para asegurar tiempos de respuesta aceptables, incluso bajo cargas de trabajo altas.
- **Usabilidad:** La interfaz del sistema debe ser intuitiva y fácil de usar, permitiendo que los ingenieros de software aprovechen las nuevas funcionalidades sin una curva de aprendizaje pronunciada. Esto incluye una interfaz gráfica clara, guías de usuario y documentación accesible.
- **Fiabilidad:** El sistema debe ser estable y confiable, asegurando un tiempo de inactividad mínimo y una alta disponibilidad para los usuarios. Debe ser capaz de manejar errores de manera robusta y recuperar su estado normal sin pérdida de datos.
- **Mantenibilidad:** El sistema debe estar diseñado de manera que sea fácil de mantener y actualizar. Esto incluye una arquitectura modular que permita realizar mejoras y correcciones de errores sin afectar el funcionamiento general del sistema. La documentación del código debe ser clara y completa para facilitar su mantenimiento.
- **Escalabilidad:** La arquitectura del sistema debe permitir la escalabilidad para manejar un creciente número de usuarios y la cantidad de datos procesados sin degradar el rendimiento del sistema. Esto incluye la capacidad de distribuir la carga de trabajo y aumentar la capacidad del sistema según sea necesario.

3.1.3. Restricciones

Las restricciones del proyecto son los factores que limitan las opciones de diseño y desarrollo, imponiendo ciertos límites que deben ser respetados.

- **Dependencia de servicios de IA:** La solución depende del uso de la API de servicios de OpenAI, lo cual implica un costo asociado y posibles limitaciones en la disponibilidad del servicio. Es crucial gestionar adecuadamente los costos y tener planes de contingencia para posibles interrupciones del servicio.
- **Licencias de software *open source*:** Todo el software *open source* utilizado debe cumplir con sus respectivas licencias, asegurando que no se infrinjan derechos de propiedad intelectual. Es importante revisar y entender las condiciones de cada licencia para evitar problemas legales.
- **Integración con sistemas existentes:** La herramienta implementada debe integrarse sin problemas con la herramienta TDDT4IoTS existente y otros sistemas de apoyo. Esto puede requerir adaptaciones específicas para asegurar la compatibilidad y un funcionamiento fluido entre los diferentes componentes del sistema resultante.

3.1.4. Recursos Utilizados

Los recursos utilizados para el desarrollo del proyecto incluyen tanto herramientas de software como servicios externos, para lograr los objetivos propuestos.

- **Modelos de IA:** Utilización de modelos de IA para procesar texto y extraer información relevante. Los modelos pueden ser seleccionados y configurados según las necesidades del desarrollador, optimizando así el proceso de diseño de sistemas.
- **Framework Spring:** Uso de Spring para el desarrollo de microservicios que gestionan las funcionalidades del sistema y la comunicación con la base de datos PostgreSQL. Spring Boot proporciona una plataforma robusta y escalable para construir aplicaciones empresariales.
- **PostgreSQL:** Base de datos relacional utilizada para almacenar la información relevante del sistema, como descripciones textuales, estructuras JSON generadas y diagramas de clases. PostgreSQL es conocido por su fiabilidad y rendimiento, siendo adecuado para aplicaciones de misión crítica.

- **gRPC:** Protocolo de comunicación utilizado para la interacción eficiente entre los microservicios de Spring y el componente de Python que maneja la integración con la API de servicios de inteligencia artificial. gRPC permite una comunicación rápida y eficiente para aplicaciones distribuidas.
- **HTTPS:** Protocolo utilizado para la comunicación entre el servicio de gestión de usuarios y el servidor existente de la herramienta, validando las peticiones mediante tokens de sesión. HTTPS es ampliamente utilizado y proporciona un medio estándar para la comunicación en la web.

3.2. Arquitectura

La arquitectura del sistema desarrollado para la extensión de la herramienta TDDT4IoTS está diseñada para ser modular, escalable y fácil de mantener. Esta arquitectura se compone de varias capas, cada una con responsabilidades específicas, lo que permite una clara separación de preocupaciones y facilita tanto el desarrollo como la integración de nuevas funcionalidades. La estructura en capas asegura que cada componente del sistema pueda evolucionar independientemente, mejorando la robustez y la flexibilidad del mismo.

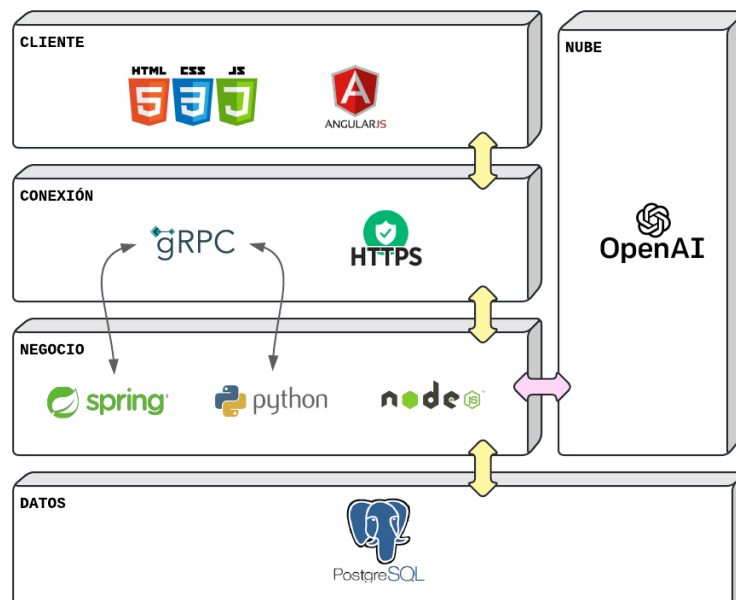


Figura 3.1: Arquitectura de la extensión.

La figura 3.1 presenta las tecnologías utilizadas en las diferentes capas que componen la arquitectura del sistema: la capa del cliente, la capa de conexión, la capa de negocio, la capa de la nube y la capa de datos, describiéndose cada una de ellas en los siguientes subapartados que se incluyen a continuación. Cada una de estas capas juega un papel importante en el funcionamiento del sistema, asegurando que las interacciones entre los usuarios y las funcionalidades avanzadas basadas en inteligencia artificial se realicen de manera eficiente y segura. Esta organización en capas también facilita la gestión de la lógica del negocio, el almacenamiento de datos y la comunicación entre los diferentes servicios que forman parte de la solución.

3.2.1. Capa del cliente

La capa del cliente es la interfaz donde los usuarios interactúan directamente con la herramienta y la extensión implementada. Esta capa se encarga de proporcionar una experiencia de usuario intuitiva y accesible mediante interfaces web. Los usuarios pueden acceder a las funcionalidades de la herramienta, como la creación y análisis de casos de uso, y visualizar los diagramas de clases generados. Esta capa incluye todas las vistas y componentes de la interfaz de usuario que permiten a los ingenieros de software realizar sus tareas de manera eficiente.

- **Interfaces web:** Desarrolladas utilizando tecnologías como HTML, CSS y JavaScript, junto con frameworks modernos como React o Angular, estas interfaces permiten a los usuarios interactuar con las funcionalidades de la herramienta.
- **Interacción usuario-sistema:** Los usuarios pueden enviar descripciones de casos de uso, recibir los resultados del análisis efectuado y gestionar sus proyectos directamente desde el navegador web.

3.2.2. Capa de conexión

La capa de conexión se encarga de manejar las comunicaciones entre los diferentes componentes del sistema. Esta capa es crucial para garantizar que los datos fluyan de manera eficiente y segura entre las diferentes partes del sistema.

- **Conexión HTTPS:** Utilizada para la comunicación entre el cliente y la capa de negocio. Las peticiones HTTPS son manejadas por el *framework* Spring, que gestiona la validación de las solicitudes y la entrega de respuestas adecuadas.

- **Conexión gRPC:** Establece la conexión entre los microservicios implementados con Spring y Python que se encuentran en la capa de negocio, y que manejan la integración con los modelos de IA. gRPC proporciona una comunicación rápida y eficiente, esencial para el procesamiento intensivo de datos requerido por los modelos de IA.

3.2.3. Capa de negocio

La capa de negocio alberga todos los microservicios implementados que contienen la lógica del negocio necesaria para manejar el uso de los modelos de inteligencia artificial y el entrenamiento de los mismos. Esta capa es el núcleo funcional de la aplicación, donde se llevan a cabo las principales operaciones.

- **Microservicios:** Para el control de todas las peticiones que son enviadas por el cliente se desarrollaron los microservicios con Spring Boot. Python fue implementado de forma interna para controlar las peticiones que se realizarían con la API de OpenAI, comunicándose igualmente mediante gRPC con los microservicios de Spring Boot. Finalmente Node.js fue implementado para tener a disponibilidad como un servicio web, el interprete del lenguaje de símbolos que cuenta la herramienta TDDT4IoTS, también mantiene una comunicación interna con los microservicios de Spring Boot.
- **Lógica de negocio:** Cada microservicio está diseñado para realizar tareas específicas, como la validación de peticiones, el análisis de texto utilizando modelos de IA, y la generación y refinamiento de diagramas de clases conceptuales.

3.2.4. Capa de la nube

La capa de la nube se refiere al uso de servicios externos, en este caso, los modelos de OpenAI. Esta capa proporciona las capacidades de inteligencia artificial necesarias para analizar las descripciones de los casos de uso y generar estructuras JSON que representen diagramas de clases.

- **Modelos de IA:** Se utilizó el modelo base *GPT-3.5* de OpenAI para procesar texto y extraer información relevante. Los modelos pueden ser seleccionados y configurados según las necesidades del desarrollador.
- **API de OpenAI:** La integración con la API de OpenAI permite el acceso a potentes modelos de IA que facilitan el análisis y procesamiento de las descripciones textuales, optimizando el proceso de diseño de sistemas.

3.2.5. Capa de datos

La capa de datos es responsable de almacenar toda la información relevante generada y utilizada por el sistema. Esto incluye los modelos y entrenamientos realizados, así como los diagramas de clases generados a partir de los análisis efectuados.

- **Base de datos PostgreSQL:** Utilizada para almacenar descripciones textuales, estructuras JSON generadas, diagramas de clases y cualquier otro dato relevante. PostgreSQL es una base de datos relacional robusta y escalable, adecuada para aplicaciones de misión crítica.
- **Almacenamiento de modelos:** Los datos relacionados con los modelos de IA, incluyendo los resultados del entrenamiento y las configuraciones utilizadas, son almacenados de manera segura en tablas relacionales dentro de la base de datos de la herramienta TDDT4IoTS para asegurar su disponibilidad y fácil acceso cuando sea necesario.

3.3. Desarrollo e Implementación

El desarrollo e implementación de la extensión de la herramienta TDDT4IoTS se llevó a cabo a través de una serie de fases bien definidas, cada una de las cuales fue importante para cumplir con el objetivo principal del proyecto. Este enfoque estructurado permitió un desarrollo iterativo, asegurando que cada componente del sistema fuera desarrollado y probado de manera correcta antes de proceder a la siguiente iteración. El proceso de desarrollo consiste en los siguientes pasos, realizados de manera reiterativa hasta alcanzar los objetivos planteados: análisis detallado de los requisitos, seguido por el diseño de la extensión, la configuración de los frameworks necesarios y finalmente, el desarrollo de los componentes en un sistema funcional. En la figura 3.2 se presenta gráficamente el enfoque metodológico seguido en cada iteración del proceso completo de desarrollo.

En la primera fase, se realizó un análisis de los requisitos para definir las funcionalidades y características necesarias. En la segunda fase, se diseñaron los componentes de la extensión basándose en los requisitos definidos. La tercera fase implicó la configuración de los frameworks y herramientas que soportarían el desarrollo, mientras que en la cuarta y última fase se desarrollaron los componentes individuales que conllevarían a la integración de todos los componentes desarrollados, asegurando que funcionaran conjuntamente.

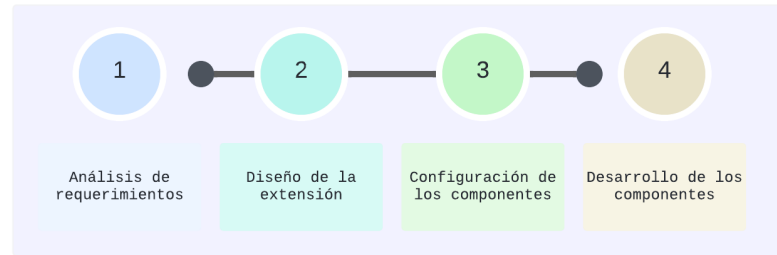


Figura 3.2: Fases de Desarrollo.

3.3.1. Fase 1: Análisis de requerimientos

En esta fase, se han definido los requisitos funcionales y no funcionales que guiarán la extensión de la herramienta TDDT4IoTS, asegurando que cumpla con las necesidades específicas de los ingenieros de software que utilizan el sistema para analizar descripciones de casos de uso en entornos IoT.

Identificación de usuarios y casos de uso

La extensión puede funcionar exitosamente con la manipulación de dos tipos de perfiles de usuario, el desarrollador (ingenieros de software) y un administrador. A continuación se describen cada uno de los perfiles y como utilizan la extensión mediante la herramienta TDDT4IoTS:

- **Desarrollador:** Es el usuario encargado de ingresar los datos de entrada mediante las descripciones del sistema IoT a desarrollar en forma de casos de uso extendidos. Mediante la herramienta podrá transformar estas descripciones en un diagramas de clases y poder validar la arquitectura previa del sistema IoT. Además, será el encargado de seleccionar el modelo de OpenAI que crea conveniente a utilizar.
- **Administrador:** Tendrá la posibilidad de registrar los modelos base de OpenAI y tener el privilegio de entrenar modelos, claramente deberá tener conocimiento sobre los modelos base que permitan realizar un ajuste fino, debido a que la herramienta necesita estos modelos para poder realizar los debidos entrenamientos y se puedan utilizar los modelos entrenados en los desarrollos de los sistemas IoT ingresados en la herramienta.

Los casos de uso que se describen a continuación son las interacciones directas entre los perfiles de usuarios y la extensión. Cada caso de uso corresponde a una funcionalidad clave que la extensión debe proporcionar a los usuarios.

- **Crear/Editar casos de uso:** El desarrollador puede crear o editar descripciones detalladas de casos de uso para sistemas IoT, introduciendo los actores y componentes involucrados. El sistema valida y almacena esta información.
- **Generar automáticamente el diagrama de clases:** A partir de las descripciones ingresadas, la extensión genera automáticamente un diagrama de clases conceptual utilizando el modelo base *GPT-3.5* de OpenAI, que luego es almacenado y presentado gráficamente.
- **Entrenar modelos:** El usuario administrador tendrá el privilegio de poder entrenar los modelos que se crean necesarios y aporten un mejor funcionamiento a la herramienta mediante la extensión.
- **Editar diagrama de clases generado:** Una vez generado el diagrama de clases completamente, el desarrollador podrá modificar el diagrama según las necesidades que crea conveniente. Los cambios realizados se guardaran dentro de la herramienta para futuras revisiones.

La figura 3.3 muestra el diagrama de casos de uso identificado para comprender mejor el funcionamiento de la extensión sobre la herramienta. Permite identificar los actores principales y los casos de uso previamente mencionados.

