PROYECTO DE TITULACIÓN

Departamento de Ciencias de la Computación

Carrera de Tecnologías de la Información

Estudio Comparativo entre Modelos Basados en la Arquitectura Transformer para Detectar Texto Maligno

Ponce Bravo, Bryan Steeven

Ing. Benavides Astudillo, Diego Eduardo, Mgtr.

Santo Domingo, 01 de marzo 2024

REPORTE DE VERIFICACIÓN DE CONTENIDO

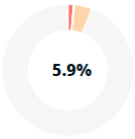
Plagiarism and AI Content Detection Report

Proyecto UIC Bryan Ponce.pdf

Scan details

Scan time:	Total Pages:	Total Words:
March 5th, 2024 at 16:0 UTC	115	28537

Plagiarism Detection



Types of plagiarism		Words
Identical	1.2%	356
Minor Changes	0.2%	60
Paraphrased	3.9%	1110
Omitted Words	9.9%	2821

AI Content Detection

Text coverage		Words
AI text	0%	0
Human text	100%	25716

Learn more

Firmado electrónicamente por:
DIEGO EDUARDO BENAVIDES ASTUDILLO

.....

Ing. Diego Eduardo Benavides Astudillo, Mgtr.

C.C: 1712883063

Departamento de Ciencias de la Computación

Carrera de Tecnologías de la Información

Certificación

Certifico que el trabajo de integración curricular: **"Estudio Comparativo entre Modelos Basados en la Arquitectura Transformer para Detectar Texto Maligno"**, fue realizado por el señor **Ponce Bravo, Bryan Steeven,** el mismo que cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, además fue revisado y analizado en su totalidad por la herramienta de prevención y verificación de similitud de contenidos; razón por la cual me permito acreditar y autorizar para que se lo sustente públicamente.

Santo Domingo de los Tsáchilas, 01 de marzo de 2024

Firmado electrónicamente por:
DIEGO EDUARDO
BENAVIDES ASTUDILLO

.....

Benavides Astudillo, Diego Eduardo

C. C: 1712883063

Departamento de ciencias de la computación

Carrera de tecnologías de la información

Responsabilidad de Autoría

Yo, **Ponce Bravo, Bryan Steeven**, con cédula de ciudadanía n°2300626781, declaro que el contenido, ideas y criterios del trabajo de integración curricular**: Estudio Comparativo entre Modelos Basados en la Arquitectura Transformer para Detectar Texto Maligno,** es de mi autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos, y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Santo Domingo de los Tsáchilas, 01 de marzo de 2024

.....

Ponce Bravo, Bryan Steeven

C.C.: 2300626781

Departamento de ciencias de la computación

Carrera de tecnología de la información

Autorización de Publicación

Yo, **Ponce Bravo, Bryan Steeven**, con cédula de ciudadanía n°2300626781, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de integración curricular: **Estudio Comparativo entre Modelos Basados en la Arquitectura Transformer para Detectar Texto Maligno**: en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Santo domingo de los Tsáchilas, 01 de marzo 2024

Firma

……..

Ponce Bravo, Bryan Steeven

C.C.:2300626781

Dedicatoria

En primer lugar, dedico este logro a Dios por guiarme en cada paso, por infundir en mi corazón la luz de la perseverancia y por brindarme la fortaleza en momentos de duda.

A mi querida madre Ramona Bravo, pilar de comprensión y amor incondicional, le dedico mi esfuerzo como un humilde testimonio del amor y la dedicación que ha vertido en mí. Su fe inquebrantable en mis capacidades y su apoyo incansable han sido la brújula que ha guiado mi camino hacia mis sueños.

A mi estimado padre Tulio Ponce, cuya sabiduría y ejemplo de trabajo ha construido mi carácter y aspiraciones, le ofrezco este trabajo. Su enseñanza de que la verdadera victoria se encuentra en la perseverancia y la entrega. Gracias por ser mi roca y por mostrarme el valor de la dedicación y el trabajo duro.

A mis estimados docentes, arquitectos del conocimiento y guardianes incansables de la sabiduría, extiendo mi más sincero reconocimiento. Su pasión por enseñar, su paciencia infinita y su dedicación para iluminar el camino del aprendizaje han dejado una huella imborrable en mi alma. Gracias por desafiar mis límites, por alimentar mi curiosidad y por inspirarme a buscar siempre la excelencia.

