Revista Innovación y Software
Vol. 6, No. 2, Mes Septiembre - Febrero, 2025
ISSN: 2708-0935
Pág. 171-191
https://revistas.ulasalle.edu.pe/innosoft

Tipo de artículo: Artículos de revisión
Temática: Ingeniería de software
Recibido: 29/4/2025 | Aceptado: 13/6/2025 | Publicado: 30/9/2025

Identificadores persistentes:
DOI: 10.48168/innosoft.s24.a287
ARK: ark:/42411/s24.a287
PURL: 42411/s24.a287

Identificación y Medición de Deuda Técnica Autoadmitida en Herramientas de Aprendizaje Profundo: Una Revisión Sistemática

Identification and Measurement of Self-Technical Debt in Deep Learning Frameworks: A Systematic Review

Elizabeth Cuatecontzi Cuahutle[1][0009-0008-0531-0354]*, **María Guadalupe Medina Barrera**[2][0000-0003-3074-0029], **Raúl Cortes Maldonado**[3][0000-0001-8463-1325], **Carlos Bueno Avendaño**[4][0000-0003-3203-5884]

[1]Tecnológico Nacional de México – Instituto Tecnológico de Apizaco. Tlaxcala-México. Código Postal: 90491. elizabeth.cc@apizaco.tecnm.mx

[2]Tecnológico Nacional de México – Instituto Tecnológico de Apizaco. Tlaxcala-México. Código Postal: 90491. guadalupe.mb@apizaco.tecnm.mx

[3]Tecnológico Nacional de México – Instituto Tecnológico de Apizaco. Tlaxcala-México. Código Postal: 90491. raul.cm@apizaco.tecnm.mx

[4]Tecnológico Nacional de México – Instituto Tecnológico de Apizaco. Tlaxcala-México. Código Postal: 90491. carlos.ba@apizaco.tecnm.mx

*Autor para correspondencia: elizabeth.cc@apizaco.tecnm.mx

Resumen
La Deuda Técnica en el desarrollo de Software se refiere a las consecuencias de decisiones que priorizan soluciones rápidas sobre soluciones óptimas. Este concepto, introducido por Ward Cunningham en 1992, ha sido ampliamente estudiado para mejorar la calidad del software. En el contexto del aprendizaje profundo, la DT también está presente debido al uso de herramientas que, aunque facilitan la creación de modelos, pueden generar DT y afectar su rendimiento. Con un proceso de tres fases, este trabajo presenta una revisión sistemática de la literatura con el objetivo de identificar los tipos de DT presentes en herramientas de aprendizaje profundo, así como las técnicas empleadas para su identificación y medición. Los estudios revisados muestran que la DT puede aparecer en diversas fases del desarrollo, como el diseño, definición de requisitos, pruebas, documentación, código, algoritmos y compatibilidad. Además, se identifican aspectos adicionales afectados, tales como los datos, los modelos, el conocimiento y la infraestructura.Para identificar la DT, se han utilizado enfoques como el análisis de comentarios en código estático, *pull requests* y *commits*, aplicando técnicas manuales, minería de texto, redes neuronales y algoritmos de procesamiento de lenguaje natural. En cuanto a su medición, predominan los métodos estadísticos. Los hallazgos de esta revisión permiten comprender mejor cómo la DT impacta las herramientas de aprendizaje profundo y ofrecen una base para orientar investigaciones futuras sobre su gestión y mitigación en el desarrollo de sistemas inteligentes.

Palabras claves: Aprendizaje profundo, deuda técnica, herramientas de aprendizaje profundo, medición de la deuda técnica, tipos de deuda técnica.

Revista Innovación y Software
Vol. 6, No. 2, Mes Septiembre - Febrero, 2025
ISSN: 2708-0935
Pág. 171-191
https://revistas.ulasalle.edu.pe/innosoft

Abstract
Technical Debt in software development refers to the consequences of decisions prioritizing quick solutions over optimal ones. This concept, introduced by Ward Cunningham in 1992, has been widely studied to improve software quality. In the context of deep learning, Technical Debt is also present due to the use of tools that, while facilitating model creation, may generate debt and negatively impact performance. Through a three-phase process, this study presents a systematic literature review to identify the types of Technical Debt found in deep learning tools and the techniques used for its identification and measurement. The reviewed studies show that Technical Debt can arise in various development phases, such as design, requirements definition, testing, documentation, source code, algorithms, and compatibility. Other affected aspects include data, models, knowledge, and infrastructure. Several approaches have been used to identify technical debt, such as analyzing comments in static code, pull requests, and commits, applying manual techniques, text mining, neural networks, and natural language processing algorithms. In terms of measurement, statistical methods are predominantly used. The findings of this review provide a better understanding of how Technical Debt impacts deep learning tools and offer a foundation for guiding future research on its management and mitigation in the development of systems within intelligent environments.

Keywords: *Deep learning, deep learning tools, technical debt, types of technical debt, technical debt measurement*

---

1. Introducción

La aplicación de tecnologías como la visión por computadora, el procesamiento de lenguaje natural y el reconocimiento de voz ha permitido que la Inteligencia Artificial (Artificial Intelligence, AI por sus siglas en inglés) revolucione y beneficie a múltiples sectores de la sociedad. Para lograrlo, la AI utiliza herramientas como el aprendizaje automático (Machine Learning, ML) y aprendizaje profundo (Deep Learning, DL).

El ML consiste en extraer conocimiento de los datos mediante algoritmos que permiten a la AI imitar la forma en que los humanos aprenden [1]. En esta disciplina, el DL es una subárea basada en redes neuronales, cuya estructura y funcionamiento están inspirados en el cerebro humano [2]. La arquitectura de los modelos de DL está compuesta por unidades interconectadas, llamadas neuronas, organizadas en una capa de entrada, una o más capas ocultas y una capa de salida. Las redes de DL emplean múltiples capas ocultas profundamente anidadas, lo que les permite realizar operaciones avanzadas, como convoluciones. Gracias a esta estructura, las redes DL pueden procesar datos de entrada sin requerir un preprocesamiento extenso, generando automáticamente representaciones útiles para la tarea de aprendizaje [3]. De acuerdo con [4] esta capacidad ha convertido al DL en una herramienta poderosa para resolver problemas complejos, lo que ha impulsado su popularidad en la comunidad científica para el desarrollo de aplicaciones.

Durante la creación de este tipo de aplicaciones, los desarrolladores suelen apoyarse en herramientas y marcos de trabajo (*frameworks)* de código abierto que facilitan su implementación, una evalución de herramientas populares de DL son presentadas en [2] y [5]. Sin embargo, debido a la falta de conocimiento, la presión del

tiempo, el contexto complejo, entre otros factores, surgen incertidumbres durante su desarrollo lo que lleva realizar suposiciones sobre su uso, lo que podrá producir resultados no esperados [6].