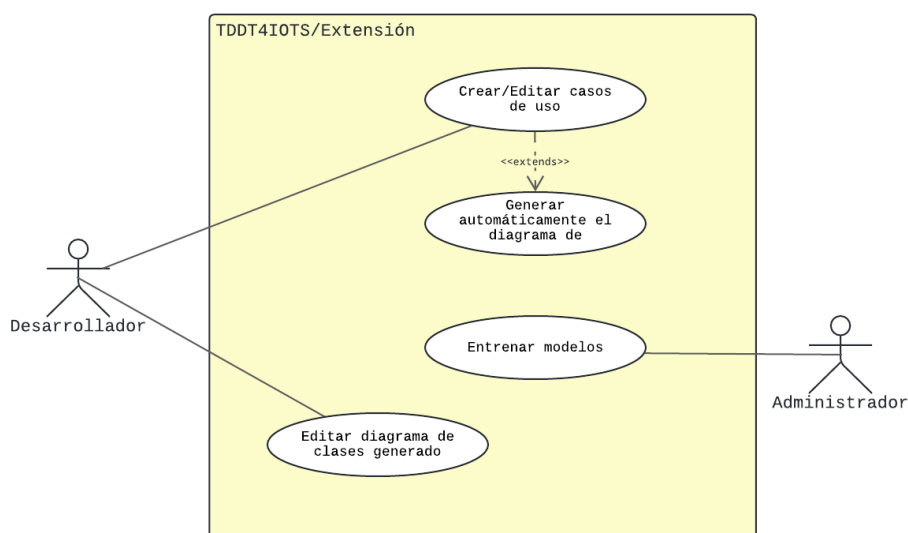


Figura 3.3: Diagrama de casos de uso de la extensión.

3.3.2. Fase 2: Diseño de la extensión

En esta fase se explicará detalladamente el comportamiento de cada proceso necesario para que la extensión de la herramienta funcione correctamente. Se dedicó tiempo a analizar cuál sería la mejor forma de adaptar la nueva funcionalidad a la herramienta sin afectar los componentes existentes.

Diseño de la arquitectura

En esta subsección se explicará a detalle que tecnologías fueron necesarias implementar en cada capa y para que se utilizaron dichas tecnologías para lograr con el objetivo general y específicos del proyecto.

Capa del cliente

- **AngularJS:** Para mantener el correcto funcionamiento de la herramienta TDDT4IoTS, se decidió continuar utilizando este framework para generar las nuevas interfaces web. Esto permite que los usuarios perciban y usen la herramienta de la misma manera que lo hacían anteriormente.

Capa de conexión

- **gRPC:** Este protocolo de comunicación facilita la integración interna de componentes en el servidor. Su objetivo es aprovechar las ventajas de diversas tecnologías y permitir que trabajen conjuntamente, asignando cargas de trabajo que pueden ser mejor ejecutadas por otras tecnologías.
- **HTTPS:** Es el protocolo estándar para el desarrollo de aplicaciones web. Permite enviar y recibir datos entre un navegador web y el servidor, lo que es esencial para cumplir con los objetivos de cualquier proyecto implementado. En la extensión de la herramienta, permite realizar las peticiones necesarias al servidor y obtener respuestas que se mostrarán a los usuarios finales.

Capa de negocio

- **Spring Framework:** Para procesar todas las peticiones de la extensión, se utilizó el framework Spring, que emplea Java como lenguaje de programación. Java es conocido por su robustez y su capacidad para manejar un alto volumen de solicitudes. Considerando la cantidad de peticiones que se recibirán cuando se utilice la extensión mediante la

herramienta, se busca mantener una conexión estable y aprovechar todas las ventajas que ofrece Spring Framework. Este framework es ideal para este proceso debido a su capacidad para manejar concurrencia, su soporte para transacciones, y su eficiente gestión de recursos. Además, Spring Framework permite ofrecer respuestas rápidas a las peticiones, gracias a su alta capacidad de procesamiento de datos y a su arquitectura modular, que facilita el mantenimiento y la escalabilidad de la aplicación.

- **NodeJs:** La herramienta cuenta con un intérprete que analiza la redacción de los casos de uso creados mediante la misma. Este intérprete, denominado ArmadilloJs, fue desarrollado como una librería JavaScript. Para generar los archivos necesarios para el entrenamiento de los modelos de OpenAI, surgió la necesidad de migrar la librería a un servicio web. Dado que la librería fue programada en JavaScript, se decidió implementar el servicio web con Node.js. Esta elección permite utilizar el intérprete de forma interna a través de los microservicios proporcionados por Spring Framework, aprovechando la compatibilidad y eficiencia de Node.js para manejar solicitudes asíncronas y el robusto manejo de transacciones y recursos de Spring Framework.
- **Python:** Como parte de la extensión, se hace uso de la API de OpenAI. Python se destaca como uno de los lenguajes más utilizados en tecnologías de inteligencia artificial debido a su simplicidad, versatilidad y una amplia gama de bibliotecas especializadas. Aprovechando la existencia de una biblioteca de OpenAI para Python, se implementó la comunicación con OpenAI para mantener una interacción más fluida y bien desarrollada. Además, Python ofrece ventajas significativas en el desarrollo de aplicaciones de IA, como una sintaxis clara y legible, una gran comunidad de desarrolladores y soporte extensivo para frameworks y herramientas de aprendizaje automático y procesamiento de datos. Estas características hacen de Python la elección ideal para integrar OpenAI y aprovechar al máximo sus capacidades de IA en la herramienta.

Capa de la nube

- **OpenAI:** La comunicación con la API de OpenAI permite utilizar los recursos almacenados en sus servidores. Esto da una ventaja de no utilizar recursos locales para implementar modelos de inteligencia artificial desde cero. También aprovechando los recursos de OpenAI se almacenan los datos entrenados por cada cuenta de usuario que este registrada en sus servicios. Cabe recalcar que también se almacena ciertos datos de forma local.

Capa de datos

- **PostgreSQL:** La comunicación con la API de OpenAI permite utilizar los recursos alojados en sus servidores, lo cual ofrece la ventaja de no necesitar recursos locales para implementar modelos de inteligencia artificial desde cero. Al aprovechar los recursos de OpenAI, los datos entrenados se almacenan por cuenta de usuario registrada en sus servicios. Además, es importante destacar que algunos datos también se almacenan localmente para mejorar la eficiencia y disponibilidad. Esta combinación de almacenamiento en la nube y local asegura un rendimiento óptimo y una gestión eficiente de los datos.

Diseño de los componentes creados

En la figura 3.4 se puede visualizar la comunicación entre todos los componentes que controlan el correcto funcionamiento de la extensión. El funcionamiento de los componentes es del a siguiente manera:

1. **db-repository-tddt4iots:** Contiene todos los objetos necesarios para acceder a los datos de forma óptima.
2. **ms-core-tddt4iots:** Mantiene separados los recursos reutilizables por toda la extensión, además de contener los DTO necesarios para las peticiones entre el cliente y el servidor.
3. **ms-core-tddt4iots-openai:** Es el componente principal del proyecto, consumiendo los recursos de los componentes mencionados anteriormente, utilizados como dependencias mediante Maven.
4. **ms_core_tddt4iots_py:** Establece la comunicación con OpenAI y configura lo necesario para realizar entrenamientos e implementar los modelos.
5. **armadillo-api:** API RestFul que interpreta casos de uso redactados en lenguaje de símbolos para obtener texto en lenguaje natural y crear archivos de entrenamiento.

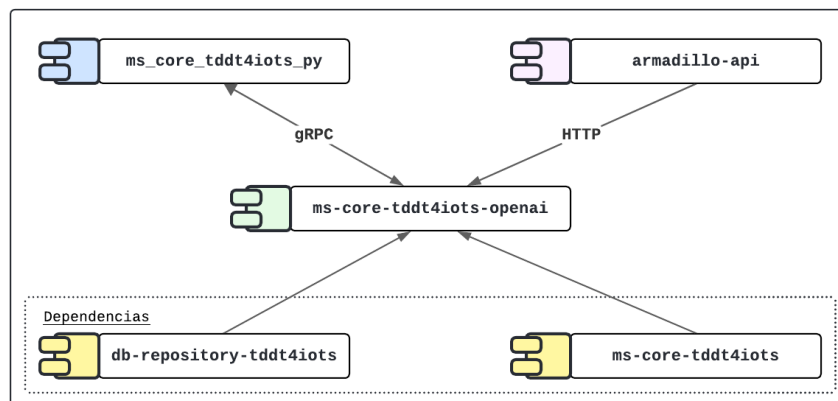


Figura 3.4: Componentes de la extensión.

Modelos de IA

OpenAI ofrece una variedad de modelos que pueden ser utilizados mediante el ajuste fino. Esta técnica permite entrenar modelos para especializarlos en tareas específicas, adaptándolos a las necesidades particulares del proyecto en el que se esté trabajando. Se llegó a la conclusión de registrar los posibles modelos base dentro de la herramienta, lo que permite seleccionar el modelo base adecuado para iniciar un nuevo entrenamiento y ponerlo a disposición de todos los usuarios.

Además, el proceso de ajuste fino no solo optimiza la precisión del modelo, sino que también mejora su eficiencia en la tarea específica, reduciendo el tiempo y los recursos necesarios para obtener resultados precisos. De esta manera, OpenAI asegura que sus usuarios puedan aprovechar al máximo las capacidades avanzadas de sus modelos de inteligencia artificial, facilitando la implementación de soluciones innovadoras y efectivas en diversos ámbitos.

Entrenamiento de modelos

El ajuste fino permitió entrenar modelos existentes para cumplir con los requisitos funcionales del proyecto. Cada entrenamiento quedará registrado dentro de la herramienta, permitiendo llevar un historial detallado de todos los entrenamientos realizados a los modelos seleccionados. Esta tarea es llevada a cabo por los usuarios interesados en entrenar modelos. Para entrenar modelos, se debe seguir una serie de pasos que se han tratado de automatizar por completo. Es necesario contar con una secret key de OpenAI para poder entrenar estos modelos. Además, se necesita un conjunto de datos que serán enviados al modelo mediante el ajuste fino y esperar los resultados del entrenamiento.

El archivo debe contener las posibles peticiones y respuestas que el modelo debe procesar cuando sea utilizado por otros usuarios mediante la herramienta. Este archivo actúa como una guía de comportamiento, asegurando que el modelo responda adecuadamente a diferentes escenarios. Además, es crucial que los datos estén bien estructurados y sean representativos de los casos de uso reales para garantizar la eficacia del entrenamiento. De esta manera, se optimiza el rendimiento del modelo y se mejora la precisión de sus respuestas, facilitando su integración y uso en aplicaciones prácticas.

Uso de modelos

Teniendo algunos modelos entrenados, se puede hacer uso de ellos para interpretar las descripciones de los casos de uso redactados en lenguaje natural. El usuario interesado en el entrenamiento realizado por otro usuario podrá seleccionarlo desde el historial de entrenamiento. Para llevar a cabo este proceso, también se debe contar con una secret key para poder acceder al modelo entrenado. Debido a que OpenAI solo permite acceder al modelo entrenado con la misma secret key con la que fue entrenado, se implementó la funcionalidad de compartir la secret key del usuario que entrenó el modelo para facilitar el acceso compartido.

3.3.3. Fase 3: Configuración de los componentes

En esta fase se detallan los componentes que fueron configurados para abarcar la funcionalidad completa de la extensión. Cada componente está diseñado para cumplir con una funcionalidad específica, asegurando que el sistema no solo cumpla con los requisitos actuales, sino que también sea escalable y mantenible a lo largo del tiempo. Se han tomado en cuenta prácticas de diseño modular y principios de ingeniería de software que permiten una fácil actualización y expansión de la herramienta, facilitando así su adaptación a futuras necesidades y mejoras tecnológicas.

- **db-repository-tddt4iots:** Para mantener la conexión a la base de datos del proyecto, se configuró este componente como una dependencia del proyecto principal. El objetivo de este componente es centralizar todas las operaciones relacionadas con el acceso a datos, incluyendo el mapeo de entidades de la base de datos a clases Java, la ejecución de consultas SQL, y otras tareas de manejo de datos. Esta configuración permite una gestión más eficiente y organizada de las interacciones con la base de datos, asegurando un acceso a datos consistente y fiable.
- **ms-core-tddt4iots:** Este componente se encargará de contener todos los DTO necesarios para realizar las peticiones y respuestas, tanto del cliente al servidor como del servidor a la base de datos. Además,

incluirá métodos utilitarios que podrán ser utilizados por otros componentes o proyectos generales, facilitando la reutilización de código y mejorando la eficiencia del desarrollo.

- **ms-core-tddt4iots-openai** Este componente constituye el proyecto principal y utiliza como dependencias los demás proyectos mencionados anteriormente. En este proyecto se configurarán todos los microservicios que se liberarán y que podrán ser consumidos por la extensión, asegurando así una integración fluida y coherente. Además, este proyecto se encarga de configurar los parámetros necesarios para acceder a la base de datos, manteniendo los objetos de datos aislados en sus respectivas dependencias. Esta separación de responsabilidades no solo mejora la organización del código, sino que también facilita el mantenimiento y la escalabilidad del sistema. Al centralizar la configuración de los microservicios y las conexiones a la base de datos, este componente principal actúa como el núcleo del sistema, permitiendo una gestión eficiente de las interacciones entre los diferentes módulos y garantizando que todas las partes del proyecto funcionen de manera integrada y eficiente.
- **ms_core_tddt4iots_py**: Para implementar el uso del lenguaje de programación Python, se configuró este proyecto como un servidor gRPC. Este componente facilita la comunicación eficiente y rápida con la API de OpenAI mediante las librerías correspondientes, permitiendo tanto el entrenamiento como el uso de los modelos existentes. Dentro de este proyecto, se gestionan todas las interacciones con OpenAI, asegurando que las solicitudes y respuestas sean manejadas de manera óptima. La configuración del servidor gRPC permite una comunicación robusta y de baja latencia entre los diferentes servicios y el backend de OpenAI, lo que es crucial para operaciones que requieren procesamiento en tiempo real. Además, este proyecto no solo se limita a gestionar el entrenamiento de los modelos, sino que también supervisa su implementación y actualización continua. Al utilizar gRPC, se garantiza que las conexiones sean seguras y eficientes, facilitando el desarrollo y la integración de nuevas funcionalidades basadas en IA.
- **armadillo-api**: La herramienta originalmente incluía un intérprete de descripciones de casos de uso redactadas en un lenguaje de símbolos. Este componente se configuró como un servicio web, permitiendo su utilización a través de una API REST que integra la biblioteca de JavaScript correspondiente. Esta configuración facilita el acceso y la interpretación de los casos de uso, asegurando que las descripciones sean procesadas de manera eficiente y precisa. Además, la implementación como servicio web permite una mayor flexibilidad y escalabilidad, permitiendo que otros sistemas y aplicaciones puedan interactuar con

el intérprete de manera sencilla y consistente.

3.3.4. Fase 4: Desarrollo de los componentes

En esta fase se detalla todo lo desarrollado dentro de cada componente. Durante el desarrollo de esta fase, se logró aclarar y abordar todos los requisitos funcionales mencionados en la documentación del proyecto. A continuación, se describirán los paquetes implementados en cada componente y los pasos que se definieron para llevar a cabo la interpretación de los casos de uso redactados en lenguaje natural para asegurar el funcionamiento completo de la extensión en la herramienta y satisfacer las necesidades del usuario final.