Ponce Bravo, Bryan Steeven

Agradecimiento

Quiero expresar mi gratitud más sincera a la Universidad de las Fuerzas Armadas-ESPE por brindarme las herramientas, los recursos y un entorno favorable para mi formación académica y personal. La excelencia y dedicación de la institución han sido esenciales en mi camino hacia mi título.

El docente tutor ingeniero Eduardo Benavides merece un reconocimiento especial, ya que su orientación, paciencia y conocimiento académico han sido cruciales para lograr el trabajo de tesis. Su constante apoyo, críticas constructivas y motivación me han permitido superar los desafíos y alcanzar mis objetivos con mayor claridad y confianza.

La Universidad de las Fuerzas Armadas-ESPE y el ingeniero Eduardo Benavides han sido fundamentales para mi crecimiento académico y profesional. Gracias por creer en mi potencial y por estar a mi lado en este momento crucial de mi carrera.

Ponce Bravo, Bryan Steeven

Índice

Índice de fíguras

Índice de tablas

Índice de Algoritmos

RESUMEN

El phishing es un tipo de ataque cibernético cuyo objetivo es engañar a los usuarios, generalmente a través de páginas web aparentemente benignas. Actualmente, una de las formas más comunes de detectar estas páginas de phishing es mediante el análisis de su contenido. Esto implica analizar el texto de las páginas web y posteriormente examinar ese contenido con algoritmos de Deep Learning (DL). Según el estado del arte, el texto se introduce de forma secuencial en los algoritmos de DL, es decir, sin considerar el orden o el significado de las palabras. Este método, por lo tanto, ignora la riqueza semántica inherente a las relaciones entre las palabras. La innovación de este estudio propone un modelo que emplea el Procesamiento de Lenguaje Natural (NLP) y algoritmos Transformer de DL para detectar ataques de phishing basándose en el texto extraído de páginas web sospechosas. En este trabajo, se utiliza la Metodología de Descubrimiento de Conocimiento en Bases de Datos (KDD) para realizar un análisis comparativo de cuatro modelos basados en la arquitectura Transformer, junto con NLP, para identificar ataques de phishing a partir del texto contenido en dichos ataques. Se inicia con la selección y preprocesamiento de un conjunto de datos obtenido del sitio PhishTank, que incluye textos de páginas de phishing y textos de páginas legítimas (ham). Posteriormente, se aplican técnicas de limpieza, tokenización y lematización para preparar los datos para el análisis. Para maximizar las características semánticas y sintácticas del contenido de la página web, se emplea la capa de incrustación de Keras con GloVe. Finalmente, estos datos se procesan con cuatro modelos Transformer para determinar qué algoritmo funciona mejor: BERT, GPT, T5 y XLNet. Las evaluaciones experimentales demostraron que el modelo BERT, con un 93.34%, obtuvo la mejor precisión. Como complemento a nuestra investigación, se desarrolló una extensión para el navegador Chrome que permite a los usuarios verificar si una página es de phishing o no.

Palabras clave: Phishing, Aprendizaje Profundo, Procesamiento de Lenguaje Natural, Transformer, BERT

ABSTRACT

Phishing is a form of cyber attack whose purpose is to deceive users, generally through websites that appear benign. Currently, one of the most common ways to detect these phishing pages is based on their content. That is, analyzing the text on web pages and then analyzing that content with deep learning (DL) algorithms. According to the state of the art, this text is entered consecutively into the DL algorithms, regardless of the order or meaning of the words. In this way, this method loses the richness of semantics between the relationship between words. The innovation of our study is to propose a model that uses Natural Language Processing (NLP) and DL Transformer algorithms to detect phishing attacks based on the text extracted from suspicious web pages. In this work, the Knowledge Discovery Methodology in Databases (KDD) is used to perform a comparative analysis of four models based on the Transformer architecture together with NLP to detect phishing attacks on the text contained in these attacks. We begin with selecting and pre-processing a data set obtained from the Phishload site, including text from phishing pages and text from legitimate (ham) pages. Then, we use cleansing techniques, tokenization, and lemmatization to prepare them for analysis. To maximize the semantic and syntactic features of the web page content, we utilize the Keras embedding layer with GloVe. Finally, this data was processed with four Transformer models to determine which algorithm works best: BERT, GPT, T5, and XLnet. Experimental evaluations showed that the 93.03% BERT model obtained the best accuracy. An extension was made to complement our research in the Chrome browser, allowing users to determine if a page is phishing.