Al igual que en los sistemas tradicionales, Suculley et al. en [7] señalan que, en las herramientas y sistemas de DL, factores como la presión del tiempo incrementan el riesgo de priorizar soluciones rápidas sobre soluciones óptimas. Esta tendencia, que antepone la velocidad de desarrollo a los costos y estándares de calidad conduce a la acumulación de Deuda Técnica (DT). La DT es una metáfora introducida por Ward Cunningham en 1992 para describir los costos a largo plazo asociados con decisiones aceleradas durante el desarrollo de software. La DT tiene una variedad de forma y puede afectar múltiples cualidades del software que incluye per no limita a la legibilidad, el desempeño, y estructura [8]. En el contexto de ML, en [7] se abordan los desafíos y costos ocultos relacionados con la construcción y mantenimiento de sistemas de aprendizaje automático, argumentan que estos sistemas pueden acumular DT, ya que enfrentan problemas de mantenimiento típicos del código tradicional y desafíos específicos del ML, como la dependencia de datos, la configuración de los datos y la representación del modelo.

Con una adaptación del proceso de tres fases propuesto por Brereton et al. [9] y Kitchenham et al. [10], este trabajo presenta los resultados de una Revisión Sistemática de la Literatura (RSL) sobre las prácticas actuales en la identificación y medición de la DT en herramientas de DL. El objetivo es identificar los tipos de DT más comunes, los procesos, técnicas y herramientas utilizadas para su detección, así como los criterios de medición aplicables.

Los resultados de esta revisión motivan futuras investigaciones sobre el impacto de la DT en herramientas DL para desarrollar sistemas DL aplicables en la solución de problemas en diversos sectores de la sociedad.

Este documento se organiza en cuatro secciones. En la sección dos, se expone la motivación que dio origen a esta RSL. La sección tres detalla el proceso metodológico de la revisión. En la sección cuatro, se discuten los resultados obtenidos y se propone una metodología para un estudio más amplio sobre la DT en herramientas de DL y su impacto en aplicaciones de DL.

2. Motivación

Las necesidades tecnológicas en el desarrollo de las ciencias y en áreas como la medicina, la educación, la manufactura y la ciberseguridad son cada vez más diversas, lo que hace que la incorporación de tecnologías emergentes como la AI, el ML y el DL sea de gran utilidad.

El DL ha contribuido con soluciones innovadoras en áreas como el procesamiento de lenguaje natural, el

reconocimiento del habla y la visión por computadora. En DL se han desarrollado métodos basados en datos que emplean múltiples capas de unidades interconectadas (neuronas) para aprender patrones y representaciones a partir de datos. Una revisión sobre modelos de DL y sus aplicaciones es presentada en [11], mientras que [4] presentan un analisis del progreso y los desafíos actuales del DL.

La literatura tambien reporta un gran número de aplicaciones de sistemas basados en DL. Por ejemplo, en [2], [12] y [13] presentan una revisión del uso del DL en sistemas de salud, destacando aplicaciones clave como el diagnóstico por imágenes, el procesamiento de señales y el análisis de texto médico. En sus trabajo, señalan desafíos como la falta de datos, la calidad de los resultados, problemas éticos, la interpretabilidad de los modelos y la escalabilidad. Además, abordan cuestiones como la integración de modelos en entornos clínicos reales y la interoperabilidad tecnológica.

Por otro lado, para apoyar el desarrollo de sistemas de DL, se han diseñado y utilizado herramientas que integran bibliotecas de funciones y algoritmos de DL, que se aplican a una amplia variedad de contextos científicos y tecnológicos. Una evaluación integral de herramientas populares de DL (TensorFlow, MXNet, PyTorch, Theano, Chainer y Keras) en arquitecturas como CNN, R-CNN y LSTM es presentada en [2]. Su estudio evaluó la precision, el tiempo de entrenamiento y consumo de recursos de cada una. Así mismo, en [5] se ofrece un análisis de las herramientas y marcos de trabajo más recientes que facilitan la creación, implementación y expansión de modelos de DL, evalúan las capacidades de marcos como Tensor Flow, Pytorch y Keras. Por su parte, en [14] evaluaron el rendimiento de modelos de aprendizaje automático y herramientas de DL como PyTorch, Keras, TensorFlow, Caffe y Theano. Destacan que, aunque el uso de estas herramientas está ampliamente difundido, es necesario evaluar su calidad, los desarrolladores de sistemas DL pueden asumir supuestos sobre el uso y desempeño de estas herramientas, lo que podría derivar en vulnerabilidades, fallos, inconsistencias o un incremento en los costos en las aplicaciones de DL.

Uno de las primeras investigaciones sobre la caracterización de la DT en herramientas de DL es realizada y presentada en [15]. Sin embargo, el creciente desarrollo de sistemas basados en DL y el uso extendido de estas herramientas motivan la necesidad de actualizar y profundizar estos hallazgos. Por su parte, Bathia et al. [16] afirma que a pesar de los esfuerzos recientes para comprender la introducción y eliminación de SATD en herramientas de aprendizaje profundo, todavía no se sabe mucho sobre la difusión y evolución de la deuda técnica en los sistemas basados en ML utilizando tales herramientas.

De igual relevancia y en relación con la identificación y mitigación de la DT es de suma importancia estudiar la aplicación de prácticas de ingeniería de software para el ML. Nascimento et al, en [17] analizan como la ingeniería de software ha sido aplicada en el desarrollo de la AI y el ML, por su parte Serban et al. [18] destacan la importancia de las prácticas de ingeniería de software y publican una guía práctica para el desarrollo de

software con componentes de aprendizaje automático.

3. Metodología

Con una adaptación del proceso de tres fases propuesto por Breton et al. [9] y Kitchenman et al. [10], se realizó una RSL sobre la Identificación y Medición de la Deuda Técnica en Herramientas de Aprendizaje Profundo. La figura 1 muestra la metodología que se siguió en este trabajo.

Figura 1. Proceso de RSL sistemática de la literatura. Adaptación de la propuesta de [9] y [10]

Con esta propuesta se realizó la búsqueda de artículos científicos en bases de datos académicas como IEEE Xplore, ACM Digital Library, Science Direct, Springer Link, Google Scholar y Arxiv, para ello se inició definiendo la cadena de búsqueda utilizando palabras clave relacionadas con los términos de "deuda técnica", "herramientas de aprendizaje profundo", "categorías o tipos de deuda técnica" y "medición de deuda técnica". Los criterios de inclusión se centraron en estudios que abordaran explícitamente la caracterización, identificación y medición de la DT en herramientas de aprendizaje profundo, finalmente se obtiene una síntesis con los resultados de esta Revisión.