- **db-repository-tddt4iots:** Dentro de este componente se empezó con el desarrollo de todos los paquetes necesarios para estructurar todos los objetos que intervienen en el manejo de los datos.
 - *config*: configuración de los repositorios y transacciones JPA para la extensión de la herramienta.
 - *entity*: entidades que mapean las tablas de la base de datos.
 - *repository*: contiene todas las sentencias SQL que permitirán obtener los datos para las entidades.
 - *impl*: implementaciones de las interfaces del paquete *service* para liberar los métodos existentes al proyecto que esté utilizando este componente como dependencia (ver figura 3.5).

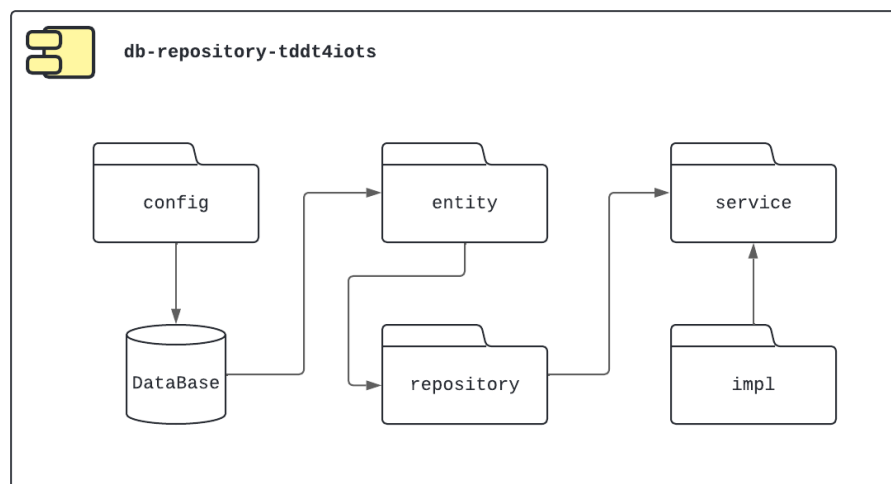


Figura 3.5: Paquetes desarrollados dentro del componente db-repository-tddt4iots.

- **ms-core-tddt4iots:** Se estructura en varios paquetes, cada uno con objetivos específicos:
 - *config*: configuración necesaria para la comunicación gRPC y puede ser escalable para desarrollar otras clases que permitan la conexión con otros tipos de protocolo
 - *cons*: almacena constantes utilizadas en todo el proyecto
 - *util*: contiene utilidades y funciones auxiliares reutilizables
 - *proto*: incluye los archivos `.proto` que definen los métodos y mensajes gRPC para la interacción entre componentes
 - *dto*: alberga los *Data Transfer Objects* (DTO).
 - *grpc_service*: contiene las implementaciones de los servicios gRPC definidos en los archivos `.proto`, asegurando una integración eficiente y consistente en todo el proyecto (ver figura 3.6).

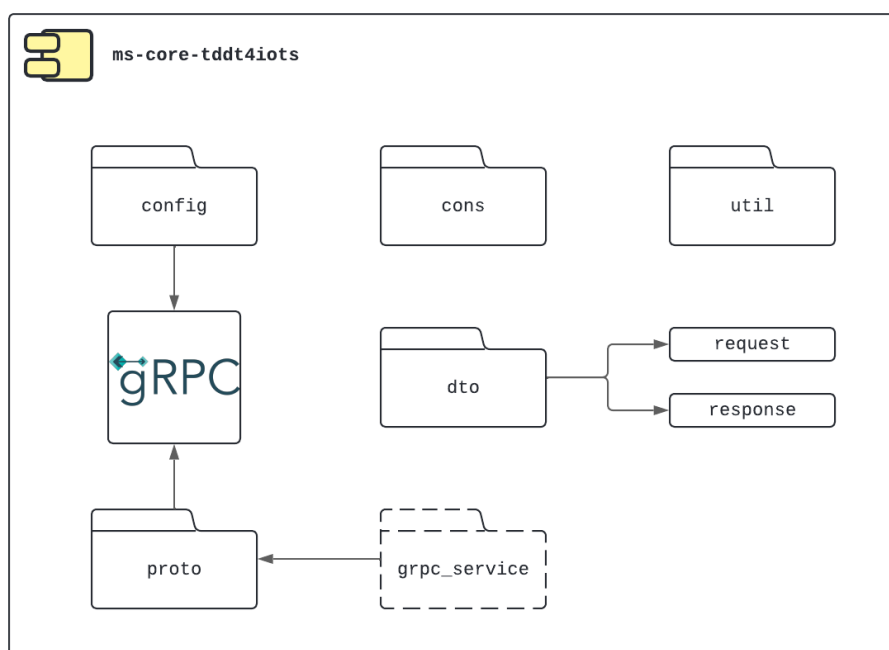


Figura 3.6: Paquetes desarrollados dentro del componente `ms-core-tddt4iots`.

- **ms-core-tddt4iots-openai:** La figura 3.7 muestra la estructura y como se comunican los paquetes dentro de este componente. A continuación se detalle el funcionamiento de cada paquete:

- *config*: gestiona la configuración necesaria para el funcionamiento del proyecto, configura el uso de las propiedades declaradas en el archivo `application.properties` y también incluye la configuración del cliente gRPC.
- *rest*: se encarga de las interfaces **REST**, permitiendo la interacción del proyecto a través de llamadas HTTP.
- *communication*: alberga la lógica de comunicación tanto gRPC como **REST**, facilitando la interacción fluida entre los componentes internos y externos.
- *bo*: define las clases de negocio y la lógica empresarial, encapsulando los procesos del sistema.
- *impl*: contiene las implementaciones concretas de estas interfaces y servicios.
- *grpc*: maneja las operaciones relacionadas con gRPC, asegurando una comunicación eficiente con el servidor de Python que utiliza la librería de OpenAI.
- *mapper*: incluye los mapeadores necesarios para convertir entidades entre diferentes capas del sistema.
- *datasource*: gestiona las fuentes de datos, configurando las conexiones a la base de datos y otros servicios de almacenamiento.

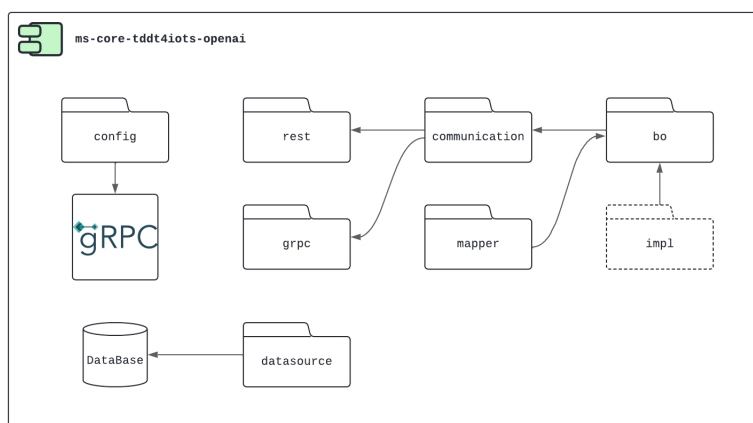


Figura 3.7: Paquetes desarrollados dentro del componente `ms-core-tddt4iots-openai`.

Dentro de este componente se detallan los tres microservicios que realizan las tareas principales del proyecto:

1. Creación de los archivos en formato JSONL que almacenarán los **prompt**, especificando los posibles mensajes de entrada y salida que el modelo debería devolver.
2. Entrenamiento de el modelo, especificando los pasos que la extensión realizará de forma interna para llevar a cabo el entrenamiento de los modelos específicos almacenados localmente.
3. Uso de un modelo entrenado, detallando cómo se recibirá la solicitud del cliente y cómo el modelo devolverá la respuesta esperada.

En la figura 3.9 se detalla el proceso analizado para la creación de los archivos JSONL y su almacenamiento, los cuales serán utilizados posteriormente en el entrenamiento de los modelos de OpenAI.

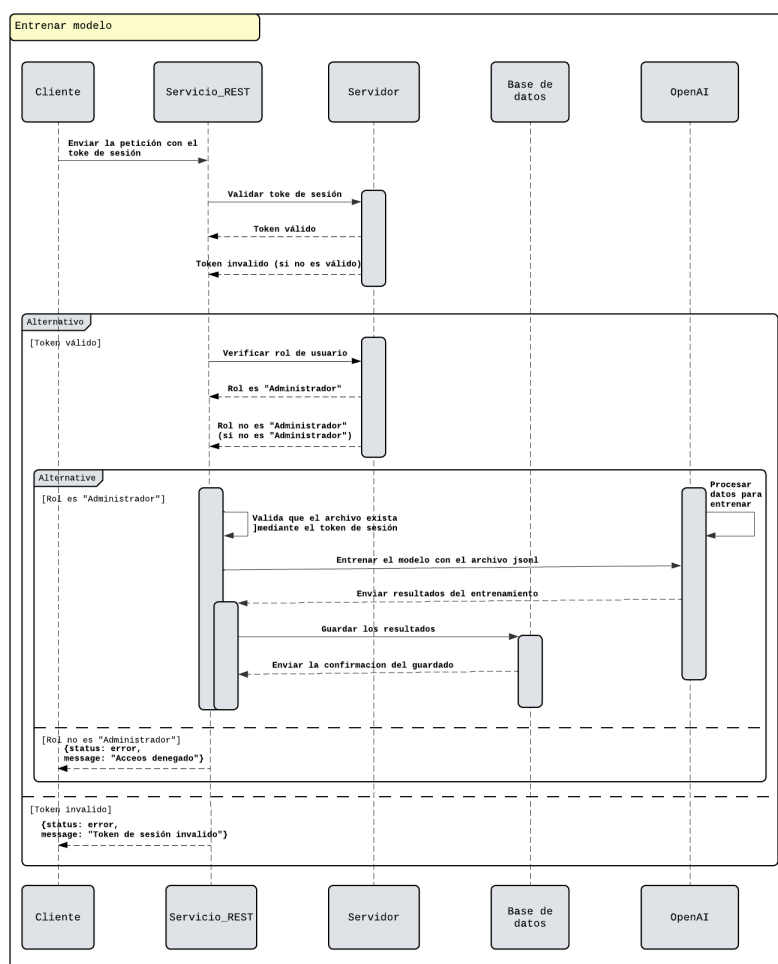


Figura 3.8: Diagrama de secuencia para explicar a detalle el proceso de entrenamiento del modelo mediante el archivo jsonl.

Este diagrama de secuencia ilustra cada paso necesario, desde la generación de los prompts hasta su estructuración en archivos JSONL, así como la ubicación específica donde estos archivos serán almacenados. el Cliente envía una petición con un token de sesión al Servicio REST, que verifica el token con el Servidor. Si el token es válido, el servidor procede a validar el rol del usuario (requiriendo el rol de “Administrador”). Si el rol es correcto, el sistema valida la existencia del archivo para el entrenamiento y envía los datos a la API de OpenAI para procesar el modelo, almacenando luego los resultados en la Base de Datos. Aunque en la implementación real el Servicio REST y el Servidor podrían ser el mismo objeto, se han separado en el diagrama para detallar mejor las responsabilidades: el Servicio REST maneja la interacción inicial con el cliente, mientras que el Servidor gestiona la lógica interna de validación de roles y el procesamiento de solicitudes complejas, asegurando claridad en el flujo de validación y manejo de datos.

La figura 3.8 proporciona una visión clara y estructurada del flujo de acciones necesarias para el entrenamiento del modelo, asegurando que cada etapa del proceso se realice de manera correcta. Comenzando con el Cliente enviando una petición que incluye un token de sesión al Servicio REST. El Servicio REST valida el token consultando con el Servidor. Si el token es válido, el flujo continúa; de lo contrario, se devuelve un mensaje de error al cliente. En el caso de que el token sea válido, el Servidor verifica si el usuario tiene el rol de “Administrador”. Si el rol es correcto, el flujo avanza para validar la existencia del archivo para el entrenamiento del modelo. El Servidor luego procede a enviar el archivo a la API de OpenAI para entrenar el modelo, y los resultados obtenidos son almacenados en la Base de Datos. Finalmente, el servidor envía una confirmación al cliente una vez que los resultados se han guardado correctamente.

Finalmente, la figura 3.10 muestra cómo el Cliente envía una petición con un token de sesión y un texto en lenguaje natural al Servicio REST, que valida el token consultando al Servidor. Si el token es válido, el servidor verifica el rol del usuario (debe ser Administrador o Desarrollador) y, si es correcto, envía el texto a la API de OpenAI para que lo interprete. La API devuelve una respuesta en formato JSON, que es reenviada al cliente con un mensaje de éxito. Si el rol o el token no son válidos, se devuelve un mensaje de error. Aunque el Servicio REST y el Servidor podrían ser un único componente, se han separado en el diagrama para detallar mejor las responsabilidades: el Servicio REST maneja la autenticación y el Servidor gestiona la lógica de roles y el envío de la solicitud a OpenAI.

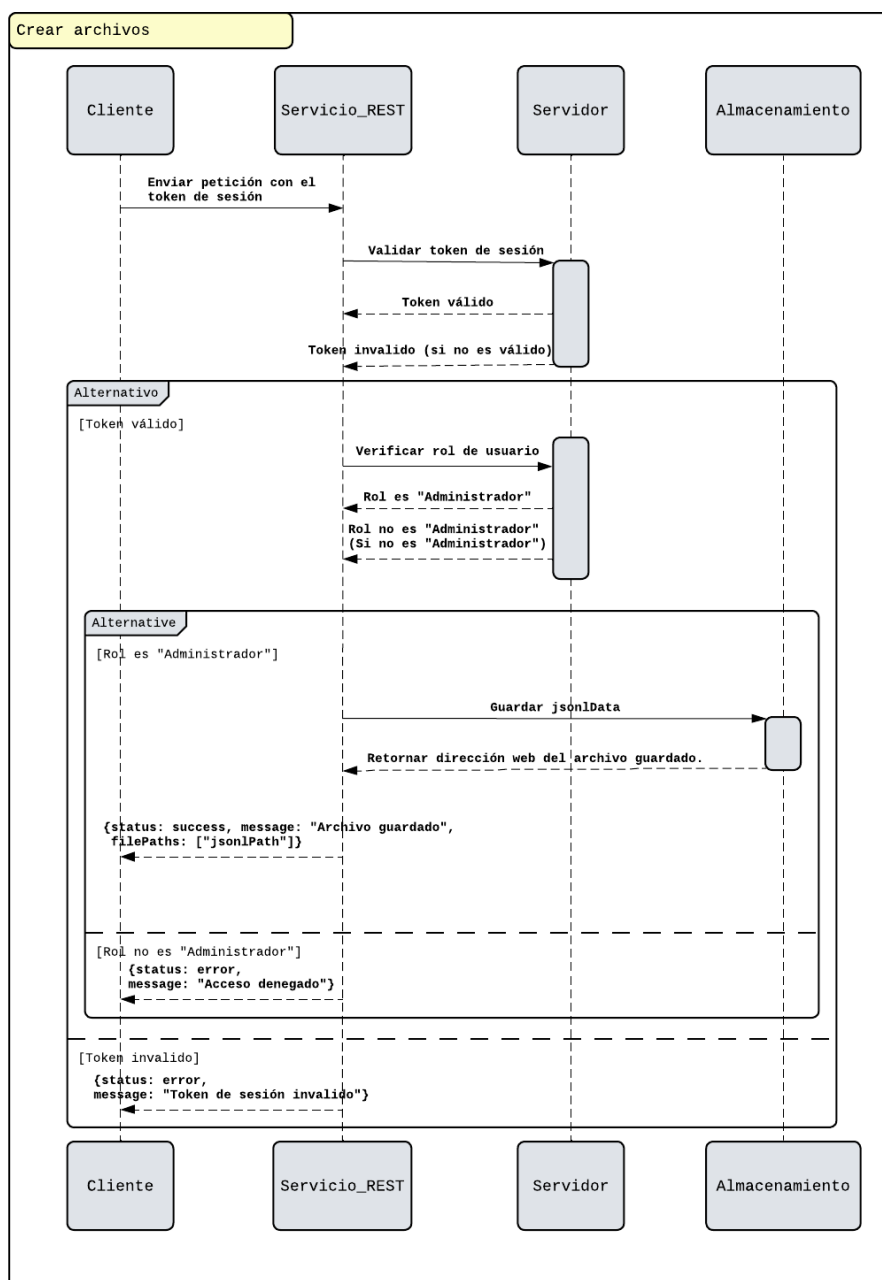


Figura 3.9: Diagrama de secuencia para explicar a detalle el proceso de crear los archivos jsonl.

