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, 4 de julio de 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: palabra clave 1, palabra clave 2, ..., palabra clave N

Resumen

Con el fin de ayudar a los ingenieros de software que desarrollen sistemas IoT, automatizando parte de sus tareas, para que puedan realizar su trabajo más eficientemente y minimizar posibles errores humanos, se propone extender la herramienta TDDT4IoTS (Test-Driven Development Tool for IoT-based Systems), dotándola de una mayor inteligencia y de nuevas capacidades para mejorar significativamente la eficiencia del proceso de desarrollo de sistemas IoT, facilitando su diseño e implementación y acortando tiempos. Para ello, habrá que analizar las técnicas y tecnologías existentes de Inteligencia Artificial (IA) que podrían ser más adecuadas para extender y mejorar esta herramienta, con el fin de integrarlas para el análisis de información textual referente a la especificación de un sistema IoT. En la práctica, se partirá de las descripciones del sistema a desarrollar especificadas por los desarrolladores en forma de casos de uso extendidos, que son las entradas proporcionadas a la herramienta TDDT4IoTS. Con el análisis mediante técnicas de IA, se 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.

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

Dúval Carvajal Suárez

Keywords: keyword 1, keyword 2, ..., keyword N

Abstract

In order to assist software engineers developing IoT systems by automating part of their tasks so they can work more efficiently and minimize potential human errors, it is proposed to extend the TDDT4IoTS (Test-Driven Development Tool for IoT-based Systems) tool, equipping it with greater intelligence and new capabilities to significantly improve the efficiency of the IoT systems development process, facilitating its design and implementation, and shortening time frames. To achieve this, existing Artificial Intelligence (AI) techniques and technologies that could be most suitable for extending and improving this tool will need to be analyzed in order to integrate them for the analysis of textual information regarding the specification of an IoT system. In practice, the starting point will be the system descriptions to be developed specified by the developers in the form of extended use cases, which are the inputs provided to the TDDT4IoTS tool. Through analysis using AI techniques, the key elements (possible classes, attributes, relationships, etc.) that should be considered to automatically generate a conceptual class diagram will be identified. This can then be refined to obtain a class diagram for the solution's design domain.

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 4 de julio 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 4 de julio de 2024.

Los directores:

Miguel J. Hornos Barranco **Carlos Rodríguez Domínguez**

Agradecimientos

Poner aquí los agradecimientos.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

... Quisiera saber como están las demás secciones para redactar la introducción con las observaciones en toda la memoria

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 cómo se pueden automatizar procesos que los programadores realizamos manualmente o de memoria. Considerando el impacto global de la inteligencia artificial, 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 (Test-Driven Development Tool for IoT-based Systems) mediante la integración de técnicas y tecnologías de inteligencia artificial, con el fin de automatizar tareas en el desarrollo de sistemas IoT, mejorar la eficiencia y minimizar errores humanos.

1.2.2. Objetivos especificos

1. Análisis de IA: Investigar y evaluar las técnicas y tecnologías de inteligencia artificial más adecuadas para integrar en la herramienta TDDT4IoTS, enfocadas en el análisis de información textual referente a la especificación de sistemas IoT.

2. Automatización del Análisis: Desarrollar e implementar algoritmos 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. Generación de Diagramas: Crear un módulo dentro de TDDT4IoTS que, a partir del análisis de casos de uso, genere automáticamente diagramas de clases conceptuales, que luego puedan refinarse para obtener diagramas de clases del dominio del diseño de la solución.

1.3. Estructura del trabajo

El resto de la memoria está organizada de la siguiente manera:

Capítulo 2: está compuesto por varias secciones que abordan diferentes aspectos de los modelos de lenguaje y ajuste fino, la generación automática de UML, la integración de software con IA, y la generación automática de código y prototipos. 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 UML y la integración del software con IA, abordando temas como optimización y portabilidad de modelos, mapas sistemáticos de técnicas, y análisis de fallos en

redes de software. 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.