Keywords: Phishing, Deep Learning, Natural Language Processing, Transformer, BERT

1. Introducción y estado del arte:

La ingeniería social es un tipo de ataque cibernético en el que el atacante engaña y estafa a la víctima utilizando manipulación psicológica, abusando de su comportamiento desprevenido o sus rasgos ingenuos de la personalidad [1]. El phishing mediante uso de sitios web fraudulentos, frecuentemente son una copia de la página real, es una de las formas por las que se puede llevar a cabo este tipo de ataque de ingeniería social [2]. En otras palabras, el phishing es un ataque de ingeniería social destinado a engañar a los usuarios, para cometer fraude contra esos mismos usuarios o sus organizaciones. El phishing es un delito donde se emplea ingeniería social y técnicas de engaño para robar los datos de identidad y las credenciales de cuentas financieras de los consumidores [3].

Según [4], phishing es un acto escalable de engaño utilizando una identidad falsa para obtener información de un objetivo. Identificar si una página web está en una lista negra es el método más común para detectar phishing. El problema de las listas negras es que solo almacenan URL identificadas como páginas de phishing; por lo tanto, este método no tiene la función para identificar sitios web de phishing nuevas [5]. Varios métodos para abordar el mismo problema de detección de phishing utilizan Machine Learning (ML). Los modelos de DL son superiores a los modelos ML en términos de precisión, mientras que los modelos ML son superiores en términos de tiempos de ejecución [6].

Por otra parte, NLP es una forma de representar el lenguaje común de las personas. Los enfoques secuencial y no secuencial son las dos opciones disponibles para los trabajos revisados de DL que analiza el texto de las páginas de phishing. Los algoritmos de DL generalmente reciben texto de manera no secuencial, es decir, no les importa en qué orden se introducen las palabras; como resultado, pierden el significado semántico del texto que reciben [7]. En este estudio, se empleó un enfoque secuencial, encapsulando la secuencia de datos, manteniendo el significado semántico y sintáctico, y utilizando la distancia dimensional para describir las relaciones entre las palabras [8].

El propósito del trabajo, es analizar y comparar cuatro modelos basados en la arquitectura Transformer junto con NLP para detectar ataques de phishing sobre el texto contenido en esos ataques, utilizando la metodología de Descubrimiento de Conocimientos en Bases de Datos (KDD, por sus siglas en inglés, Knowledge Discovery in Databases). Utilizando esta técnica, se identifican patrones y características importantes en el contenido de phishing, pueden ser

complejos para los métodos de detección convencionales.

El presente trabajo comienza con una revisión de la literatura sobre ataques de phishing y tecnologías de detección, seguido de una descripción detallada de la metodología KDD y la implementación de modelos Transformer en NLP. Posteriormente, se plantea un análisis comparativo de los hallazgos y se discute su relevancia para la creación de herramientas de seguridad cibernética más efectivas y robustas. Finalmente, concluimos con una reflexión sobre los hallazgos del estudio y recomendaciones para investigaciones futuras.

1.1. Estado del arte:

Los autores [9], proponen la obtención de conocimientos útiles a partir de las URL con la ayuda de diferentes modelos, especialmente modelos basados en Transformer como las Representaciones Codificadoras Bidireccionales de Transformer (BERT, por sus siglas en inglés, Bidirectional Encoder Representations from Transformers), destaca por su capacidad para descifrar los complejos patrones y la semántica innata en las URL mediante el sofisticado mecanismo de NLP. Después de utilizar modelos pre entrenados para la extracción de optimizaciones a partir de características particulares, se puede proseguir con la aplicación de la estrategia de apilamiento involucrando diferentes modelos de ML, lo que permite mejorar significativamente la predicción. El enfoque de conjunto, que combina los puntos fuertes de varios modelos para lograr una mayor calidad de predicción, representa un esfuerzo innovador para prometer una predicción más fiable y precisa en el análisis de URL.