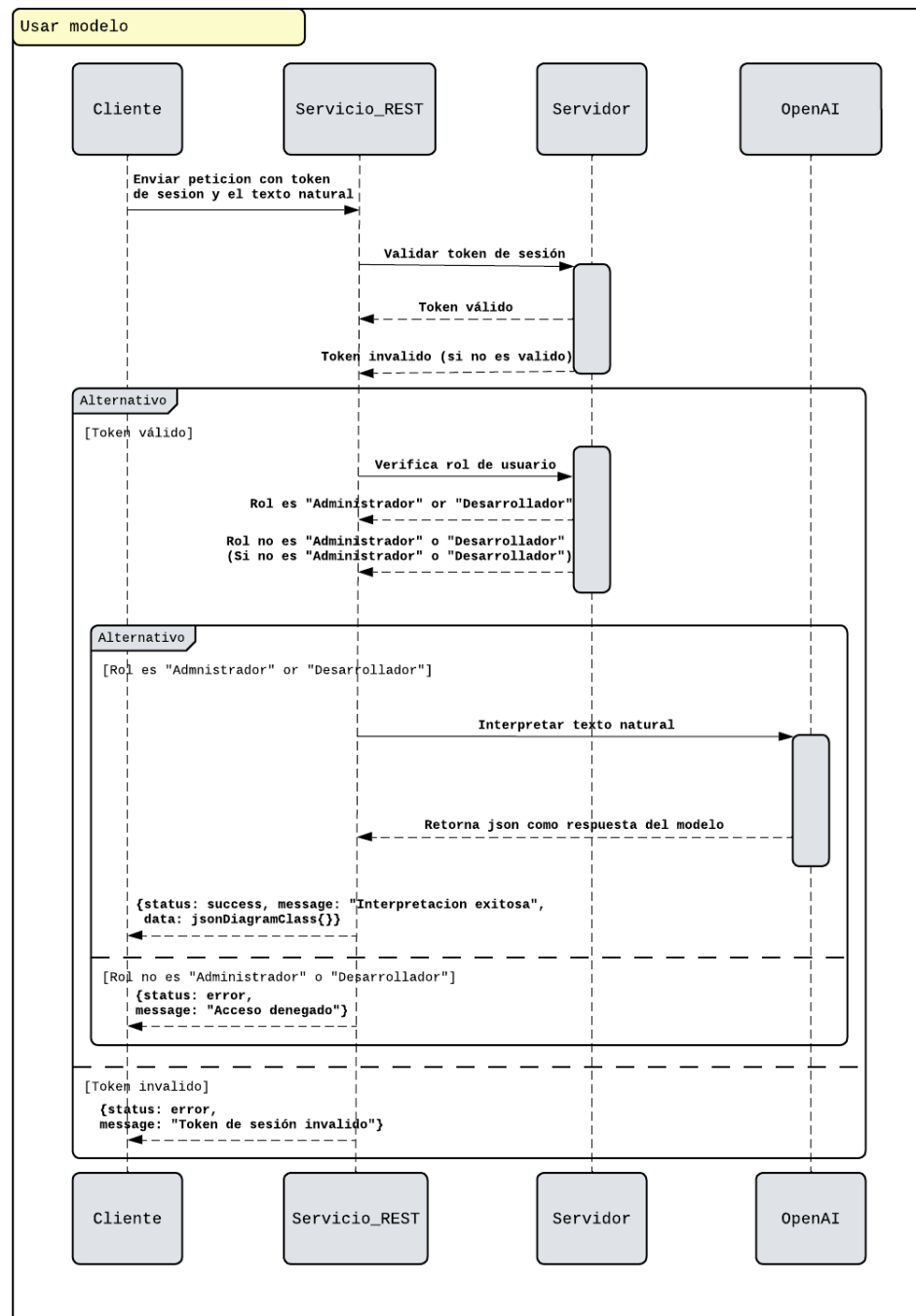


Figura 3.10: Diagrama de secuencia para explicar a detalle el proceso de usar el modelo.

- **ms_core_tddt4iots_py:** En este componente se desarrolló el servidor gRPC para lograr la comunicación con el componente ms-core-tddt4iots-openai, con el objetivo de mantener los microservicios REST de Spring conectados con el cliente de AngularJS. Se implementaron dos métodos principales: uno para el entrenamiento del modelo y otro para su utilización, asegurando una integración eficiente y efectiva entre los distintos componentes del sistema (ver figura 3.11).

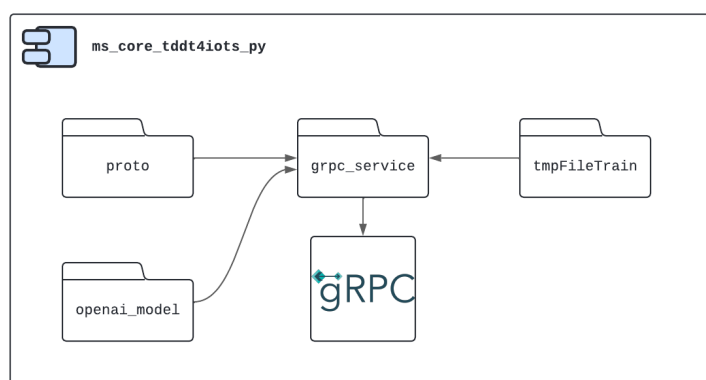


Figura 3.11: Paquetes desarrollados dentro del componente de python.

- *proto*: contiene los archivos `.proto` que definen las especificaciones de los servicios y mensajes gRPC, fundamentales para generar el código de comunicación.
- *grpc_service*: implementa estos servicios gRPC, facilitando la comunicación entre el servidor y otros componentes.
- *tmpFileTrain*: es responsable de almacenar los archivos JSONL descargados, que son utilizados para el entrenamiento de los modelos.
- *openai_model*: alberga toda la lógica necesaria para conectarse y interactuar con la API de OpenAI, manejando las solicitudes y respuestas para integrar las funcionalidades de OpenAI en el proyecto.

En la figura 3.12 se muestra el diagrama de flujo sobre el procedimiento seguido para el entrenamiento del modelo. El método recibe una solicitud con formato de cadena JSON con los parámetros necesarios, configura el cliente de OpenAI con una clave API, descarga el archivo JSONL desde la URL especificada, y lo guarda localmente

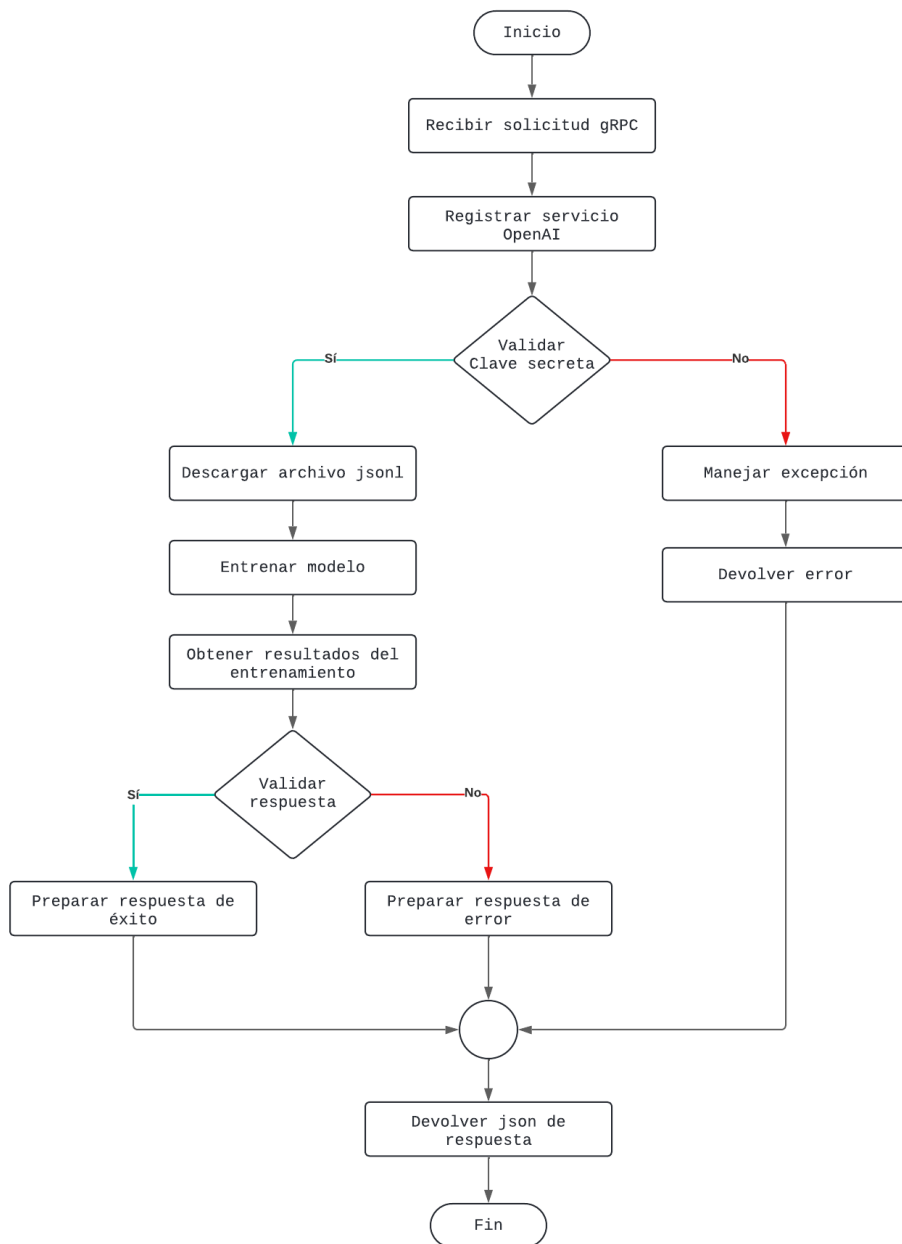


Figura 3.12: Entrenamiento de los modelos de OpenAI.

Luego, sube este archivo a OpenAI para iniciar el proceso de fine-tuning del modelo especificado. Si el proceso se realiza con éxito, devuelve un JSON indicando el estado del entrenamiento y los detalles del fine-tuning. En caso de error, retorna un mensaje de error.

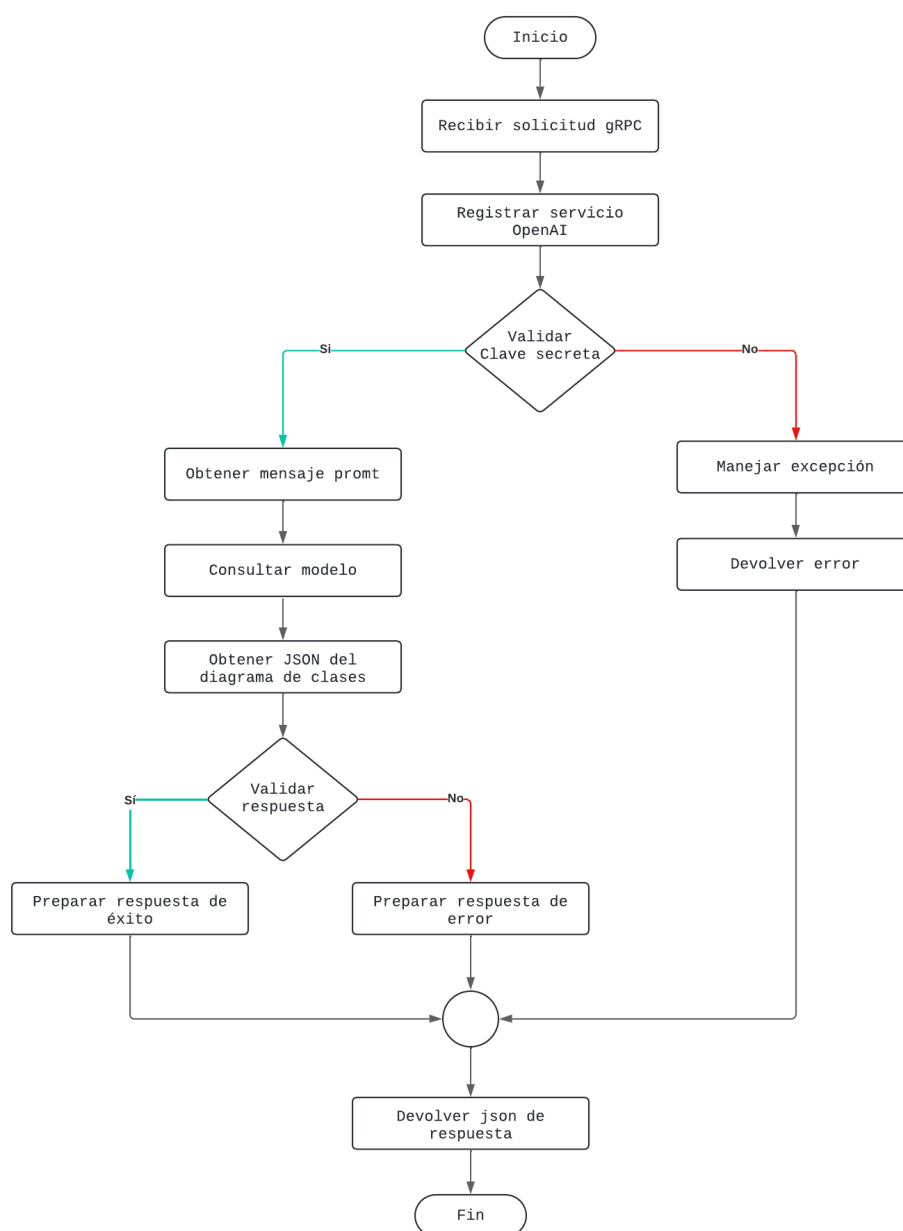


Figura 3.13: Uso de los modelos entrenados por OpenAI.