Fase 1. Planeación de la RSL.

Preguntas de investigación. Se pretende identificar trabajos que estudian la presencia de DT en herramientas de DL, los tipos que pueden existir, características, etapa de desarrollo en que ocurren, y que herramientas de aprendizaje profundo se presentan y con qué frecuencia. De igual forma se tiene interés en identificar procesos, técnicas, herramientas y recursos para identificarla y medirla.

Así las preguntas que guían esta RSL son:

Pregunta 1. ¿Cuáles son los tipos de deuda técnica de mayor presencia en herramientas de aprendizaje profundo?

Pregunta 2. ¿Qué proceso y herramientas se utilizan para identificar y medir la DT en DL a través de un estudio descriptivo?

Protocolo de búsqueda. Para dar respuesta a estas preguntas se forma una cadena de búsqueda con las palabras clave: Deuda Técnica, herramientas de aprendizaje profundo y criterios de medición.

("Technical Debt.ᴼR "TechDebt.ᴬND charac) AND ("Deep Learning Frameworks.ᴼR "DL Frameworks") AND (Measurement)*

Esta se aplica en los motores de búsqueda de las principales Bibliotecas Digitales científicas como la IEEE Xplore, ACM Digital Library, Springer Link, Science Direct, Google Académico y ArXiv.

Criterios de inclusión y exclusión. En la RSL se trata de identificar trabajos relevantes sobre el estado actual del estudio de la DT en herramientas de DL. Por lo que se consideran reportes de conferencias, congresos y artículos de los últimos cinco años, además se busca que los trabajos sean resultado de investigaciones primarias realizadas por profesionales en centros de investigación y/o universidades. Especialmente trabajos que documenten aspectos de presencia DT, y su categorización de tipos de DT presentes en herramientas de DL, o bien que informen de herramientas o técnicas que permitan identificar y medir su presencia e impacto. Se considera publicaciones anteriores a este periodo en el caso de trabajos cuya aportación ha tenido un impacto relevante en el tema de DT y herramientas de DL. Los trabajos que no se consideran en esta RSL son aquellos que no sean primarios o trabajos informales como blogs de autores o instituciones no reconocidos, artículos no sujetos a revisión por pares o reportes del desarrollo de sistemas de DL.

Fase 2. Ejecución de la RSL.

Selección de los estudios relevantes. En cada Biblioteca Digital se aplicó la cadena de búsqueda definida en el protocolo de la RSL, como resultado de esta búsqueda se obtuvieron 244 documentos relacionados, cabe señalar que se realizaron ajustes para cumplir con la sintaxis del motor de búsqueda. A continuación, se realiza una revisión general en los metadatos de los artículos encontrados y se seleccionan solo aquellos que cumplen con los criterios de inclusión definidos.

Registro de la extracción de datos del documento investigado. Con la información de los trabajos seleccionados,

se realiza una base de datos simple con los siguientes campos como elementos: Título del artículo; año, autor(s); biblioteca digital; objetivo(s) de la investigación, metodología, conceptos relevantes utilizados para describir el dominio del tema; resultados y trabajos futuros propuestos.

Sintetizar la información. Para una mejor comprensión de la relación de los trabajos seleccionados y su impacto con el tema de interés, se analizan y se crea una síntesis, identificando la información que da respuesta a las preguntas de investigación que se plantearon para esta RSL y también información adicional relevante que aporte al tema.

Fase 3. Documentar la RSL.

Escribir el informe. Se presenta el informe de la RSL de la literatura realizada con las preguntas planteadas.

Pregunta 1. ¿Cuáles son los tipos de deuda técnica de mayor presencia en las herramientas de aprendizaje profundo?

Para identificar los tipos de DT en software, [19] analizarón comentarios en el código fuente de proyectos de código abierto alojados en repositorios como GitHub. Estos comentarios introducidos deliberadamente por los propios desarrolladores hacen referencia a observaciones sobre tareas pendientes, errores o posibles fallas que ocurren en distintas fases del desarrollo como la codificación o las pruebas. Generalmente, estos problemas surgen de la adopción de soluciones rápidas o temporales, lo que representa DT auto admitida. Por su parte, [20] clasificaron y cuantificaron diferentes tipos de DT auto admitida mediante la extracción y análisis de comentarios en el código. Identificaron cuatro tipos principales: deuda de diseño, deuda de defectos, deuda de documentación y deuda de pruebas.

De forma similar, [15] exploraron la presencia de DT en herramientas de DL de código abierto, para ello, extrajeron comentarios realizados por los desarrolladores en diferentes etapas, incluyendo problemas de implementación, pull requests y commits. Este enfoque les permitió identificar la existencia de DT auto admitida en herramientas DL. En este trabajo, los investigadores analizaron herramientas populares como TensorFlow, Keras, PyTorch, Caffe, MXNet, CNTK y DL4J. En su estudio, encontraron una cantidad significativa de DT auto admitida, incluyendo diseño subóptimo, defectos sin resolver, documentación incompleta, deficiencias en las pruebas e implementación incompleta de métodos requeridos. Además, identificaron DT de algoritmo, refiriéndose a ella como las implementaciones subóptimas de la lógica algorítmica de los modelos de DL y módulos de redes neuronales. También detectaron DT de compatibilidad, asociada con dependencias inmaduras entre proyectos, lo que obliga a utilizar soluciones temporales para garantizar la funcionalidad. Posteriormente, [21] realizaron un estudio sobre la introducción y eliminación de diferentes tipos de DT en herramientas de DL.

En el trabajo realizado por [22], centraron su investigación en la refactorización de código, haciendo énfasis en la relación entre el diseño y la fase de pruebas para mejorar el desempeño y alcanzar el requerimiento. Por su parte, en [6] destacaron la necesidad de abordar la DT auto admitida en diversas fases del desarrollo, incluyendo requisitos, diseño, codificación, algoritmos, pruebas y documentación. También en [8] reconocieron la presencia de DT en las fases de requerimientos, diseño, pruebas y documentación.

Así mismo, [16] estudiaron la DT auto admitida e identifican DT de código, de diseño, de documentación de defectos, de pruebas y de requerimientos, las cuales afirma ocurren tanto en software tradicional y de configuración que es especifica de sistemas ML. La Tabla 1 resume los tipos de DT auto admitida en herramientas de DL, identificados en distintas fases del desarrollo y reportados por diversos investigadores.