En la investigación [10], sugiere un método para crear preguntas de desafío respuesta generadas dinámicamente que se basen en interacciones históricas, como la comunicación diaria por correo electrónico o plataformas de mensajería. Para lograrlo, crearon modelos DL con BERT y realizaron análisis semánticos para extraer y validar preguntas de desafío de respuesta muy efectivas. Las preguntas están diseñadas para que los usuarios reales las respondan fácilmente, pero también presentan un desafío impredecible para los atacantes potenciales. Los investigadores desarrollaron un sistema de verificación de remitentes de correo electrónico que puede identificar remitentes sospechosos en tiempo real, demostrando un caso de uso efectivo de su sistema propuesto. Este sistema crea preguntas de desafío basadas en intercambios de correo electrónico anteriores con usuarios auténticos, lo que permite identificar correos electrónicos de spear-phishing avanzados que de otra manera permanecerían sin detectar. Utilizando conjuntos de datos anotados y un estudio con usuarios, se evaluó la eficacia del sistema VeriActor. Los resultados experimentales

fueron positivos: la precisión de verificación del 87.8% y una precisión de protección del 83.33%. En la investigación [11], presenta un enfoque para la detección de Compromisos de Correo Electrónico Empresarial utilizando un método basado en Transformer, permite capturar las propiedades lingüísticas de los correos electrónicos, reduciendo la dependencia de indicadores explícitos. La metodología combinada de BERT y BiLSTM, logra capturar tanto el contexto global como la interdependencia local, resultando en una comprensión rica y matizada del texto del correo electrónico. Los experimentos realizados demostraron que el enfoque propuesto supera todas las soluciones de última generación al alcanzar un 99% de precisión, revelando así el alto potencial de los modelos basados en Transformer para la detección de ataques de Correo Electrónico Empresarial. El avance indica un cambio significativo en la eficiencia para detectar y prevenir tales amenazas, representando un momento trascendental en la ciberseguridad y en la defensa contra el compromiso de la comunicación empresarial.

En el artículo [12], Los investigadores en ciberseguridad están motivados a crear modelos inteligentes y brindar servicios seguros en la web debido a la creciente sofisticación y maldad de los ataques de phishing. Implementaron una técnica novedosa para la detección de phishing en URL, basada en la extracción de características mediante BERT y un método de DL. Utilizando BERT, se obtuvo el texto de las URL del conjunto de datos ”Phishing Site Predict”, aplicando posteriormente un algoritmo de NLP a la columna de datos única y extrayendo un gran número de características de datos útiles en términos de información textual significativa. Luego, se empleó un método de red neuronal convolucional profunda (CNN, por sus siglas en inglés, Convolutional Neural Network) para detectar URL de phishing, constituyendo palabras o secuencias de caracteres para extraer características de nivel superior y clasificando los datos en URL legítimas y de phishing. Para evaluar el método propuesto, utilizaron un conocido conjunto de datos público de URL de sitios web de phishing, con un total de 549.346 entradas. Se desarrolló tres escenarios para comparar los resultados del método propuesto utilizando conjuntos de datos similares, basando el proceso de extracción de características en técnicas de NLP. Los experimentos demostraron que el método propuesto alcanzó un 96.66% de precisión en los resultados, los cuales se compararon posteriormente con otros trabajos de revisión bibliográfica. La investigación [13], se centra en la segmentación de URL para proponer un nuevo método de tokenización, denominado URL-Tokenizer, que combina el tokenizador BERT y el tokenizador WordSegment, además de adoptar simultáneamente segmentaciones a nivel de carácter y a nivel de palabra. Las evaluaciones experimentales en la detección de URL de phishing muestran que

el método propuesto alcanza una alta precisión del 95.7% con un conjunto de datos equilibrado, y del 97.7% con un conjunto de datos desequilibrado, mientras que los modelos de referencia alcanzaron el 85.4% con un conjunto de datos equilibrado y el 85.1% con un conjunto de datos desequilibrado.