Para utilizar un modelo entrenado, es fundamental contar con un modelo base previamente entrenado con los datos extraídos de la herramienta. En la figura 3.13 se muestra el proceso paso a paso que se sigue para garantizar el correcto funcionamiento de la extensión y su uso eficiente con la API de OpenAI. El proceso es muy similar al anterior

con la deferencia que existe un paso denominado **Obtener mensaje prompt**. Este paso contiene el texto redactado en lenguaje natural y además 1 mensaje en específico detallando lo que debe realizar el modelo entrenado.

- **armadillo-api:** En la herramienta actual, había un intérprete denominado ArmadilloJs que permitía convertir las descripciones de los casos de uso redactados en un conjunto de símbolos, los cuales identificaban los componentes para su diagrama de clases. Este intérprete estaba desarrollado como una biblioteca de JavaScript. Sin embargo, se ha planteado la necesidad de transformar esta biblioteca en un servicio web, para poder utilizar sus funciones internamente dentro del componente que entrena el modelo de inteligencia artificial. El objetivo de esta transformación es interpretar las descripciones de los casos de uso de los proyectos existentes y obtener el texto en lenguaje natural, con el fin de entrenar el modelo de inteligencia artificial de manera más eficaz (ver figura 3.14).

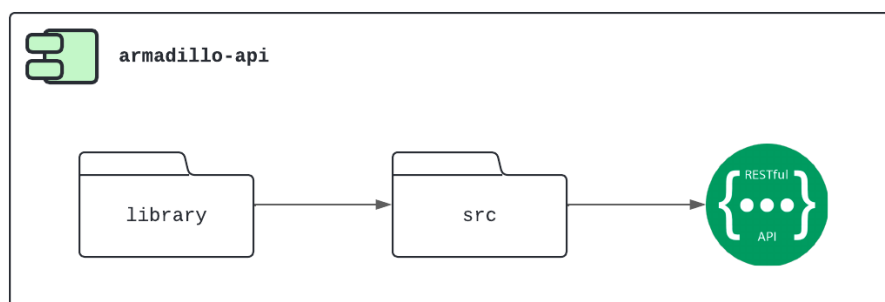


Figura 3.14: Estructura de los paquetes para el componente armadillo-api.

Herramientas de soporte al desarrollo de software

Durante el desarrollo de nuestro proyecto, se utilizaron diversas herramientas de software proporcionadas por JetBrains (<https://www.jetbrains.com>) para maximizar la eficiencia y productividad. JetBrains ofrece un conjunto robusto de entornos de desarrollo integrado (IDE) que se adaptan a diversas necesidades y lenguajes de programación. El uso de estas herramientas avanzadas nos permitió abordar diversas fases del desarrollo con mayor eficiencia y precisión. Las herramientas utilizadas y su aportación al desarrollo del software realizado se describen a continuación.

IntelliJ IDEA

Para el desarrollo en Java, utilizamos **IntelliJ IDEA 2024.1.4**, que pro-

porcionó un entorno robusto y eficiente. Este IDE fue fundamental para la integración de Spring Framework y la gestión de microservicios, permitiéndonos escribir, probar y depurar código de manera efectiva. Las características avanzadas de IntelliJ, como la autocompletación inteligente, el análisis de código en tiempo real y las herramientas de refactorización, mejoraron significativamente nuestra productividad y calidad del código.

WebStorm

En el desarrollo de interfaces web, **WebStorm 2023.3.1** fue la herramienta elegida por su capacidad para manejar tecnologías modernas como AngularJS. WebStorm facilitó la creación de interfaces web responsivas y dinámicas, proporcionando un entorno de desarrollo con soporte completo para JavaScript, TypeScript, HTML y CSS. Las capacidades de depuración en vivo y las herramientas de desarrollo front-end de WebStorm nos permitieron iterar rápidamente y mantener un alto estándar de calidad en nuestras interfaces de usuario.

PyCharm

Para los scripts en Python que gestionan la comunicación con la API de OpenAI, utilizamos **PyCharm 2023.3.4**. Este IDE es reconocido por su excelente soporte para el desarrollo en Python, especialmente en proyectos de inteligencia artificial y análisis de datos. PyCharm proporcionó herramientas integradas para la gestión de entornos virtuales, el análisis de código y la depuración, lo que facilitó la implementación de soluciones robustas y eficientes para la comunicación con OpenAI.

DataGrip

La gestión de bases de datos fue mejorada significativamente mediante el uso de **DataGrip 2023.2.3**. Esta herramienta nos permitió realizar un análisis eficiente y una integración perfecta con los demás componentes del sistema. DataGrip soporta múltiples sistemas de gestión de bases de datos, ofreciendo características avanzadas como la navegación por esquemas, la edición inteligente de SQL y la visualización de datos, lo que mejoró nuestra capacidad para gestionar y manipular grandes volúmenes de datos de manera efectiva.

NetBeans

En ciertos desarrollos específicos, utilizamos **NetBeans 22**, un IDE versátil que soporta múltiples lenguajes de programación. NetBeans facilitó el desarrollo de aplicaciones modulares y permitió una integración sencilla con diversas herramientas de desarrollo. Su interfaz intuitiva y sus poderosas características de depuración y gestión de proyectos lo hicieron una opción adecuada para tareas específicas dentro del proyecto.

Postman

Para probar y depurar las API, **Postman** fue una herramienta esencial. Postman nos permitió realizar solicitudes HTTP de manera eficiente, probar los endpoints del servidor y validar las respuestas. Su interfaz amigable y sus capacidades avanzadas de testing automatizado nos ayudaron a asegurar que todas las comunicaciones entre los componentes del sistema fueran precisas y confiables.

3.4. Metodología para desarrollar la aplicación web

La metodología Modelo-Vista-Controlador (MVC) [37] es una de las arquitecturas de software más utilizadas en el desarrollo de aplicaciones web debido a su capacidad para separar las preocupaciones, facilitando así el mantenimiento y escalabilidad del proyecto. Esta metodología divide la aplicación en tres componentes principales: el Modelo, la Vista y el Controlador.

- **Modelo:** Representa la lógica de la aplicación y la estructura de datos. Es responsable de gestionar los datos de la aplicación, respondiendo a las solicitudes del controlador y actualizando la vista cuando la información cambia. El modelo interactúa con la base de datos y maneja las reglas de negocio.
- **Vista:** Su principal función es mostrar los datos del modelo al usuario en un formato adecuado y capturar la entrada del usuario. La vista es independiente de la lógica del negocio, lo que permite cambiar la interfaz de usuario sin afectar al resto de la aplicación.
- **Controlador:** El controlador actúa como intermediario entre el modelo y la vista. Recibe las entradas del usuario a través de la vista, las procesa (interactuando con el modelo si es necesario) y devuelve la salida adecuada a la vista. El controlador contiene la lógica de la aplicación que responde a las acciones del usuario y gestiona las rutas de la aplicación.

La implementación de la metodología MVC permite desarrollar aplicaciones web de manera más estructurada y organizada, facilitando la colaboración entre los desarrolladores y la implementación de nuevas funcionalidades de manera eficiente.

3.4.1. Interfaces de la aplicación web

La extensión de la herramienta case conlleva a realizar varios ajustes y desarrollar nuevas pantallas y funcionalidades que se adapten y cumplan con el

objetivo de la investigación. En la figura 3.15 se observa como el nombre de usuario dentro de la pantalla principal de la aplicación, se agrega una sombra de color verde. Esta sombra de color verde indica la nueva implementación que existe con los modelos de inteligencia artificial.

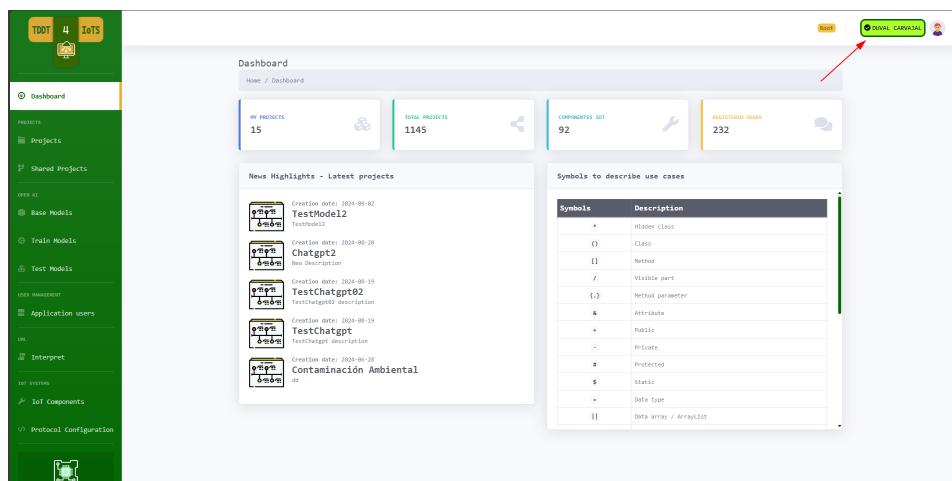


Figura 3.15: Interfaz principal de la aplicación.

La figura 3.16 muestra las nuevas opciones que se despliegan al dar click en el nombre de usuario. La opción *"Secret Key OpenAI"* permite poder ingresar la clave secreta de la API de OpenAI.

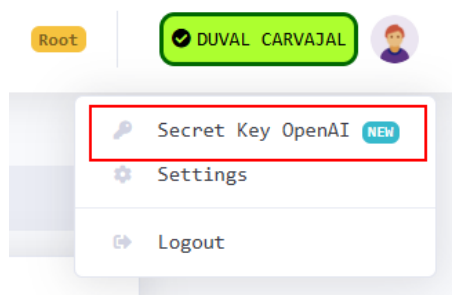


Figura 3.16: Opción para agregar la clave secreta de la API de OpenAI.

En la figura 3.17 se observa el formulario que aparecerá cuando ingresemos a la opción de *"Secret Key OpenAI"*.

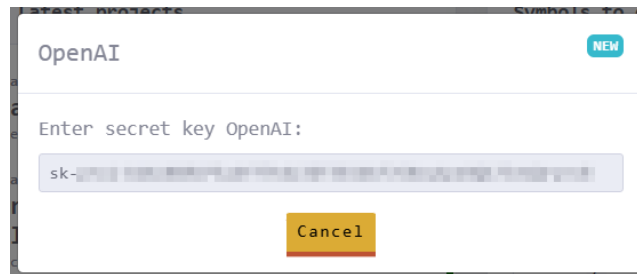
A screenshot of a web form titled "OpenAI" with a "NEW" badge in the top right corner. The form contains a label "Enter secret key OpenAI:" followed by a text input field. The input field contains the text "sk-" followed by a series of dots representing a masked key. Below the input field is a yellow "Cancel" button.

Figura 3.17: Formulario para ingresar la clave secreta de OpenAi.

La figura 3.18 muestra en el recuadro rojo la opción para configurar los modelos de OpenAI disponibles dentro de la herramienta.

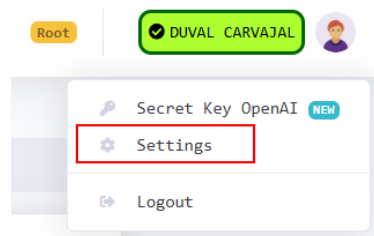


Figura 3.18: Opcion para configurar la API de OpenAI.

Al acceder a la opción de “Settings”, aparece un formulario que está dividido en dos secciones (ver figura 3.19). La opción de “OpenAI” muestra los modelos disponibles en la herramienta.

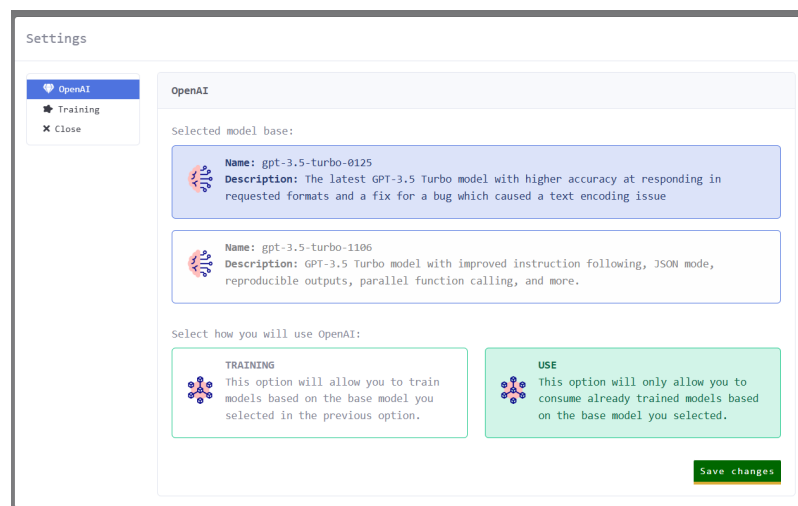
A screenshot of the "Settings" page. On the left is a sidebar with three items: "OpenAI" (selected with a heart icon), "Training" (with a star icon), and "Close" (with an X icon). The main content area is titled "OpenAI" and contains two sections. The first section, "Selected model base:", lists two models: "gpt-3.5-turbo-0125" with a description about higher accuracy and a bug fix, and "gpt-3.5-turbo-1106" with a description about improved instruction following and JSON mode. The second section, "Select how you will use OpenAI:", has two options: "TRAINING" (allowing training on the selected base model) and "USE" (allowing consumption of already trained models). A "Save changes" button is at the bottom right.

Figura 3.19: Formulario de configuración de los modelos de OpenAi. Modelos base y el uso que le dará el usuario.

En la primera subsección, se visualizan los modelos base de la herramienta, destacándose en color azul el modelo base que ha sido seleccionado por el usuario. En la segunda subsección, se muestra el tipo de uso que el usuario está aplicando al modelo. En este caso, el modelo base seleccionado está siendo utilizado específicamente para interpretar el texto natural enviado a través de las descripciones de los casos de uso.

La sección de “*Training*” muestra la información sobre los entrenamientos realizados sobre el modelo base que se encuentra seleccionado. Además, se observa el nombre del usuario quien realizó el entrenamiento respectivo al modelo (ver figura 3.20).