En la realización de esta RSL también se identificarón trabajos que analizan tipos específicos de DT en ML y DL. Por ejemplo, [22] examinan la relación entre la deuda técnica de código y la refactorización. Proponen una taxonomía jerárquica de refactorizaciones en sistemas de aprendizaje automático, considerando aspectos como la reorganización del código, mejoramiento del desempeño, migración de código, eliminación de código duplicado, mejoras en la seguridad, problemas de configuración, comprensión del código y reutilización del código, entre otros. Los autores ofrecen recomendaciones y mejores prácticas para desarrollar sistemas de DL, para minimizar la acumulación de DT.

amssymb

Tabla 1. Tipos de DT auto admitida en herramientas de DL.

Estudio	Diseño	Requisitos	Documentación	Pruebas	Defectos	Código	Algoritmo	Compatibilidad
[15]	✓	✓	✓	✓	✓		✓	✓
[23]	✓			✓		✓		
[6]	✓	✓	✓	✓		✓	✓	
[21]	✓	✓	✓	✓	✓		✓	✓
[8]	✓	✓	✓	✓	✓	✓		
[16]	✓	✓	✓	✓	✓	✓		

Tabla 2. Problemáticas identificadas en herramientas y sistemas DL, que inducen DT

[6]	[8]	[24]	[25]
Problemas de: Configuración del modelo y los datos. Definición del Tensor y variable Diseño Varias.	Problemas de: Dependencia de datos Dependencia de código Escalabilidad Confiabilidad Desempeño	Problemas de: Infraestructura. Hardware APIS para la integración. Ciclo de vida del desarrollo de DL Dato Modelo Entrenamiento Inferencia Flujo de trabajo	Categorías de problemas relacionados con: El procesamiento de datos. Arquitectura y configuración del modelo Entrenamiento del modelo Resultados de la predicción Inferencia de los resultados. Entorno de ejecución Visualización de los datos.

También han surgido investigaciones que profundizan en el estudio de la DT en herramientas de DL, considerando la naturaleza específica de estos sistemas. La tabla 2 muestra los estudios relacionados, mismos que se describen de forma breve.

Por ejemplo, [8] estudian la DT auto admitida en ML, incluyendo a DL, y reconocen su presencia en diversas fases del desarrollo, como la definición de requerimientos, las prácticas de codificación, la implementación, la deficiencia de pruebas, la falta de buenas prácticas de diseño y la documentación inadecuada. Proponen una taxonomía para el estudio de la DT auto admitida, organizada en nueve grupos: dependencia de datos, dependencia de código, concientización, modularidad, configuración, deuda de escalabilidad, confiabilidad, desempeño y eliminación de código duplicado. Cada grupo incluye subtipos que, en conjunto, forman lo que los autores denominan las 23 sombras de la DT auto admitida en software de ML.

Otros afirman que durante el desarrollo de sistemas DL, surgen hipótesis relacionadas con las herramientas de DL, [6] sostiene que las suposiciones son conocimientos en el desarrollo de software que se dan por hecho o se aceptan como verdaderos sin evidencia, la falta de conciencia sobre estas suposiciones puede provocar problemas críticos como vulnerabilidades, fallos, inconsistencias y aumento de costos. Estas suposiciones se clasifican en categorías como configuración y contexto, y variable, diseño y miscelánea, esta última abarca aspectos como conjuntos de datos, pruebas, errores y fallos, usuarios, costo-beneficio y rendimiento. Enss [14] destacan que los *stakeholders* (partes interesadas) en el desarrollo de herramientas de DL suelen hacer suposiciones relacionadas con decisiones de diseño, requisitos y DT. Estas decisiones pueden resultar no válidas, lo que provoca fallos en

los sistemas DL.

Otro estudio es el presentado por [24], quienes, tras analizar manualmente comentarios en proyectos de código abierto desarrollados con herramientas de Deep Learning como TensorFlow y PyTorch, crean una taxonomía específica de Deuda Técnica (DT) auto admitida. En su propuesta, identifican dos categorías principales relacionadas con aspectos de infraestructura y el ciclo de vida del desarrollo en Deep Learning. La categoría de infraestructura abarca aspectos vinculados al hardware necesario para ejecutar modelos de DL y a las API para la integración de herramientas en el proceso de desarrollo. También incluye la compatibilidad entre diferentes herramientas y la optimización del uso de recursos computacionales. Por otro lado, la categoría del ciclo de vida del desarrollo comprende tareas y desafíos en distintas etapas del proceso de creación de sistemas de DL. Estas etapas incluyen: definición de datos, diseño y configuración del modelo, configuración del proceso de entrenamiento, inferencia (tratamiento y análisis de los resultados generados durante la ejecución del modelo) y el diseño y optimización del flujo de trabajo completo, desde la entrada de datos hasta la salida de resultados. Finalmente, los autores concluyen que cuando estos desafíos se abordan con soluciones subóptimas, se genera una DT específica de DL, lo que puede afectar la calidad, mantenibilidad y escalabilidad de los sistemas desarrollados.

En esta RSL también se observó constante referencia a los desafíos enfrentados en el desarrollo de sistemas de DL, por lo que se incluyeron estudios relacionados con los problemas de los desarrolladores al crear y mantener estos sistemas. Por ejemplo, [25] analizan publicaciones y preguntas que los desarrolladores realizan en sitios como *Stack Overflow,* identifican siete categorías y 63 subcategorías de problemas relacionados con los marcos de DL TensorFlow, PyTorch, y Theano. Los autores encontraron que las dificultades en la implementación representan la mayoría de las preguntas formuladas por los desarrolladores (97.7 %), lo que sugiere problemas en la estabilidad de las aplicaciones construidas con estos marcos. Estos defectos reflejan DT acumulada, lo que impacta negativamente en la calidad general de los proyectos de DL. Las categorías son: 1) Problemas relacionados con el procesamiento de datos, 2) dificultades con la arquitectura y configuración del modelo, 3) desafíos encontrados durante la fase de entrenamiento de los modelos, 4) problemas que surgen al hacer predicciones con los modelos entrenados, 5) dificultades relacionados con la evaluación del rendimiento y la precisión de los modelos, 6) dificultades asociados con el entorno en el que se ejecutan los modelos, incluyendo dependencias y compatibilidad, y 7) preocupaciones sobre la visualización de datos, el rendimiento del modelo y los resultados.