En la investigación [14], proponen un enfoque comparativo para el uso combinado de métodos de NLP como TF-IDF, Word2Vec y BERT, como algoritmos de ML, incluyendo Random Forest, Decision Tree, Logistic Regression, Gradient Boosting Trees y Naive Bayes, para detectar correos electrónicos de phishing. La evaluación se realizó en dos conjuntos de datos, uno equilibrado y otro desequilibrado, ambos compuestos por correos electrónicos del conocido corpus Enron y los correos más recientes del corpus de phishing Nazario. La mejor combinación en el conjunto de datos equilibrado resultó ser Word2Vec con el algoritmo Random Forest, mientras que en el conjunto de datos desequilibrado fue Word2Vec con el algoritmo de regresión logística.

En la investigación [15], se llevó a cabo un análisis exhaustivo de los modelos de Transformer en la tarea de detección de URL de phishing. Se consideró el modelado de lenguaje enmascarado estándar y tareas adicionales de pre entrenamiento específicas del dominio, comparando estos modelos con los modelos BERT y RoBERTa ajustados. Combinando los resultados de estos experimentos, se propone URLTran, que utiliza transformadores para mejorar significativamente el rendimiento de la detección de URL de phishing en un amplio rango de tasas de falsos positivos muy bajas en comparación con otros métodos basados en DL. Por ejemplo, URLTran produce una tasa de verdaderos positivos del 86.80% en comparación con el 71.20% de la siguiente mejor línea de base a una falsos positivos del 0.01% lo que resulta en una mejora relativa de más del 21.9%. Además, se tienen en cuenta algunos ataques de suplantación de identidad clásicos, como los basados en caracteres similares visualmente y en la división de palabras compuestas, para mejorar la estabilidad de URLTran. Se considera un ajuste fino adicional con estas muestras adversarias y se demuestra que URLTran puede mantener unos falsos positivos bajos en estos escenarios.

En la investigación [16], los autores presentaron la eficacia de la incrustación de palabras en la clasificación de correos electrónicos no deseados. Se ajusta el modelo BERT, que ya ha sido para que pueda detectar correos electrónicos de spam y los que no lo son. BERT utiliza capas de atención para tener en cuenta el contexto del texto. Los resultados se comparan con un modelo de red neuronal profunda que contiene una capa de memoria a corto plazo bidireccional a largo plazo y dos capas densas apiladas. Además, se comparan los resultados con un conjunto

de clasificadores clásicos como k-nearest neighbors y Naive Bayes. Se realizaron pruebas con dos conjuntos de datos de código abierto, uno para entrenar el modelo y otro para probar la persistencia y estabilidad del modelo frente a datos no vistos. El método propuesto obtuvo una precisión del 98.67% y una puntuación F1 del 98,66%.

En [17], se examina la eficacia de las incrustaciones de palabras para identificar correos electrónicos no deseados, adaptando el modelo BERT, un modelo Transformer destacado por su capacidad para analizar el contexto completo de un texto a través de sus capas de atención. Este modelo se contrasta tanto con una red neuronal profunda estándar, que incluye una capa BiLSTM seguida por dos capas densas, como con clasificadores tradicionales como el k-nearest neighbors y el Naive Bayes. Utilizando dos bases de datos públicas para el entrenamiento y la validación, los investigadores evalúan la eficiencia, precisión y estabilidad de estos modelos. Los resultados demuestran que el modelo adaptado logra una precisión del 98.67% y una puntuación F1 del 98.66%, subrayando su superioridad en la tarea de clasificación de correos electrónicos no deseados y marcando un avance significativo en la aplicación de técnicas de NLP para la seguridad informática.

Este estudio [18], los autores investigan la detección de ataques de phishing por correo electrónico utilizando técnicas de DL como RNN, redes de memoria a corto plazo de largo plazo, RNN y BERT. Se utilizaron datos de correos electrónicos maliciosos y phishing y se extrajeron características pertinentes utilizando técnicas de NLP. El modelo de DL propuesto se entrenó y probó con el conjunto de datos. Se encontró que, en comparación con otras investigaciones de vanguardia, puede detectar el phishing de correo electrónico con una alta precisión. Los mejores resultados se observaron cuando se utilizaron BERT y LSTM.

En la Tabla I, se realiza una comparación entre investigaciones relacionadas a la detección de phishing con DL y Transformer, y nuestro estudio.