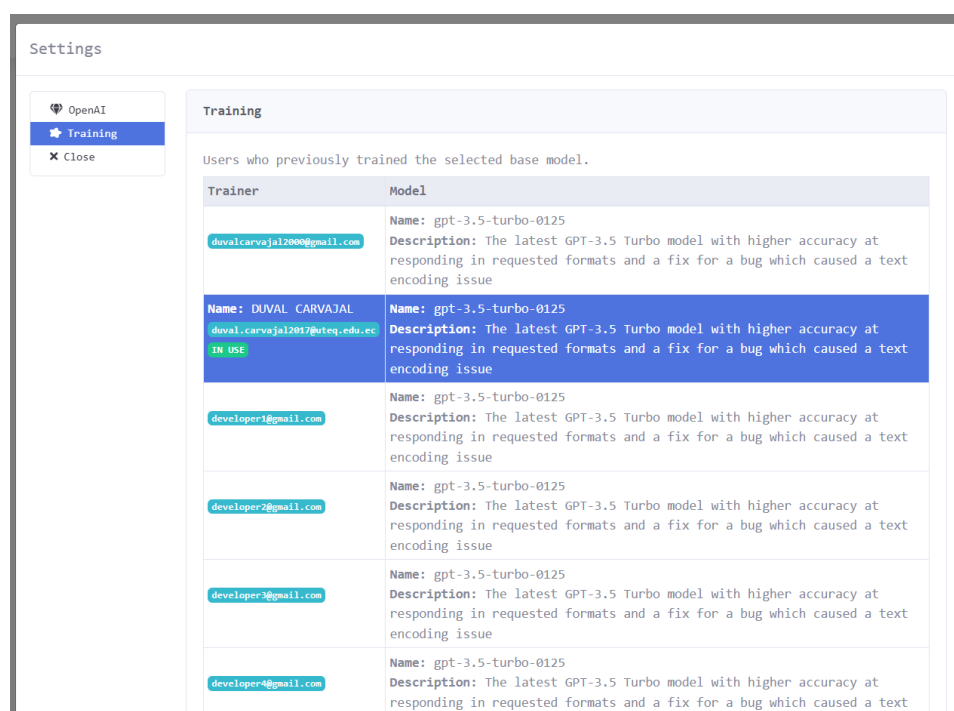


Figura 3.20: Formulario de configuración de los modelos de OpenAi. Entrenamiento de los modelos base y su modelo seleccionado.

Dentro de la herramienta se agregaron 3 opciones al menú principal con el objetivo de gestionar los modelos base que podrán ser ingresados, los entrenamientos y una ultima opción para poder probar el modelo entrenado de forma independiente a los proyectos que tenga la herramienta. La figura 3.21 muestra la opción “*Base Models*” que permitirá ingresar los modelos base de OpenAi que se crean necesarios para ser utilizados por la herramienta.

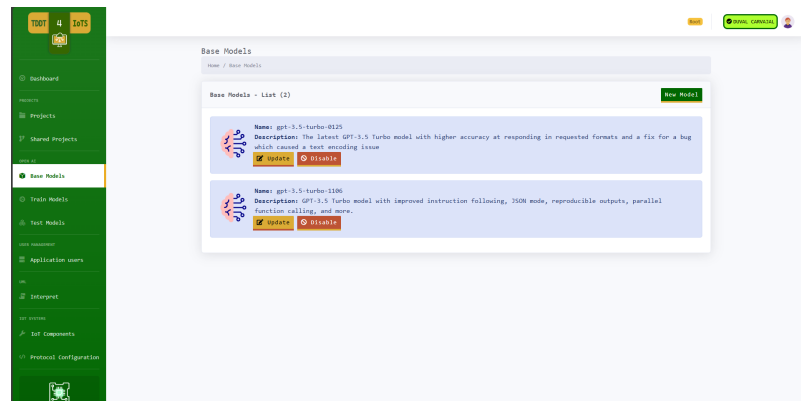


Figura 3.21: Opción para gestionar los modelos base de OpenAI dentro de la herramienta.

La figura 3.22 muestra la opción “*Train models*”. En esta opción se podrán gestionar los entrenamientos al modelo base seleccionado. Se puede observar una lista de los entrenamientos y el estado en el que se encuentra o quedo cada entrenamiento que fue realizado desde la herramienta. Los datos para el entrenamiento son en base a los datos que actualmente se encontraron en la herramienta.

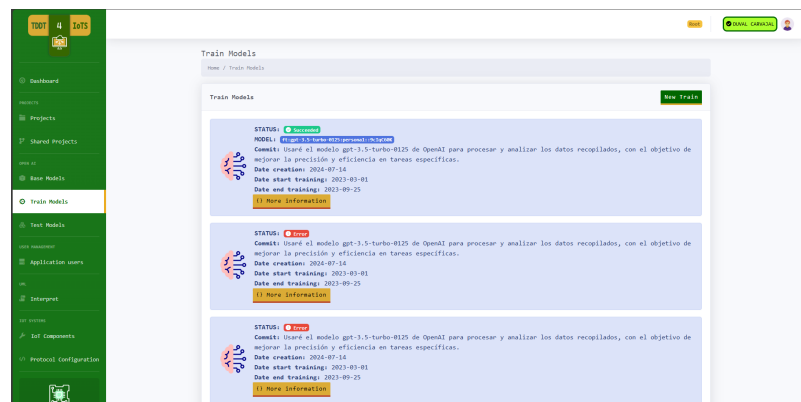


Figura 3.22: Opción para gestionar los entrenamientos de los modelos base de OpenAI dentro de la herramienta.

Para conocer mas información detallada sobre cada entrenamiento realizado, se puede ingresar a la opción de “*More information*” (ver figura 3.23). Al ingresar a esa opción en la figura 3.24 se observa como se desplegara un formulario con el *JSON* de respuesta devuelto por OpenAI del entrenamiento realizado.

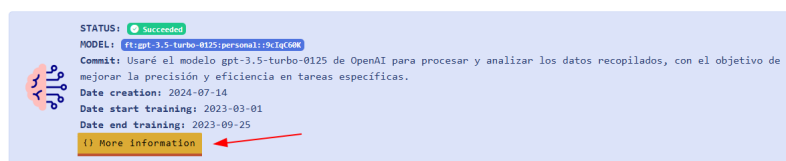


Figura 3.23: Estado de un entrenamiento realizado dentro de la herramienta y sus opciones disponibles.

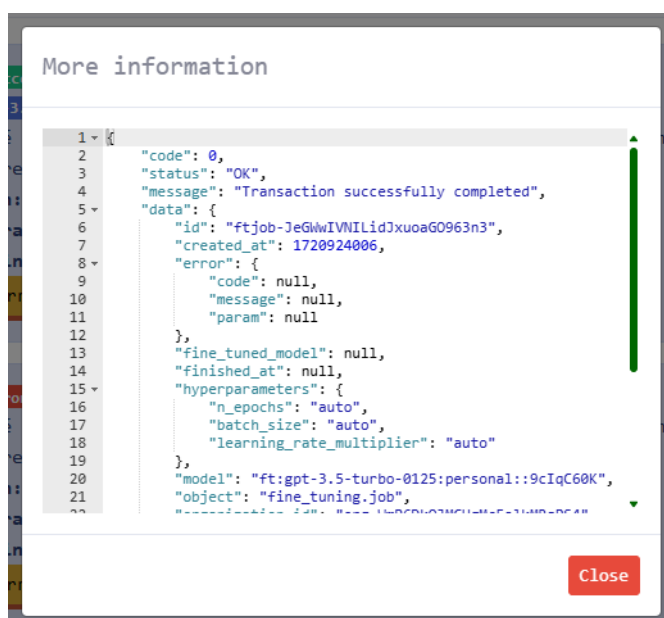


Figura 3.24: Información detallada del entrenamiento.

La figura 3.25 muestra la última opción denominada “*Test Models*”. Esta opción permite al usuario probar el modelo seleccionado para utilizarlo dentro de la herramienta *case*. Para realizar la prueba, solo se requiere proporcionar un texto en lenguaje natural que describa una parte de los casos de uso del sistema informático que se esté desarrollando en ese momento mediante la herramienta. Luego, se espera la respuesta de OpenAI, y la herramienta genera el diagrama de clases correspondiente según la solicitud descrita en el texto enviado.

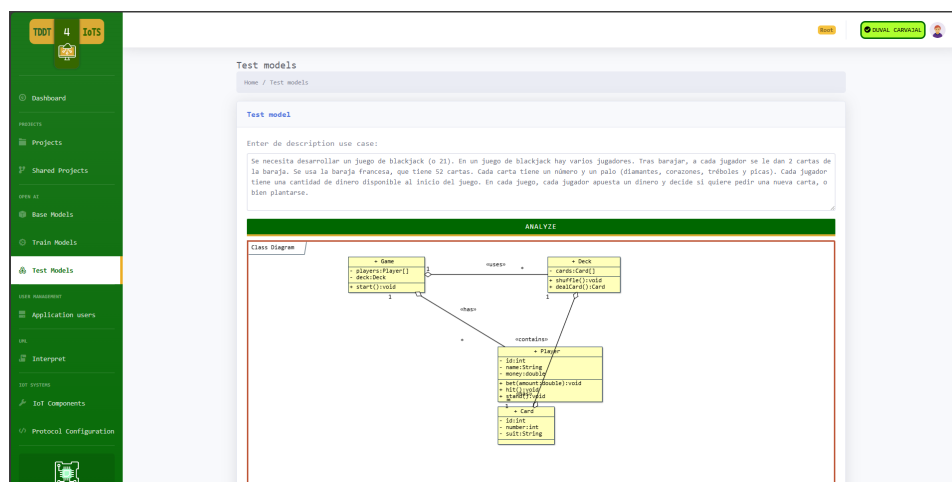


Figura 3.25: Prueba del modelo de OpenAI entrenado.

Una vez que el modelo entrenado esté listo y se haya comprobado que puede analizar correctamente la descripción del caso de uso enviado, se puede proceder a la siguiente etapa de la herramienta. En esta etapa, se utilizará el modelo con todas las descripciones de los casos de uso del sistema de interés para generar un diagrama de clases lo más ajustado al objetivo del sistema a desarrollar. En la figura 3.26, se muestra cómo la flecha señala una opción denominada “*Activate OpenAI*”. Permite activar o desactivar el uso del modelo para analizar las descripciones de los casos de uso. Dado que la herramienta incluye un intérprete de descripciones de casos de uso, esta opción determina si se desea emplear el modelo en el análisis o no.

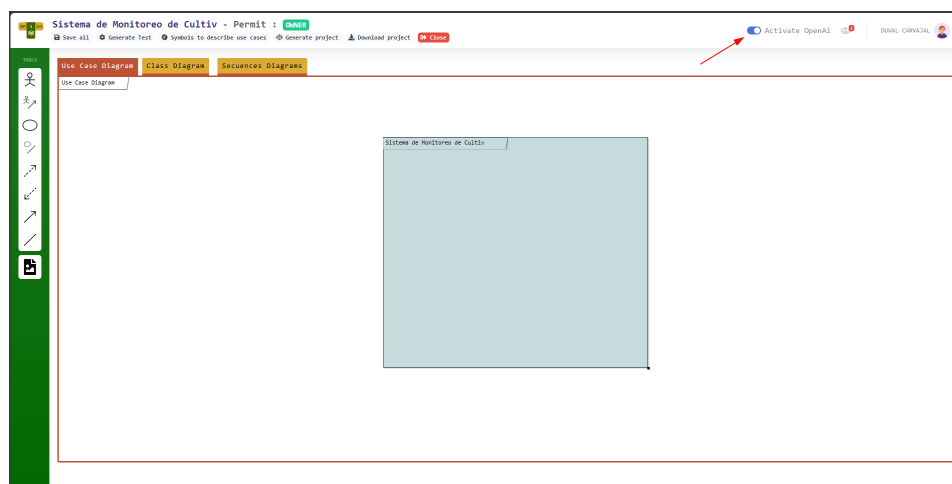


Figura 3.26: Área para crear un diagrama de casos de uso e indicar si se usará el modelo de OpenAI.

La figura 3.27 muestra como la descripción del caso de uso actual esta ingresada con texto natural. Esperando que el modelo sea capaz de interpretar cada sección del caso de uso y también todos los casos de uso que permitan crear el diagrama de clases.

Figura 3.27: Formulario para ingresar la descripción de cada caso de uso del sistema.

Cuando el modelo este trabajando sobre los casos de uso se podrá observar una leyenda que indicara el momento que OpenAI está analizando las descripciones de los casos de uso (ver figura 3.28).

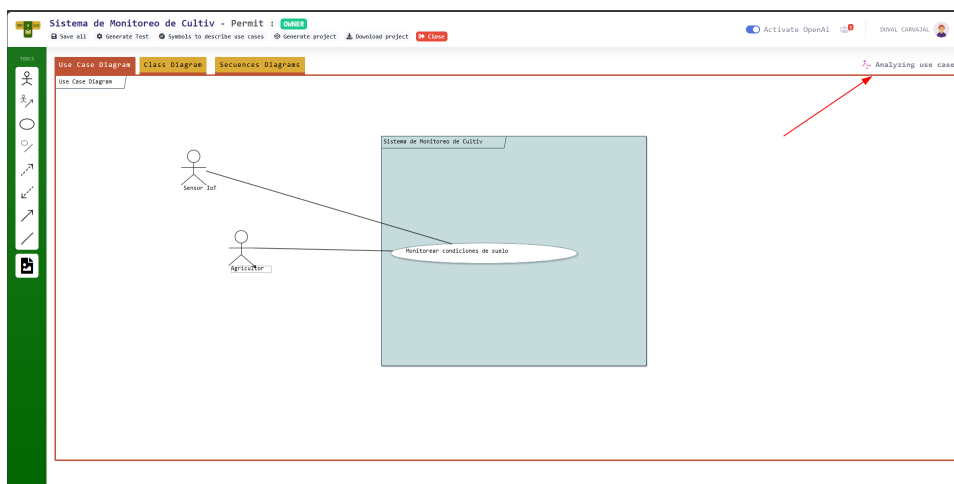


Figura 3.28: Estado del modelo al analizar los casos de uso.

Finalmente, en la figura 3.29 se muestra como la herramienta fue capaz de

generar el diagrama de clases mediante las descripciones de los casos de uso del sistema a desarrollar.

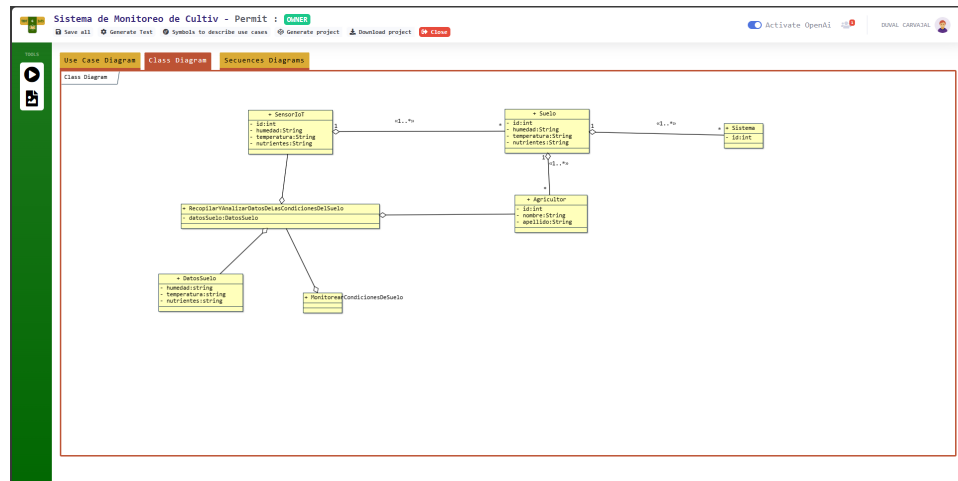


Figura 3.29: Diagrama de clases generado por la herramienta y el modelo de OpenAI.