Revista Innovación y Software
Vol. 6, No. 2, Mes Septiembre - Febrero, 2025
ISSN: 2708-0935
Pág. 171-191
https://revistas.ulasalle.edu.pe/innosoft

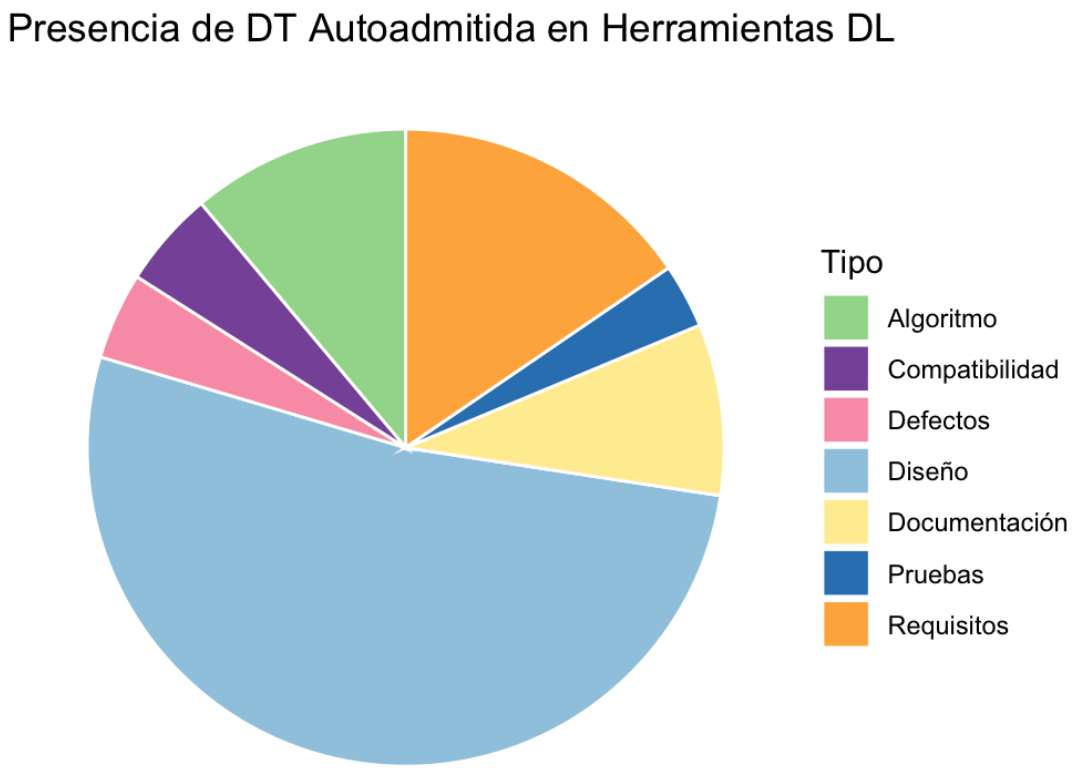

Figura 2. Distribución de la DT en DL según su tipo. Elaboración propia con datos de [15] y [21]

En relación con la distribución de la DT en las herramientas de DL, [15] analizaron comentarios en repositorios de código en GitHub e identificaron la presencia de DT auto admitida en herramientas de DL como Tensor Flow, Keras, Caffe, PyTorch, MXNet, CNTK y DL4J. La Figura 2 muestra la evolución de la DT en los periodos analizados, destacan que la deuda de diseño es la más prevalente durante el proceso de desarrollo, seguida por la deuda de requisitos y la deuda de algoritmo. Posteriormente [21] encontraron que la DT de requisitos, seguida por la DT de diseño, son las que se eliminan con mayor frecuencia. Por el contrario, la DT de documentación y la DT de pruebas son las que menos se remueven, mientras que la DT de compatibilidad y la DT de pruebas presentan los procesos de eliminación más lentos. La Tabla 3 resume la ocurrencia de los distintos tipos de DT en los marcos de DL analizados, proporcionando una visión comparativa según su frecuencia y persistencia en el tiempo.

Pregunta 2. ¿Qué procesos y herramientas se utilizan para identificar y medir la DT en DL a través de un

estudio descriptivo?

Para identificar la deuda técnica auto admitida los trabajos revisados analizan comentarios que los desarrolladores realizan en el código, en los commits o en pullrequest de las herramientas, en plataformas de preguntas y respuestas. Para medir su presencia estudios proponen el uso métricas cuantitativas basadas en el análisis del código estático, mientras que otros sugieren enfoques cualitativos, como revisiones de código, hay incluso propuestas de herramientas de minería texto y técnicas de procesamiento de lenguaje natural (PLN). La tabla 4 resume las fuentes, mecanismos y herramientas y/o técnicas que los investigadores han utilizado para identificar la DT en herramientas DL.

Tabla 3. Presencia de la deuda Técnica en herramientas de aprendizaje profundo. Puede observarse que CNTK presentó el mayor porcentaje de DT en los dos años estudiados.

Año	**2020**		**2021**	
Tipo de DT	**Mayor**	**Menor**	**Mayor**	**Menor**
Diseño	CNTK - 65.27 %	Keras - 24.07 %	CNTK - 76.33 %	Caffe 48.52 %
Requisitos	DL4J - 20.67 %	Caffe - 5.62 %	Keras - 16.75 %	CNTK - 7.04 %
Documentación	Caffe - 15.62 %	Keras - 0.00 %	Caffe - 20.00 %	Keras - 0.00 %
Pruebas	Tensorflow - 7.70 %	Keras - 0.00 %	Tensorflow - 5.12 %	Keras - 0.52 %
Defectos	MXNet - 8.47 %	Keras - 1.85 %	CNTK - 5.54 %	MXNEt - 3.01 %
Algoritmo	Keras - 31.48 %	Tensorflow - 7.09 %	Keras - 14.14 %	DL4J - 4.82 %
Compatibilidad	Keras - 35.18 %	DL4-J 0.21 %	Caffe - 10 %	DL4-J 0.0 %

En el estudio realizado por [15] extrajeron comentarios del código fuente, *commits y pullrequest* en herramientas de aprendizaje profundo como *TensorFlow, Keras, DL4J, Caffe, PyTorch, MXNet y CNTK*, accesando a repositorios alojados en *GitHub*. Para este proceso, emplearon herramientas como *srCML* y el módulo *tokenize* de la librería estándar de *Python,* que proporciona un escáner léxico para identificar comentarios en el código. Con los datos recolectados, realizaron una revisión y clasificación manual en cinco categorías: deuda de diseño,

deuda de defectos, deuda de requerimientos, deuda de documentación y deuda de pruebas. La propuesta de estas categorías fue tomada de [20]. Este proceso de análisis de comentarios incluyó varias iteraciones, evaluadas cualitativamente por varios investigadores para garantizar la coherencia en la clasificación. Para medir el grado de acuerdo entre los evaluadores, los autores utilizaron la métrica estadística del coeficiente de Cohen's kappa. El análisis del código abarcó distintos niveles de granularidad, incluyendo comentarios en métodos, clases y archivos que contenían metainformación de la herramienta. Un ejemplo de este análisis es el, por ejemplo, el *commit*:

"TODO: this option should be abstracted, if we decide to generalize this training master" - [from DL4J]

"TODO: make it public?" - [from Keras],

indica qué, si bien la implementación actual ha cumplido las funcionalidades explícitas en el requisito, el diseño del código no es óptimo.