Tabla I: Estado del arte de investigaciones relacionadas en la detección de phishing con DL y Transformer

Artículo	**Email**	**Página web**	**NLP**	**Precisión**	**Aplicación**
[9]	No	URL	Sí	N/A	N/A
[10]	Sí	N/A	Sí	87.8%	Sí
[11]	Sí	N/A	Sí	99%	N/A
[12]	No	URL	Sí	96.66%	N/A
[13]	No	URL	Sí	95.7%	N/A
[14]	Sí	N/A	Sí	N/A	N/A
[15]	No	URL	Sí	86.80%	N/A
[16]	Sí	N/A	Sí	98.66%	N/A
[17]	Sí	N/A	Sí	98.67%	N/A
[18]	Sí	N/A	Sí	99.68%	N/A
Nuestro estudio	No	Texto	Sí	93.34%	Sí

1.2. Objetivos

1.2.1. Objetivo General: Analizar y comparar diferentes modelos basados en la arquitectura Transformer junto con el procesamiento de lenguaje natural para detectar ataques de phishing sobre el texto contenido en esos ataques.

1.2.2. Objetivos Específicos:

- Determinar que aplicaciones usan actualmente las tecnologías Transformer para detectar ataques de phishing sobre el texto contenido en esos ataques.
- Determinar cuál de los modelos Transformer brinda mejores resultados para la detección de ataques de phishing sobre el texto contenido en esos ataques.
- Entrenar un modelo con el algoritmo Transformer y ponerlo en producción para la detección de ataques de phishing.
- Ofrecer un aplicativo en un navegador web para detectar ataques de phishing que sea amigable para el usuario.

2. MARCO TEÓRICO/MARCO CONCEPTUAL

En este capítulo, se construye todos los conceptos relacionados con el enfoque de estudio de este proyecto. Esto implica detallar todos los conceptos clave que son esenciales para entender el alcance y la profundidad de nuestro estudio.

2.1. Ingeniería Social

La ingeniería social es un método que busca persuadir a las personas para que cometan errores de seguridad, utilizando el engaño para obtener acceso a información confidencial, acceso o activos valiosos. Los delincuentes en este tipo de fraude cibernético engañan a los usuarios inadvertidos para que revelen información, distribuyan malware o permitan el acceso a sistemas de seguridad informática protegidos. Los ataques pueden ocurrir a través de una variedad de canales, como las interacciones en línea, las interacciones cara a cara u otros tipos de comunicación [19].

2.1.1. Tipos de ataques de Ingeniería Social: Los ataques de ingeniería social utilizan técnicas de manipulación psicológica para engañar a las personas y obtener datos confidenciales de manera no autorizada. Estas técnicas explotan la predisposición natural humana a confiar en otros. Entre los ataques de ingeniería social más habituales se incluyen: el Quishing, que utiliza el engaño a través de códigos QR; el Compromiso del Correo Electrónico Empresarial; la distribución de Software Falso; la Trampa de Miel; el Scareware; el Ataque del Agujero de Agua; el Buceo en Contenedores (Dumpster Diving); el Tailgating o Cola de Rata; el Pretexto (Pretexting); el Cebo (Baiting); y el Phishing, siendo este último el más reconocido, caracterizado por el envío de correos electrónicos fraudulentos para sustraer información sensible [20]. El phishing será el principal enfoque de este estudio.

2.2. Phishing

Es una práctica maliciosa generalmente realizada a través de correo electrónico o páginas web, con el objetivo de robar información personal [21]. Es el método más común utilizado por los piratas informáticos para realizar ataques con el fin de obtener información confidencial como credenciales de nuestra cuenta personal, datos bancarios e incluso información de redes sociales. En este tipo de ataque, los piratas informáticos engañan a los usuarios haciéndoles creer que están interactuando con una fuente confiable, con el propósito de manipularlos para

que divulguen información confidencial o ejecuten un pago, que terminan en manos del atacante [22].

2.2.1. Tipos de ataques de phishing: Existen varios tipos de ataques de phishing, tales como: phishing Tradicional, spear phishing, whaling, smishing, vishing, clone phishing, business email compromise, angler phishing y web phishing [23]. Este estudio se basa en la detección de web phishing.