3.4.2. Evaluación de la extensión dentro de la aplicación web

Para evaluar la efectividad de la herramienta, se contactó con 15 estudiantes de la Universidad Técnica Estatal de Quevedo, Ecuador. A estos estudiantes se les solicitó que utilizaran la herramienta y cargaran uno de los proyectos en los que estuvieran trabajando durante sus estudios, creando el diagrama de casos de uso para su sistema IoT. Una vez redactadas las descripciones de los casos de uso en lenguaje natural, la herramienta analizó cada proyecto y generó automáticamente el diagrama de clases correspondiente para cada proyecto.

A cada estudiante se le envió la invitación a un documento compartido para que pudieran ingresar las descripciones de los casos de uso y tener documentado todos los proyectos que se ingresaron en la herramienta para su respectiva evaluación. En el siguiente enlace (<https://acortar.link/pqSAZi>) se muestra el documento de texto que fue compartido y se observan las descripciones de los casos de uso de cada proyecto.

Para las descripciones de los casos de uso se les indicó una plantilla para que puedan documentarla y sea más fácil ingresarlos dentro de la herramienta.

3.4.3. Ejemplo de proyecto usado para evaluar la extensión de la herramienta

A continuación se mostrará un ejemplo de uno de los proyectos evaluados.

Proyecto: Sistema de Monitoreo de Cultivos Inteligente con IoT

Descripción: Desarrollo de una red de sensores IoT para monitorear condiciones del suelo, humedad y temperatura en cultivos, optimizando el riego y mejorando la productividad agrícola.

Caso de uso: Monitorizar condiciones de suelo	
Tipo:	Primario
Actores:	Sensor IoT, Agricultor
Propósito:	Permitir al sistema recopilar y analizar datos de las condiciones del suelo (como humedad, temperatura y nutrientes) para ayudar en la toma de decisiones del agricultor.
Precondición:	El Sensor IoT debe estar correctamente instalado y configurado para transmitir datos al sistema. El agricultor debe tener acceso al sistema.
Postcondición:	Los datos del suelo se registran en el sistema y están disponibles para su visualización y análisis.
Descripción:	Este caso de uso permite que el sistema recoja y analice datos del suelo utilizando sensores IoT. Los datos recopilados son accesibles para el agricultor en tiempo real, y se pueden usar para la toma de decisiones sobre el manejo de los cultivos.
Flujo normal	
Actor	Sistema
1. El Sensor IoT recopila datos del suelo en intervalos regulares.	
	2. El sistema recibe y almacena los datos.

	3. El sistema analiza los datos para detectar patrones o tendencias.
4. El agricultor puede acceder a los datos en tiempo real a través del sistema.	
Flujo alternativo	
Actor	Sistema
1. Si el Sensor IoT no puede transmitir los datos, el sistema genera una notificación al agricultor para tomar acciones correctivas.	

Caso de uso: Notificar anomalías	
Tipo:	Primario
Actores:	Agricultor
Propósito:	Informar al agricultor sobre cualquier condición anómala detectada en el suelo para que pueda tomar medidas inmediatas.
Precondición:	El sistema debe estar monitoreando las condiciones del suelo activamente y contar con una base de datos de condiciones normales.
Postcondición:	El agricultor es informado de la anomalía y puede tomar acciones correctivas.
Descripción:	El sistema detecta automáticamente cualquier condición anómala en el suelo y notifica al agricultor para que pueda actuar rápidamente. Este proceso asegura que el agricultor esté siempre informado de posibles problemas que puedan afectar sus cultivos.
Flujo normal	
Actor	Sistema

	1. El sistema detecta una condición anómala en los datos del suelo (ej. baja humedad, presencia de plagas).
	2. El sistema genera una notificación automática.
3. El agricultor recibe la notificación en su dispositivo.	
Flujo alternativo	
Actor	Sistema
1. Si el agricultor no responde a la notificación, el sistema envía recordatorios o escala la notificación a otros medios (como SMS).	

Caso de uso: Visualizar datos en tiempo real	
Tipo:	Primario
Actores:	Agricultor
Propósito:	Permitir al agricultor acceder a una interfaz que muestre los datos actuales de las condiciones del suelo, facilitando el monitoreo continuo.
Precondición:	El sistema debe estar recibiendo y procesando datos del Sensor IoT en tiempo real.
Postcondición:	Los datos se muestran correctamente y están actualizados en la interfaz del usuario.
Descripción:	El agricultor puede monitorear las condiciones del suelo en tiempo real mediante una interfaz visual proporcionada por el sistema, lo que le permite tomar decisiones informadas sobre el manejo de sus cultivos.
Flujo normal	
Actor	Sistema

1. El agricultor accede al sistema a través de su dispositivo.	
	2. El sistema muestra los datos en una interfaz gráfica en tiempo real.
3. El agricultor puede observar y analizar los datos para tomar decisiones inmediatas.	
Flujo alternativo	
Actor	Sistema
1. Si la conexión con el Sensor IoT falla, el sistema muestra una advertencia y proporciona los últimos datos disponibles.	

Caso de uso: Automatizar riego	
Tipo:	Primario
Actores:	Agricultor
Propósito:	Automatizar el proceso de riego basado en los datos del suelo, optimizando el uso de agua y asegurando que las plantas reciban la cantidad adecuada de humedad.
Precondición:	El sistema debe tener acceso a un controlador de riego automatizado y los datos del Sensor IoT deben estar disponibles.
Postcondición:	El riego se activa o desactiva automáticamente según las condiciones del suelo.
Descripción:	Este caso de uso permite al sistema tomar decisiones automatizadas sobre el riego de cultivos basándose en los datos del suelo, optimizando así los recursos hídricos y garantizando la salud de las plantas.
Flujo normal	

Actor	Sistema
	1. El sistema analiza los datos del Sensor IoT y determina si el suelo necesita riego.
	2. Si se requiere riego, el sistema activa el sistema de riego automatizado.
	3. El sistema monitorea el riego y lo detiene una vez que se alcanzan las condiciones óptimas.
Flujo alternativo	
Actor	Sistema
1. Si el sistema de riego no responde, el sistema genera una notificación.	

La figura 3.30 muestra el diagrama de casos de uso del proyecto indicado.

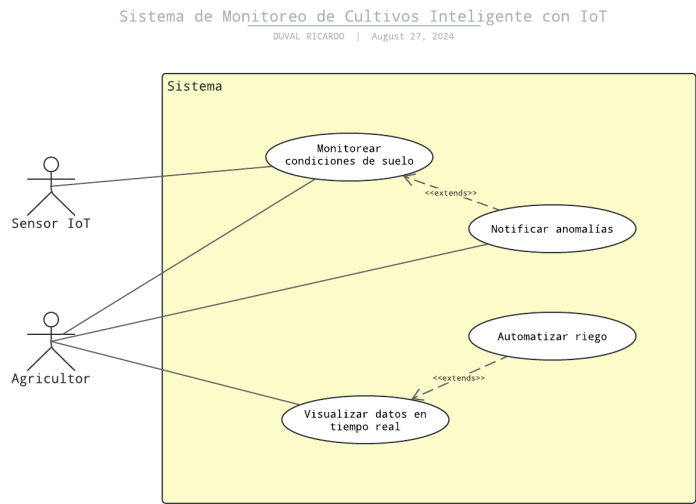


Figura 3.30: Diagrama de casos de uso para el Sistema de Monitoreo de Cultivos Inteligente con IoT.

Todos los proyectos que sirvieron para evaluar la extensión están realizados de esta forma, con el objetivo de obtener información lo más apegado a la realidad. Dichos proyectos son trabajos de los estudiantes que presentaran en sus asignaturas respectivas.

Capítulo 4

Resultados y discusión

Durante la implementación del proyecto, se evaluaron diversas tecnologías con el objetivo de seleccionar aquella que mejor se ajustara a los plazos de desarrollo y permitiera cumplir los objetivos establecidos. Aunque inicialmente se consideró la creación de un modelo de inteligencia artificial desde cero utilizando Python, los resultados indicaron que el tiempo necesario para entrenar el modelo y estabilizar sus respuestas era considerablemente elevado, lo que hacía inviable esta opción dentro de los plazos previstos.

Por otro lado, la implementación de modelos preentrenados, como los ofrecidos por OpenAI, resultó ser una alternativa mucho más eficiente. Estos modelos proporcionaron respuestas consistentes y claras tras un proceso de ajuste mínimo. Los resultados reflejan que, aunque fue necesario un grado de entrenamiento para adaptar los modelos a los objetivos específicos del proyecto, el tiempo total de desarrollo y ajuste fue significativamente menor en comparación con el desarrollo de un modelo desde cero. Además, la integración de estos modelos con la herramienta *case* preexistente fue fluida, lo que facilitó alcanzar los resultados esperados en un marco temporal reducido.

Para verificar el desempeño de los modelos de OpenAI integrados en la herramienta *case*, se realizó una prueba piloto con 15 estudiantes, quienes ingresaron descripciones de casos de uso relacionados con sistemas IoT. Posteriormente, se realizó una encuesta para evaluar la satisfacción de los estudiantes y determinar si la nueva implementación de la herramienta *case* fue útil para su aprendizaje en el área de Software.

Los resultados de la encuesta, que se presentan a continuación mediante gráficos generados en R para facilitar su visualización, revelan la percepción de los estudiantes sobre la efectividad de la herramienta. Estos hallazgos permiten comprender de manera más lógica y clara los resultados obtenidos, destacando la mejora gracias a la extensión de la herramienta *case* mediante

los modelos de OpenAI.

La figura 4.1 muestra los resultados obtenidos de la encuesta y a continuación analizaremos cada uno de ellos:

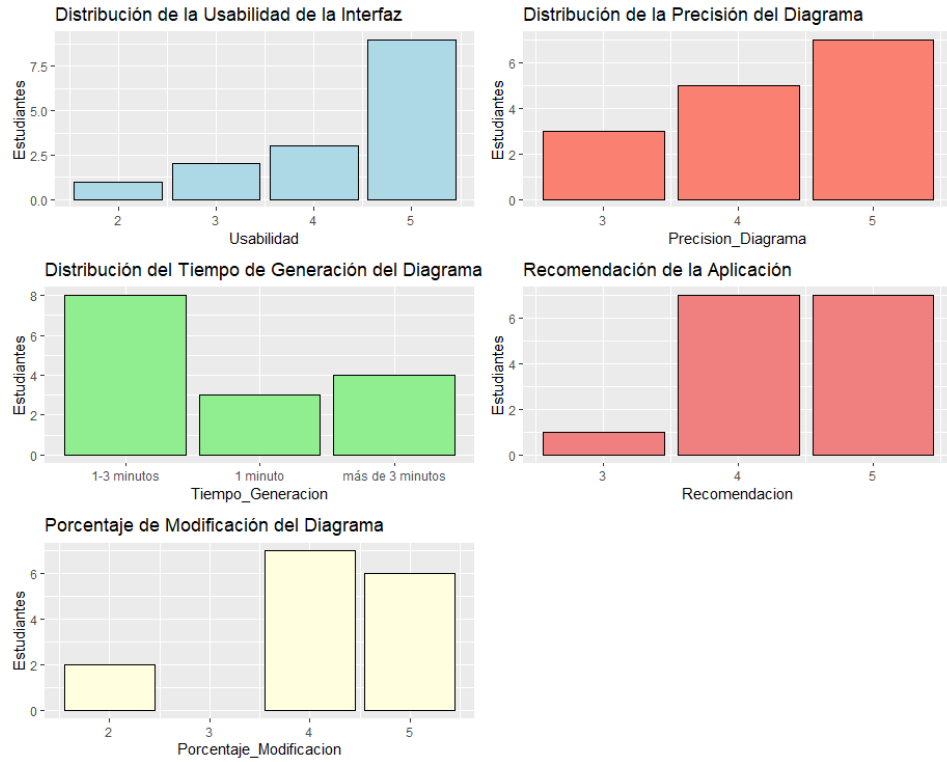


Figura 4.1: Resultados de la encuesta.

4.1. Distribución de la Usabilidad de la Interfaz

Este gráfico muestra que la mayoría de los estudiantes (aproximadamente 8) calificaron la usabilidad de la interfaz con un 5, lo que indica una alta satisfacción con la interfaz de usuario. Un menor número de estudiantes calificó la usabilidad con 3 y 4, mientras que solo unos pocos le dieron una calificación más baja (2).

Interpretación: Esto sugiere que la interfaz de la herramienta es bien valorada por la mayoría de los usuarios, lo cual es un indicador positivo de la facilidad de uso del sistema implementado.

4.2. Distribución de la Precisión del Diagrama

La mayoría de los estudiantes calificaron la precisión del diagrama generado con un 5 (el puntaje más alto), seguido por un número significativo que le otorgó un 4. Solo algunos estudiantes dieron una calificación de 3 y ninguno puntuó por debajo de esto.

Interpretación: Los usuarios perciben que los diagramas generados por la herramienta tienen un alto nivel de precisión, lo que sugiere que el sistema cumplió bien con sus expectativas en cuanto a la representación de los casos de uso.

4.3. Distribución del Tiempo de Generación del Diagrama

La mayor parte de los estudiantes (aproximadamente 8) reportaron que el tiempo de generación del diagrama fue entre 1 y 3 minutos. Un menor número de estudiantes indicó que el tiempo fue de un minuto o de más de tres minutos.

Interpretación: La mayoría de los estudiantes percibió que el tiempo de generación del diagrama fue razonable y adecuado para el contexto del uso de la herramienta. Sin embargo, un pequeño grupo experimentó tiempos de generación más largos, lo que podría ser una oportunidad de mejora en la optimización de la herramienta.

4.4. Recomendación de la Aplicación

La gran mayoría de los estudiantes calificó la recomendación de la aplicación con un 4 o 5, lo que sugiere que recomendarían su uso a otros. Solo uno o dos estudiantes dieron una calificación de 3.

Interpretación: Esto indica una alta satisfacción general con la herramienta, lo cual es un resultado importante que refleja la utilidad percibida y la disposición de los usuarios a recomendarla.

4.5. Porcentaje de Modificación del Diagrama

Este gráfico muestra que la mayoría de los estudiantes realizaron pocas modificaciones al diagrama, puntuando entre 4 y 5 en cuanto al porcentaje de modificación necesario. Algunos usuarios indicaron que se requirieron modificaciones moderadas (calificación de 3), y solo un estudiante reportó un

alto porcentaje de modificaciones.

Interpretación: Aunque los diagramas generados fueron precisos, algunos estudiantes aún necesitaron realizar ciertos ajustes para que los diagramas se ajustaran completamente a los casos de uso. Esto sugiere que el sistema genera diagramas cercanos a la solución final, con solo algunos ajustes menores por parte de los usuarios.

4.6. Conclusiones

En general, los resultados indican que la herramienta implementada, que utiliza los modelos preentrenados de OpenAI integrados en la herramienta *case*, fue bien recibida por los estudiantes. La interfaz se evaluó como fácil de usar, los diagramas generados tuvieron una buena aceptación, y los tiempos de generación fueron adecuados para la mayoría de los casos. Además, la mayoría de los usuarios indicaron que recomendarían la herramienta a otros. Aunque los diagramas generados necesitaban algunas modificaciones, estas eran mínimas, lo que refleja un buen desempeño en términos de precisión.