Posteriormente, [21] extienden su estudio para identificar la frecuencia de ocurrencias y eliminación de DT auto admitida en siete herramientas de DL, sus hallazgos indican que durante el proceso de desarrollo la deuda de diseño es la que más se introduce, la deuda de requisitos es la que más se elimina, y son los propios desarrolladores quienes eliminan la DT. Para su identificación utilizan un algoritmo basado en procesamiento de lenguaje natural (PLN) para identificar automáticamente los comentarios que indican DT. La clasificación la realizaron de forma manual, en dos iteraciones utilizando el enfoque *card sorting* de Spencer (2009). En este trabajo también evalúan el desempeño de *SATD-Detector*, que es una herramienta para la clasificación de texto con PLN desarrollada por [26] para identificarla y la cual se entrenó para ello.

Por su parte [6], al analizar supuestos auto admitidos en herramientas de DL, para su identificar su distribución, clasificación e impacto, utilizan comentarios de código de nueve herramientas populares de herramientas DL alojadas de *GitHub*, utilizan herramientas de *Visual Studio y* librerías de Python para recolectar los comentarios de código relacionados, utilizaron términos relacionados con supuestos como *assumption, assumptions, assume, assuming, assummed*. Para la clasificación una revisión cualitativa por varios autores y en diferentes rondas, utilizaron y coeficiente de Cohen Kappa para medir la concordancia entre los evaluadores. Para el análisis de datos utilizaron estadística descriptiva.

En el caso del estudio presentado por [8] para el análisis de la DT auto admitida en proyectos ML escritos en Python, minan comentarios de sistemas ML en GitHub, emplearon Boa [27], un lenguaje diseñado específicamente para extraer comentarios de repositorios de software identifica cuando un comentario ha sido introducido o eliminado en las actualizaciones del código. Con una inspección manual clasifican comentarios que representan DT auto admitida y los que no. También utilizaron la herramienta de clasificación SATD-Detector [26]

entrenado con palabras clave asociadas a la categorización, y aplicaron el coeficiente Kappa de Cohen para medir el grado de acuerdo entre evaluadores. La clasificación se realizó manualmente tras definir un esquema que considera una taxonomía propuesta por [7] y nuevamente aplicaron coeficiente Cohen Kappa.

En la propuesta de [25] analizan publicaciones en sitios de comunidades de desarrolladores, en este *Stack Overflow*. Identifican publicaciones que se relacionan con problemas de implementación en herramientas de DL, en este caso con TensorFlow, PyTorch y Theano. El estudio incluyó la recolección de datos, seguida de un proceso de refinamiento y obtención de características, para la construcción de una taxonomía; realizan una clasificación y predicción utilizando modelos de Redes Neuronales.

Tabla 4. Técnicas y herramientas para identificación y análisis de DT en herramientas de DL.

Estudio	Fuente	Datos	Técnicas y/o herramientas de extracción y análisis de DT
[15]	Repositorios GitHub de herramientas DL de TensorFlow, Theano, Pytorch, Caffe, MXNet, Keras, CNTK, DL4J, y Paddle Paddle.	Extracción de comentarios en código fuente	Herramientas de extracción: srcML, Módulo Tokenize de Python. Clasificación: Manual realizada por varios autores en varias iteraciones. Análisis y validación: Coeficiente de kappa de Cohen y métricas estadísticas.
[21]		Extracción de archivos de comentarios en versiones sucesivas de las herramientas de DL	Clasificación: Manual con técnica Card Sorting, evalúan algoritmos basados en NLP y herramienta SAT-Detector
[6]	Repositorios en GitHub de TensorFlow, Theano, Pytorch, Caffe, MXNet, Keras, CNTK, DL4J, y Paddle Paddle.	Extracción y análisis de comentarios que indican deuda técnica auto admitida.	Herramientas de extracción: Visual Studio y herramientas de Tokenización de Python Clasificación: Manual por autores Análisis y validación: Revisión cualitativa, Coeficiente de kappa de Cohen y Estadística descriptiva.

Estudio	Fuente	Datos	Técnicas y/o herramientas de extracción y análisis de DT
[8]	Repositorios GitHub de sistemas DL	Extracción de comentarios, que indican deuda técnica auto admitida	Herramientas de extracción: Lenguaje BOA para minar datos. Clasificación: Manual para identificas SAT y SATDetector de la que no lo es. Clasificación manual para clasificar de acuerdo con la taxonomía. Análisis y validación: Coeficiente de kappa de Cohen
[25]	Sitio web Stack Overflow	Publicaciones relacionadas con problemas de implementación en herramientas DL de Tensorflow y Pytorcg	Extracción: Manual Clasificación: Modelos de Redes Neuronales
[14]	Herramientas DL en GitHub	Comentarios de código, pullrequest y commits relacionados con supuestos.	Clasificación y análisis con modelos no transformadores (Máquinas de Vectores de Soporte y Árboles de Decisión), el modelo ALBERT y modelos de solo codificador.

Recientemente, [14] clasifican suposiciones de decisiones de diseño requisitos y deuda técnica en el desarrollo en herramientas de DL, a partir de la colección de un conjunto de datos de comentarios de supuestos desde el punto de vista de desarrolladores y usuarios. Los datos son recolectados de fuentes como comentarios de código, *pullrequest* y *commits* alojados repositorios de las herramientas DL en GitHub. En su primer estudio analizaron y clasificaron los supuestos de forma manual; en este estudio tras la construcción de un conjunto de datos mayor y la definición de un esquema de clasificación, exploran el desempeño de cuatro modelos no transformadores (ejemplo, Máquinas de Vectores de Soporte, Árboles de Decisión), el modelo ALBERT (a *lite Bidirectional Encoder Representations from Transformers)* y tres modelos de solo codificador (ejemplo, ChatGpt, Claude y

Gemini). Observaron que el modelo ALBERT alcanza el mejor desempeño en la clasificación. ALBERT es un modelo de PLN desarrollado por Google [28].

4. Discusión

Con la revisión sistemática de la literatura realizada se encontraron estudios relevantes sobre la identificación y medición de DT en herramientas de DL. Los hallazgos permitieron identificar los tipos de DT presentes en estas herramientas y, en algunos casos, analizar su incidencia. Además, se identificaron métodos y herramientas utilizados para su detección y medición.