Capítulo 2

Estado del arte / Trabajos previos

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 procesamiento de lenguaje natural (NLP), el uso de técnicas de fine-tuning para adaptar modelos preentrenados a tareas específicas, y las metodologías para la generación de diagramas UML a partir de texto natural. También se analizan las aplicaciones de inteligencia artificial 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 inteligencia artificial en la generación automática de un diagrama de clases a partir de descripciones de casos de uso extendidos.

2.1. Modelos de Lenguaje y NLP

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, ideal para aplicaciones que necesitan respuestas rápidas y eficientes sin sacrificar la calidad [1]. Boqi Chen menciona en su estudio [2], examinan el uso de GPT-4 para la creación de modelos de objetivos 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]. Segun se muestra en [3] se 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.

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]. Zoltán et al. [4], 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

Tunahan Gökçimen y Bihter Das [5] en su investigación 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.

Se presenta un algoritmo innovador que combina las capacidades del modelo BERT y LSTM para la detección de información 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 [6].

La investigación destaca cómo la combinación de diferentes arquitecturas de modelos de lenguaje puede mejorar significativamente las tareas de procesamiento de lenguaje natural (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.

En [7], Xiubin Zhu et al. presentan un método innovador para explicar modelos de caja negra mediante el uso de modelos de sustitutos locales basados en reglas difusas. 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 inteligencia artificial. 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 procesamiento del lenguaje natural, destacando la importancia de desarrollar herramientas que permitan a los usuarios confiar y entender las decisiones automatizadas.

En los trabajos [8][9][10], se exploran diferentes aplicaciones de la inteligencia artificial generativa y herramientas de modelado en la ingeniería de software. El primer estudio [8] 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. El segundo estudio [9] propone un enfoque para la identificación de microservicios utilizando un modelo de transformador profundo de NLP en desarrollos greenfield. 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.

El tercer estudio [10] propone un método que combina UML y un marco de requisitos no funcionales (NFR) 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. Al relacionar estos estudios, 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.

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 el ajuste fino específico para tareas, el ajuste fino general, el ajuste fino específico para dominios, el ajuste fino eficiente en parámetros, el ajuste fino semi-supervisado, el ajuste fino adversarial, el ajuste fino enfocado en la seguridad, el ajuste basado en instrucciones y el ajuste fino en 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 preentrenados para realizar de manera óptima tareas particulares. En el trabajo realizo por Zoltán Ságodi et al., 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 [4].

Un enfoque similar se observa en [11] 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.

Se han revisado diversas técnicas de ajuste fino aplicadas en la generación de texto. Aquí se destaca 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. Se 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 [12].

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. Jörn Lucke y Sander Frank en su trabajo [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 procesamiento del lenguaje natural (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.

Derek Cheng et al. [13] 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. 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 mostró que el modelo podía generar modelos de objetivos con una precisión y coherencia mejoradas, facilitando el trabajo de los ingenieros de requisitos [2].

Derek Cheng que formo parte del trabajo [2] aporto en otra investigación [13] que también aborda 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. Jiaxing Liu et al. [14] 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 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 procesamiento de lenguaje natural. 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 [15].

2.2.6. Ajuste Fino Adversarial

El ajuste fino adversarial mejora la robustez del modelo mediante la exposición a ejemplos adversariales durante el entrenamiento. Dong Lee y Neackcheol Jang mencionan en su trabajo [16] 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 y errores comunes. Junjie Li en [17] 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 generado. 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. Además, este enfoque ayuda a establecer mejores prácticas de codificación en los desarrolladores, ya que el modelo sugiere soluciones que siguen estándares de seguridad, promoviendo un entorno de desarrollo más seguro y robusto.

2.2.8. Ajuste Basado en Instrucciones

El ajuste basado en instrucciones 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 [18].

2.2.9. Ajuste Fino en Dos Etapas

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 [19].

Como conclusión de esta sección 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.

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 procesamiento de lenguaje natural (NLP). Un enfoque innovador basado en métricas de forma-topología 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 [20].

El uso de redes neuronales convolucionales (CNNs) ha sido explorado para la clasificación automática de diagramas de clases UML, mejorando significativamente la precisión y eficiencia. Esta técnica es aplicable en la automatización del reconocimiento y análisis de diagramas, reduciendo la necesidad de intervención manual y aumentando la escalabilidad en proyectos de software a gran escala [21]. Asimismo, la extracción automática de diagramas de clases de análisis a partir de descripciones de casos de uso mediante técnicas de NLP facilita la transición de los requisitos a los modelos de diseño, asegurando una representación precisa y completa de las funcionalidades del sistema [22].