Capítulo 5

Conclusiones y trabajo futuro

En este capítulo se presentan las principales conclusiones derivadas del desarrollo de este trabajo, destacando los resultados obtenidos y su impacto en el ámbito del desarrollo de sistemas IoT. Además, se discuten las posibles líneas de investigación futuras y mejoras que pueden implementarse en la herramienta desarrollada, con el fin de incrementar su funcionalidad a contextos más complejos.

5.1. Principales conclusiones del trabajo desarrollado

El objetivo de extender y mejorar la herramienta TDDT4IoTS mediante la integración de inteligencia artificial se ha cumplido con éxito. La implementación de modelos preentrenados, como los ofrecidos por OpenAI, permitió mejorar el desarrollo de tareas críticas al momento de crear sistemas IoT, mejorando la reducción de errores humanos. La evaluación con los estudiantes mostró que los diagramas generados tuvieron una buena aceptación, con tiempos de generación razonables y una alta satisfacción respecto a la usabilidad de la interfaz, lo que indica que las mejoras implementadas e incorporadas a la herramienta se vieron reflejadas en el nivel de satisfacción de los usuarios que han usado dicha herramienta.

El análisis de las tecnologías de inteligencia artificial disponibles concluyó que los modelos preentrenados de OpenAI eran uno de los más adecuados para ser integrados en TDDT4IoTS, dado su flexibilidad y tiempos de entrenamiento reducidos. A pesar de que se consideró inicialmente desarrollar un modelo desde cero, la alternativa de los modelos preentrenados resultó ser

mas adaptable, lo que permitió avanzar rápidamente en la integración de las nuevas funcionalidades implementadas en de la herramienta *case* existente.

El resultado de aplicar la implementación realizada para que modelos de IA hicieran el análisis automático de descripciones textuales en forma de casos de uso fue considerablemente buena. Los modelos integrados permitieron identificar la mayor parte de elementos clave, como clases, atributos y relaciones, a partir de las especificaciones de los sistemas IoT. Los resultados reflejaron que los estudiantes encontraron útiles los diagramas generados, siendo necesario aplicar solo pequeñas modificaciones a dichos diagramas, lo que confirma que la automatización del análisis de las especificaciones como entrada a la herramienta cumplió con los objetivos propuestos.

El módulo de generación automática de diagramas de clases demostró ser correcto, ya que los diagramas generados fueron cercanos a la solución final. Si bien algunos usuarios indicaron la necesidad de realizar ajustes manuales menores, esto es normal en procesos de diseño asistido. Los tiempos de generación, evaluados por los estudiantes, fueron en su mayoría satisfactorios, con tiempos entre 1 y 3 minutos, lo que cumple con las expectativas de un proceso automatizado dentro del desarrollo de sistemas IoT.

Finalmente, la herramienta TDDT4IoTS ha sido significativamente mejorada mediante la integración de modelos de IA, logrando un avance importante en la automatización de tareas que anteriormente requerían intervención manual. Esto no solo mejora la eficiencia del proceso de desarrollo de sistemas IoT, sino que también trata de minimizar los errores humanos, aumentando la productividad de los ingenieros de software que desarrollen sistemas IoT. Con todo ello, se ha conseguido mejorar la herramienta, proporcionando una solución robusta para los desarrolladores de este tipo de sistemas.

5.2. Trabajo futuro

Para continuar con la línea de investigación desarrollada en este trabajo, me complace comunicar que seguiré mis estudios de doctorado en la Universidad de Granada. En este contexto, planeo explorar diversos aspectos que permitirán mejorar y agregar nuevas funcionalidades a la herramienta, con el objetivo de ampliar su aplicabilidad y eficacia en el ámbito del desarrollo de sistemas.

- **Personalización de los prompts de entrenamiento:** Actualmente, la herramienta utiliza un prompt por defecto para entrenar los modelos. Una mejora sería permitir al usuario personalizar el prompt de entrenamiento según el modelo seleccionado. Esto ayudaría al modelo a entrenarse de manera más ajustada a las necesidades específicas

del usuario (ingeniero de software), facilitando una mejor comprensión de las especificaciones del sistema IoT que se desea desarrollar.

- **Integración con documentación técnica:** Se podría mejorar la capacidad de los modelos de OpenAI para trabajar con documentación técnica más detallada, como informes o manuales de sistemas IoT. A través de la integración de estos documentos en el prompt o el uso de procesamiento en múltiples pasos, la herramienta podría generar una comprensión más profunda y precisa de los sistemas, lo que permitiría generar diagramas más adecuados a partir de la documentación.
- **Chat de ayuda para redactar casos de uso extendidos:** Una mejora futura sería integrar un chat, utilizando la API de OpenAI, que asista al usuario en la redacción de casos de uso extendidos. Esto permitiría aclarar de manera más eficiente los requisitos funcionales y no funcionales de un sistema IoT. Al ayudar al usuario a describir mejor las especificaciones del sistema, el modelo podrá comprender con mayor precisión el contexto, lo que facilitará la generación de un diagrama de clases más claro y detallado.
- **Entrenar un modelo propio en lugar de depender de OpenAI:** Desarrollar y entrenar un modelo interno desde cero, en lugar de depender de los modelos preentrenados de OpenAI. Esto permitiría un mayor control sobre el proceso de entrenamiento y la capacidad de adaptar el modelo específicamente a las necesidades de la herramienta. Para lograrlo, se podría usar Python como lenguaje principal, junto con datasets extraídos de investigaciones previas, proyectos de software existentes, y especificaciones de sistemas IoT. Esta personalización del entrenamiento mejoraría la comprensión del NLP adaptado a contextos específicos, como el desarrollo de sistemas IoT, asegurando que el modelo sea más eficiente y adecuado para las necesidades del usuario.

Estas ideas representan solo una parte de las muchas mejoras y funcionalidades que se podrían implementar en la herramienta. Existe un gran entusiasmo por continuar trabajando en esta línea de investigación, explorando cómo la IA puede potenciar las herramientas *case* y facilitar el trabajo de los desarrolladores. La integración de modelos de IA personalizados y otras innovaciones permitirán crear soluciones más eficientes y adaptadas a las necesidades del desarrollo de sistemas IoT, abriendo nuevas oportunidades para mejorar la calidad y rapidez en el diseño y construcción de software.

Bibliografía

- [1] S. Altman, G. Brockman, I. Sutskever, J. Schulman, W. Zaremba, and E. Musk, “Models - OpenAI API.” <https://platform.openai.com/docs/models>, 2024.
- [2] B. Chen, K. Chen, S. Hassani, Y. Yang, D. Amyot, L. Lessard, G. Mussbacher, M. Sabetzadeh, and D. Varro, “On the use of GPT-4 for Creating Goal Models: An Exploratory Study,” *Proceedings - 31st IEEE International Requirements Engineering Conference Workshops, REW 2023*, pp. 262–271, 2023.
- [3] J. V. Lucke and S. Frank, “A few Thoughts on the Use of ChatGPT, GPT 3.5, GPT-4 and LLMs in Parliaments: Reflecting on the results of experimenting with LLMs in the parliamentary context,” *Digital Government: Research and Practice*, vol. 5, pp. 1–12, 2024.
- [4] M. Scheschenja, S. Viniol, M. B. Bastian, J. Wessendorf, A. M. König, and A. H. Mahnken, “Feasibility of GPT-3 and GPT-4 for in-Depth Patient Education Prior to Interventional Radiological Procedures: A Comparative Analysis,” *CardioVascular and Interventional Radiology*, vol. 47, pp. 245–250, 2024.
- [5] Z. Ságodi, G. Antal, B. Bogenfürst, M. Isztin, P. Hegedűs, and R. Ferenc, “Reality check: Assessing GPT-4 in Fixing Real-World Software Vulnerabilities,” pp. 252–261, 2024.
- [6] T. Gokcimen and B. Das, “Analyzing Human and ChatGPT Responses: A Comparative Study of Transformer Models in Natural Language Processing,” Institute of Electrical and Electronics Engineers Inc., 2023.
- [7] J. Muralitharan and C. Arumugam, “Privacy BERT-LSTM: a novel NLP algorithm for sensitive information detection in textual documents,” *Neural Computing and Applications*, pp. 1–16, 2024.
- [8] X. Zhu, D. Wang, W. Pedrycz, and Z. Li, “Fuzzy Rule-Based Local Surrogate Models for Black-Box Model Explanation,” *IEEE Transactions on Fuzzy Systems*, vol. 31, pp. 2056–2064, 2023.

- [9] J. Cámara, J. Troya, L. Burgueño, and A. Vallecillo, “On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML,” *Software and Systems Modeling*, vol. 22, pp. 781–793, 2023.
- [10] D. Bajaj, U. Bharti, I. Gupta, P. Gupta, and A. Yadav, “GTMicro—microservice identification approach based on deep NLP transformer model for greenfield developments,” *International Journal of Information Technology*, vol. 16, pp. 2751–2761, 2024.
- [11] M. Arif, C. W. Mohammad, and M. Sadiq, “UML and NFR-framework based method for the analysis of the requirements of an information system,” *International Journal of Information Technology (Singapore)*, vol. 15, pp. 411–422, 2023.
- [12] Q. Zhang, C. Fang, W. Sun, Y. Liu, T. He, X. Hao, and Z. Chen, “APPT: Boosting Automated Patch Correctness Prediction via Fine-Tuning Pre-Trained Models,” *IEEE Transactions on Software Engineering*, vol. 50, pp. 474–494, 2024.
- [13] J. Li, T. Tang, W. X. Zhao, J. Y. Nie, and J. R. Wen, “Pre-Trained Language Models for Text Generation: A Survey,” *ACM Computing Surveys*, vol. 56, p. 230, 2024.
- [14] D. Cheng, D. Patel, L. Pang, S. Mehta, K. Xie, E. H. Chi, W. Liu, N. Chawla, and J. Bailey, “Foundations and Applications in Large-scale AI Models: Pre-training, Fine-tuning, and Prompt-based Learning,” *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 5853–5854, 2023.
- [15] J. Liu, C. Sha, and X. Peng, “An Empirical Study of Parameter-Efficient Fine-Tuning Methods for Pre-Trained Code Models,” *Proceedings - 2023 38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023*, pp. 397–408, 2023.
- [16] Y. Su, X. Han, Y. Lin, Z. Zhang, Z. Liu, P. Li, J. Zhou, and M. Sun, “CSS-LM: A contrastive framework for semi-supervised fine-tuning of pre-trained language models,” *IEEE/ACM Transactions on Audio Speech and Language Processing*, vol. 29, pp. 2930–2941, 2021.
- [17] D. H. Lee and B. Jang, “Enhancing Machine-Generated Text Detection: Adversarial Fine-Tuning of Pre-Trained Language Models,” *IEEE Access*, vol. 12, pp. 65333–65340, 2024.
- [18] J. Li, A. Sangalay, C. Cheng, Y. Tian, and J. Yang, “Fine tuning large language model for secure code generation,” *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*, pp. 86–90, 2024.

- [19] A. Liesenfeld, A. L. Nl, A. Lopez, A. L. Nl, and M. Dingemanse, "Opening up ChatGPT: Tracking openness, transparency, and accountability in instruction-tuned text generators," *Proceedings of the 5th International Conference on Conversational User Interfaces, CUI 2023*, 2023.
- [20] L. Zhang and Y. Hu, "A fine-tuning approach research of pre-trained model with two stages," *Proceedings of 2021 IEEE International Conference on Power Electronics, Computer Applications, ICPECA 2021*, pp. 905–908, 2021.
- [21] M. Eiglsperger, M. Kaufmann, and M. Siebenhaller, "A Topology-Shape-Metrics Approach for the Automatic Layout of UML Class Diagrams," *Proceedings of ACM Symposium on Software Visualization*, pp. 189–198, 2003.
- [22] D. Suarez, H. Posadas, and V. Fernandez, "UML-Based Design Flow for Systems with Neural Networks," *2023 38th Conference on Design of Circuits and Integrated Systems, DCIS 2023*, 2023.
- [23] B. Gosala, S. R. Chowdhuri, J. Singh, M. Gupta, and A. Mishra, "Automatic Classification of UML Class Diagrams Using Deep Learning Technique: Convolutional Neural Network," *Applied Sciences 2021, Vol. 11, Page 4267*, vol. 11, p. 4267, 2021.
- [24] M. H. Chu and D. H. Dang, "Automatic Extraction of Analysis Class Diagrams from Use Cases," *Proceedings - 2020 12th International Conference on Knowledge and Systems Engineering, KSE 2020*, pp. 109–114, 2020.
- [25] A. Koenig, B. Allaert, and E. Renaux, "NEURAL-UML: Intelligent Recognition System of Structural Elements in UML Class Diagram," *Proceedings - 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2023*, pp. 605–613, 2023.
- [26] A. Milanova, "Precise identification of composition relationships for UML class diagrams," *20th IEEE/ACM International Conference on Automated Software Engineering, ASE 2005*, pp. 76–85, 2005.
- [27] S. Yang and H. Sahraoui, "Towards automatically extracting UML class diagrams from natural language specifications," *Proceedings - ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems, MODELS 2022: Companion Proceedings*, pp. 396–403, 2022.
- [28] F. Wang, "UML diagram classification model based on convolution neural network," *Optik*, p. 170463, 2022.

- [29] E. A. Abdelnabi, A. M. Maatuk, and M. Hagal, “Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques,” *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering, MI-STA 2021 - Proceedings*, pp. 288–293, 2021.
- [30] O. Petrovska, L. Clift, and F. Moller, “Generative AI in Software Development Education: Insights from a Degree Apprenticeship Programme,” *ACM International Conference Proceeding Series*, 2023.
- [31] B. Lemon, A. Riesbeck, T. Menzies, J. Price, J. D’Alessandro, R. Carlsson, T. Prifiti, F. Peters, H. Lu, and D. Port, “Applications of simulation and AI search: Assessing the relative merits of agile vs traditional software development,” *ASE2009 - 24th IEEE/ACM International Conference on Automated Software Engineering*, pp. 580–584, 2009.
- [32] A. Nasari, L. Zhai, Z. He, H. Le, J. Tao, S. Cui, D. Chakravorty, L. M. Perez, and H. Liu, “Porting AI/ML Models to Intelligence Processing Units (IPUs),” *PEARC 2023 - Computing for the common good: Practice and Experience in Advanced Research Computing*, pp. 231–236, 2023.
- [33] H. Sofian, N. A. M. Yunus, and R. Ahmad, “Systematic Mapping: Artificial Intelligence Techniques in Software Engineering,” *IEEE Access*, vol. 10, pp. 51021–51040, 2022.
- [34] C. Tao, J. Gao, and T. Wang, “Testing and Quality Validation for AI Software-Perspectives, Issues, and Practices,” *IEEE Access*, vol. 7, pp. 120164–120175, 2019.
- [35] J. Ai, W. Su, S. Zhang, and Y. Yang, “A Software Network Model for Software Structure and Faults Distribution Analysis,” *IEEE Transactions on Reliability*, vol. 68, pp. 844–858, 2019.
- [36] R. G. Cooper, “The Artificial Intelligence Revolution in New-Product Development,” *IEEE Engineering Management Review*, vol. 52, pp. 195–211, 2024.
- [37] M. Deinum and K. Serneels, “Pro spring MVC: con flujo web,” *Volumen 9781430241560, Páginas 1 - 565*, vol. 9781430241560, pp. 1–565, 2012.