Los estudios muestran que la DT en el contexto de las herramientas y sistemas de DL puede ocurrir en diversas etapas de su desarrollo y también en alguno de sus componentes. Los estudios muestran que la DT suele ocurrir en las fases de diseño, requisitos, código, pruebas y documentación, determinándose como los tipos de deuda en DL. Entre estos, la DT de mayor presencia es la de diseño, la cual puede ocurrir por diversos factores como un diseño modular deficiente o rígido que dificulta la incorporación de nuevas funcionalidades o bien, su mejoramiento.

Otros estudios abordan aspectos específicos del DL, como la DT de compatibilidad, esta ocurre cuando las herramientas o bibliotecas no son fácilmente actualizables ni compatibles entre versiones o con versiones futuras. También se encuentra DT de algoritmo, que surge al implementar un algoritmo subóptimo sin previsión de necesidades futuras, o inadecuadas para redes profundas.

En las herramientas de DL también se encuentra DT que se relaciona con los datos, los modelos, la infraestructura o incluso el conocimiento del modelo y la herramienta. La DT en datos puede ser originada por sesgos, dependencia de las fuentes de obtención o falta de trazabilidad. La DT en los modelos, puede ocurrir cuando los parámetros de este no se ajustan u optimizan correctamente para el problema en cuestión. La DT en infraestructura puede deberse a la dependencia del hardware o a la integración con los sistemas de software. La DT de conocimiento puede generarse por el mismo desconocimiento de los desarrolladores sobre los modelos o falta de formación en nuevas tecnologías.

La principal fuente de obtención de datos para el estudio de la DT son los comentarios que desarrolladores realizan directamente en el código o cuando realizan acciones de pullrequest o commits. Por lo que se le ha denominado DT auto admitida. Estos comentarios son ubicados en los repositorios de las herramientas como PyTorch, Keras, Caffe, Theano, etc., que al ser de código abierto es posible acceder a ellos. Otra fuente de obtención de datos son los comentarios que usuarios de las herramientas de DL realizan en sitios de las propias herramientas o en sitios web como *Stackoverflow.*

Para extraer los comentarios, se realiza manualmente o bien, se utilizan herramientas automáticas, para su clasificación se identificó que en varios de los trabajos aplican técnicas manuales, pero en trabajos recientes se han aplicado herramientas de minería de texto, redes neuronales e incluso algoritmos de procesamiento de lenguaje natural. En cuanto a la medición de la DT, se observó la aplicación de métodos cuantitativos y cualitativos con técnicas estadísticas

Es importante mencionar que durante la revisión se observó la existencia de diversos e importantes investigaciones relacionados con el estudio de la DT auto admitida y su caracterización. Sin embargo, muchos de estos estudios se enfocaron al software tradicional, pero existen muy pocos en los sistemas de aprendizaje automático y, en menos cantidad, se ubican los estudios relacionados con la identificación y medición de la DT en herramientas de DL. En [16] realizan un estudio sobre DT auto admitida en software tradicional y en software de ML que afirman que la DT aparece en una etapa más temprana del proceso de desarrollo.

Lo anterior podría deberse a que desde su surgimiento en 1990 el concepto propuesto por Ward Cunnigham, ha sido ampliamente estudiado en Ingeniería de Software y metodologías ágiles. Sin embargo, aunque el DL surgió en los años 40 con las primeras redes neuronales, este ha ganado relevancia en los últimos 15 años [29], además sus tecnologías e investigaciones evoluciona rápidamente, tanto en aspectos de infraestructura, modelos y herramientas, por lo que consideramos que, si bien hay avances, aún se está en proceso de definir claramente el concepto DT en DL y como medirla, para entonces poder vislumbrar su impacto. Otro aspecto importante a considerar y que no facilita su análisis es que la deuda no solo es código, también se involucran datos y modelos, incluso infraestructura, su interacción pueda ocultarla, Sculley et al. [7], además la naturaleza no determinista de los sistemas ML complica la aplicación de la Ingeniería de Software Giray [30], Nascimento et al. [17] investigaron como la ingeniería de software ha sido aplicada en el desarrollo de sistemas de ML e identificaron los retos y prácticas aplicables a las necesidades profesionales. Podemos afirmar que, para software tradicional, existen métricas bien definidas para evaluar la DT, y en DL como hemos observado se estan explorando. Nos damos cuenta también que muchas investigaciones sobre DT provienen de la comunidad en Ingeniería de Software y poco de las comunidades de Inteligencia Artificial y Aprendizaje Automático, pareciera ser que existe un puente que separa objetivos y metodologías que permitan identificar, definir, medir y mitigar esta deuda.

Con base en lo anterior podemos visualizar áreas de oportunidad como:

1. Dado que, la fuente de información principal para el estudio de la DT en DL proviene de comentarios que son introducidos deliberadamente por desarrolladores, misma que es conocida como DT auto admitida, consideramos pertinente un área futura de investigación, enfocada al estudio de la Deuda Técnica Auto Admitida en Herramientas de Aprendizaje Profundo, concepto que daría pauta para estudiar su origen,

caracterización, evolución e impacto.

2. Consideramos como área de oportunidad explorar herramientas de clasificación y predicción usando PLN, ya que ha mostrado resultados importantes en el análisis y predicción de textos; y dado que la DT auto admitida tomo como base el análisis de comentarios, una oportunidad de investigación es el estudio, desarrollo y aplicación una metodología basada en PLN que contribuyan a las existentes.

3. Finalmente, dado el crecimiento del desarrollo de sistemas basados en DL y su impacto en la sociedad, es crucial continuar investigando cómo la DT afecta su evolución y calidad. Los resultados obtenidos muestran avances relevantes, nos damos cuenta de la importancia y pertinencia de esta revisión pues puede sentar las bases para futuras investigaciones que evalúen el impacto de la DT en el desarrollo de sistemas DL y su repercusión en la sociedad.

Conclusiones

Esta revisión sistemática identificó enfoques para detectar y medir Deuda Técnica en herramientas de Aprendizaje Profundo, destacando que para su análisis se recurre a la extracción de comentarios en repositorios de GitHub, mismos que realizan los desarrolladores.

La clasificación más usada es la de [20], que distingue Deuda Tecnica en diseño, defectos, documentación y pruebas. Sin embargo, estudios recientes en herramientas de Aprendizaje profunfo incluyen categorías como Deuda Técnica de algoritmo, compatibilidad, dependencia de datos, modelo e infraestructura.

Para su medición, se observó el uso de métricas cuantitativas y enfoques cualitativos como revisiones de código, minería de texto y Procesamiento de Lenguaje Natural.