2.3. NLP

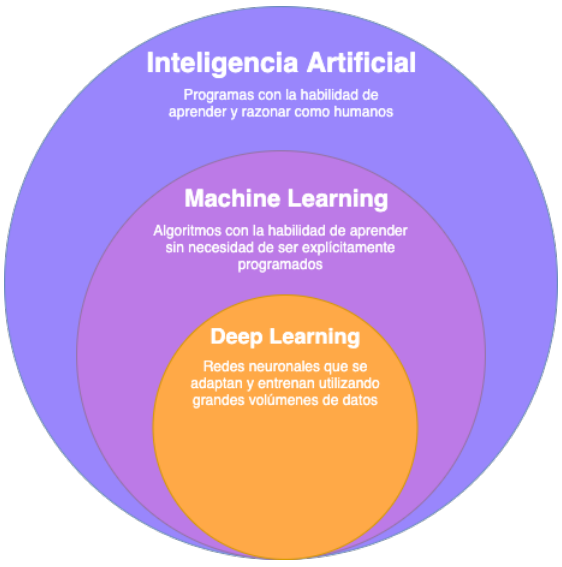
NLP es un campo que permite a los humanos comunicarse y expresarse a través del lenguaje, en este contexto se basa en la asociación de signos con significados concretos [24]. La escritura, las señales y la voz son las herramientas que utiliza el lenguaje para disponer una comunicación. Se distinguen dos tipos de lenguaje: el lenguaje natural y el lenguaje formal. El lenguaje natural incorpora idiomas como inglés, español, alemán entre otros, y que progresa constantemente sin seguir las reglas estrictas. Los lenguajes formales que se toma en los campos de la matemáticas, la lógica o la programación, y están estrictamente regulados por reglas predefinidas [25]. El lenguaje natural se distingue por su rico vocabulario y complejas construcciones gramaticales. Además, posee características como flexibilidad, ambigüedad e indeterminación, que enriquecen la comunicación humana al permitir una variedad de interpretaciones según el contexto. Sin embargo, estas mismas características representan desafíos significativos en el procesamiento computacional del lenguaje. Dificultan la implementación de procesos de razonamiento y, en particular, presentan obstáculos para la caracterización y formalización en el ámbito de la inteligencia artificial (AI, sus siglas en inglés, Artificial Intelligence) y el NLP [26]. NLP es un campo de investigación que tiene como objetivo comprender cómo funciona el lenguaje, su construcción, la generación de nuevos lenguajes y todas las tareas relacionadas con NLP. Estas tareas incluyen generar texto nuevo, traducir de un idioma a otro, preguntas y respuestas, generar resúmenes, chatbots ente otros [27].

2.4. Inteligencia Artificial

Es la capacidad que tienen las computadoras para realizar tareas que normalmente requieren la inteligencia humana [28]. Es un campo de la informática que se centra en la creación de algoritmos y programas, estos programas pueden contener procesos como el aprendizaje, el razonamiento, resolución de conjuntos de problemas, NLP, visión artificial y la toma de

decisiones. La AI se ha transformado en una rama de estudio y desarrollo clave en la computación y la tecnología [29]. La diferencia entre la AI y las capacidades humanas radica que las máquinas para operar continuamente, sin necesidad de descanso, y examinar grandes cantidades de información simultáneamente. Además, la AI tiende a tener menos probabilidades de cometer errores al realizar tareas que los humanos que realizan las mismas tareas. Este atributo resalta la eficiencia y coherencia inherentes de las operaciones de AI [28]. Esto es directamente relevante para NLP; especialmente cuando las investigaciones o experimentos buscan garantizar que las máquinas, ya sean robots o teléfonos móviles, generen oraciones y contexto en el lenguaje hablado y en general [30]. En la Figura 1, se muestra la diferencia de la AI, ML y DL. La AI se enfoca en la creación de máquinas inteligentes. ML se presenta como una subárea de la AI, permitiendo la creación de aplicaciones basadas en AI. Por otro lado, DL se posiciona como una subárea más específica de ML, haciendo uso de grandes cantidades de datos y algoritmos complejos para entrenar modelos [31].