El sistema NEURAL-UML, 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 [23]. La identificación precisa de relaciones de composición en diagramas UML es otro avance significativo, contribuyendo a la fiabilidad y precisión de los diagramas utilizados en el diseño detallado de software y arquitectura [24].

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. 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 [25]. Además, un modelo de clasificación de diagramas UML basado en CNNs 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 [26].

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. 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 [27]. Por último, una encuesta de enfoques y técnicas para generar diagramas de clases UML a partir de requisitos en lenguaje natural 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 futuros esfuerzos de investigación en este campo [28].

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 [29].

2.4. AI en Ingeniería de Software

La integración de la inteligencia artificial (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 estudio de caso, se asignaron tareas de programación a estudiantes de ingeniería de software, 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 len-

guajes 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 [30].

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 detallado 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. 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 [31].

La combinación de simulación y algoritmos de búsqueda AI 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 AI/ML

Se abordan los desafíos y técnicas para portar modelos de inteligencia artificial y aprendizaje automático a unidades de procesamiento de inteligencia (IPUs). La optimización de modelos AI/ML para hardware especializado, como las IPUs, 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 IPUs, que son cruciales para aplicaciones que requieren procesamiento intensivo de datos y en tiempo real [32].

La importancia de ajustar y optimizar los modelos AI/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 IPUs. Este enfoque es crucial para mantener la competitividad en un campo en rápida evolución como el de la inteligencia artificial y el aprendizaje automático [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 requería 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 inteligencia artificial 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 operativas. 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 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 inteligencia artificial 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].

Capítulo 3

Desarrollo del TFM

En esta parte describirás el trabajo desarrollado, y utilizarás las secciones que creas conveniente para explicarlo.

Bibliografía

[1] I. S. J. S. y. W. Z. E. M. Sam Altman, Greg Brockman, "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 parliamentarian context," *Digital Government: Research and Practice*, 1 2024.

[4] 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, 6 2024.

[5] T. Gokcimen and B. Das, "Analyzing human and chatgpt responses: A comparative study of transformer models in natural language processing," *4th International Informatics and Software Engineering Conference - Symposium Program, IISEC 2023*, 2023.

[6] 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, 5 2024.

[7] 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, 6 2023.

[8] 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, 6 2023.

[9] 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 (Singapore)*, vol. 16, pp. 2751–2761, 6 2024.

[10] 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, 1 2023.

[11] 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, 3 2024.

[12] 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, 4 2024.

[13] 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, 8 2023.

[14] 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.

[15] 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.

[16] 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.

[17] J. Li, A. Sangalay, C. Cheng, Y. Tian, and J. Yang, "Fine tuning large language model for secure code generation," pp. 86–90, 4 2024.

[18] 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*, 7 2023.

[19] L. Zhang and Y. Hu, "A fine-tuning approach research of pre-trained model with two stage," *Proceedings of 2021 IEEE International Conference on Power Electronics, Computer Applications, ICPECA 2021*, pp. 905–908, 1 2021.

[20] 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.

[21] A. R. Viesca and M. A. Lail, "Automated mitigation of frame problem in uml class diagram verification," *Proceedings - 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2023*, pp. 841–850, 2023.

[22] 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, 5 2021.

[23] 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, 11 2020.

[24] 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.

[25] 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.

[26] 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, 10 2022.

[27] F. Wang, "Uml diagram classification model based on convolution neural network," *Optik*, p. 170463, 12 2022.

[28] F. Huber and G. Hagel, "Work-in-progress: Converting textual software engineering class diagram exercises to uml models," *IEEE Global Engineering Education Conference, EDUCON*, vol. 2022-March, pp. 1–3, 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, 5 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*, 9 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, 7 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, 5 2019.

[36] R. G. Cooper, "The artificial intelligence revolution in new-product development," *IEEE Engineering Management Review*, vol. 52, pp. 195–211, 2 2024.