Como trabajo futuro, se plantea la oportunidad de realizar un estudio descriptivo de la DT en DL y con una metodología con enfoque en ingeniería de software y con el uso de algoritmos y herramientas de PLN, para en un futuro evaluar el impacto de la DT en aplicaciones DL en ciencia y sociedad.

Agradecimientos

Expreso mi profundo agradecimiento a Tecnológico Nacional de México-Instituto Tecnológico de Apizaco por brindarme la formación académica y los recursos necesarios para el desarrollo de esta investigación, también por las facilidades para poder dedicar tiempo a este proyecto, así como a mis colegas por su colaboración y valiosas contribuciones. Finalmente expreso mi agradecimiento a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación en México por su respaldo y compromiso con la innovación, lo que ha hecho posible

la realización de este trabajo.

Contribución de Autoría

Elizabeth Cuatecontzi Cuahutle: Conceptualización, Metodología, Investigación, Software, Validación, Redacción - borrador original. María Guadalupe Medina Barrera: Conceptualización, Metodología, Análisis formal, Recursos, Visualización, Supervisión, Administración de proyectos, Adquisición de fondos, Curación de datos, Escritura, revisión y edición. Raúl Cortes Maldonado: Conceptualización, Metodología, Análisis formal, Recursos, Visualización, Supervisión, Administración de proyectos, Curación de datos, Escritura, revisión y edición. Carlos Bueno Avendaño: Conceptualización, Metodología, Análisis formal, Recursos, Visualización, Supervisión, Administración de proyectos, Curación de datos, Escritura, revisión y edición.

Referencias

[1] R. Y. Choi, A. S. Coyner, J. Kalpathy-Cramer, M. F. Chiang, and J. P. Campbell, "Introduction to machine learning, neural networks, and deep learning," *Transl Vis Sci Technol*, vol. 9, no. 2, 2020.

[2] R. Elshawi, A. Wahab, A. Barnawi, and S. Sakr, "Dlbench: a comprehensive experimental evaluation of deep learning frameworks," *Cluster Comput*, vol. 24, no. 3, pp. 2017–2038, 2021.

[3] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021.

[4] M. H. M. Noor and A. O. Ige, "A survey on state-of-the-art deep learning applications and challenges," 2024.

[5] N. L. Rane, S. K. Mallick, O. Kaya, and J. Rane, "Tools and frameworks for machine learning and deep learning: A review," in *Applied Machine Learning and Deep Learning: Architectures and Techniques*. Deep Science Publishing, 2024.

[6] C. Yang, P. Liang, L. Fu, and Z. Li, "Self-claimed assumptions in deep learning frameworks: An exploratory study," in *Evaluation and Assessment in Software Engineering*. New York, NY, USA: ACM, Jun. 2021, pp. 139–148.

[7] D. Sculley, D. Holt, E. Davydov, and T. Phillips, "Hidden technical debt in machine learning systems," *Adv Neural Inf Process Syst*, 2015.

[8] D. OBrien, S. Biswas, S. Imtiaz, R. Abdalkareem, E. Shihab, and H. Rajan, "23 shades of self-admitted technical debt: an empirical study on machine learning software," in *Proceedings of the 30th ACM Joint*

European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA: ACM, Nov. 2022, pp. 734–746.

[9] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571–583, 2007.

[10] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Inf Softw Technol*, vol. 55, no. 12, pp. 2049–2075, 2013.

[11] S. Dong, P. Wang, and K. Abbas, "A survey on deep learning and its applications," *Comput Sci Rev*, vol. 40, p. 100379, 2021.

[12] S. Shamshirband, M. Fathi, A. Dehzangi, A. T. Chronopoulos, and H. Alinejad-Rokny, "A review on deep learning approaches in healthcare systems: Taxonomies, challenges, and open issues," *J Biomed Inform*, vol. 113, p. 103627, 2021.

[13] C. Aracena, F. Villena, F. Arias, and J. Dunstan, "Applications of machine learning in healthcare," *Revista Medica Clinica Las Condes*, vol. 33, no. 6, pp. 568–575, 2022.

[14] C. Yang, P. Liang, and Z. Ma, "An exploratory study on automatic identification of assumptions in the development of deep learning frameworks," *Sci Comput Program*, vol. 240, p. 103218, 2024.

[15] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "Is using deep learning frameworks free?" in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*. New York, NY, USA: ACM, Jun. 2020, pp. 1–10.

[16] A. Bhatia, F. Khomh, B. Adams, and A. E. Hassan, "An empirical study of self-admitted technical debt in machine learning software," 2024.

[17] E. Nascimento, A. Nguyen-Duc, I. Sundbø, and T. Conte, "Software engineering for artificial intelligence and machine learning software: A systematic literature review," 2020.

[18] A. Serban, K. van der Blom, H. Hoos, and J. Visser, "Software engineering practices for machine learning — adoption, effects, and team assessment," *Journal of Systems and Software*, vol. 209, p. 111907, 2024.

[19] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, Sep. 2014, pp. 91–100.

[20] E. da S. Maldonado and E. Shihab, "Detecting and quantifying different types of self-admitted technical debt," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*. IEEE, Oct. 2015, pp. 9–15.

[21] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks," *Empir Softw Eng*, vol. 26, no. 2, p. 16, 2021.

[22] Y. Tang, R. Khatchadourian, M. Bagherzadeh, R. Singh, A. Stewart, and A. Raja, "An empirical study of refactorings and technical debt in machine learning systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, May 2021, pp. 238–250.

[23] ——, "An empirical study of refactorings and technical debt in machine learning systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, May 2021, pp. 238–250.

[24] F. Pepe, F. Zampetti, A. Mastropaolo, G. Bavota, and M. D. Penta, "A taxonomy of self-admitted technical debt in deep learning systems," 2024.

[25] C. Liu, R. Cai, Y. Zhou, X. Chen, H. Hu, and M. Yan, "Understanding the implementation issues when using deep learning frameworks," *Inf Softw Technol*, vol. 166, p. 107367, 2024.

[26] Z. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "Satd detector," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. New York, NY, USA: ACM, May 2018, pp. 9–12.

[27] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa:ultra-large-scale software repository and source-code mining," *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 1, pp. 1–34, 2015.

[28] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," 2019.

[29] O. S. Ekundayo and A. E.-S. Ezugwu, "Deep learning: Historical overview from inception to actualization, models, applications and future trends," 2024.

[30] G. Giray, "A software engineering perspective on engineering machine learning systems: State of the art and challenges," *Journal of Systems and Software*, vol. 180, p. 111031, 2021.