Figura 1: La relación entre AI, ML y DL



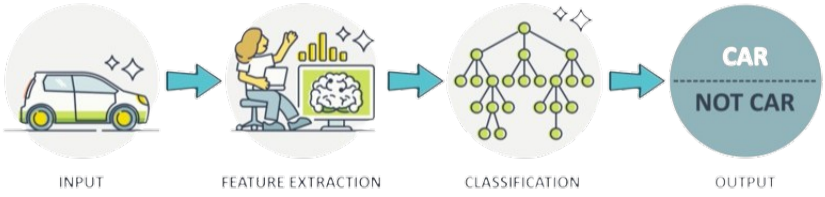
Nota. Recuperado de [32].

2.5. Machine Learning

Se refiere a un conjunto de algoritmos que aprenden y predicen a partir de datos registrados. Estos algoritmos buscan optimizar una función de utilidad específica en condiciones de incertidumbre, revelando estructuras ocultas en los datos y proporcionando descripciones detalladas de los mismos. La aplicación de ML es frecuente cuando la programación explícita no resulta práctica o se percibe como demasiado restrictiva. A diferencia del código convencional, donde los desarrolladores generan un resultado específico basado en un programa definido para una entrada determinada, ML utiliza datos para generar un código estadístico que encapsula el resultado [19]. Esta disciplina combina elementos de diversos campos, como la neurociencia, psicología, informática, física, matemáticas y medicina, se centra en el estudio de cómo se pueden extraer patrones y conocimientos de los datos [33]. Los sistemas pueden hacer proyecciones o tomar decisiones basándose en experiencias anteriores gracias a este proceso de aprendizaje de la información disponible. ML se puede dividir en tres categorías principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo [34]. ML tiene cuatro tareas de clasificación principales: clasificación binaria, clasificación multietiqueta, clasificación multiclase y clasificación de desbalanceada.

Clasificación binaria: El objetivo de una tarea de clasificación binaria es clasificar los datos de entrada en dos categorías mutuamente exclusivas. De acuerdo con el problema que se está tratando, los datos de capacitación se etiquetan en un formato binario: verdadero o falso; positivo o negativo; 0 y 1, carro o no carro [35]. En la figura 2, muestra un ejemplo de clasificación binaria de ML de un carro.

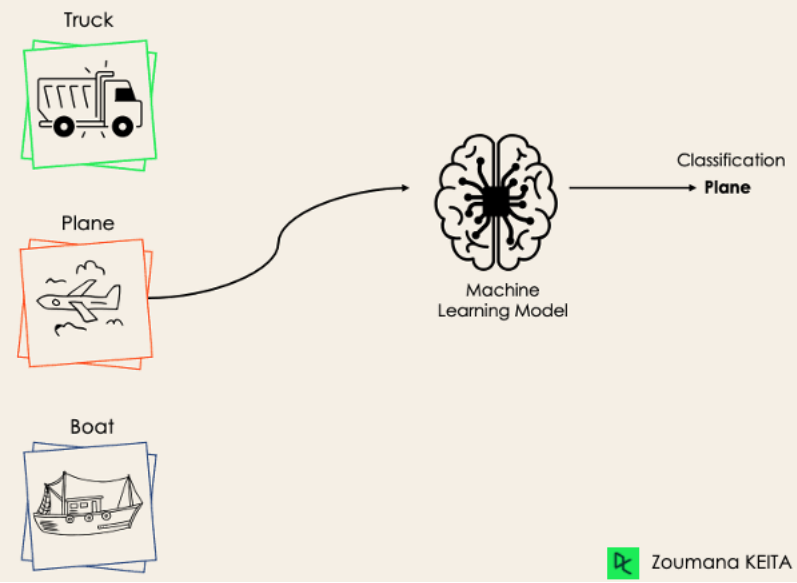
Figura 2: Clasificación binaria de ML



Nota. Recuperado de [36].

Clasificación multiclase: La clasificación multiclase tiene al menos dos etiquetas de clase mutuamente exclusivas, cuyo objetivo es predecir a qué clase pertenece un ejemplo de entrada específico. En la Figura 3, muestra un ejemplo de clasificación multiclase donde tiene varias clases. El modelo categorizó la imagen como un avión [35].

Figura 3: Ejemplo de Clasificación Multiclase en Machine Learning



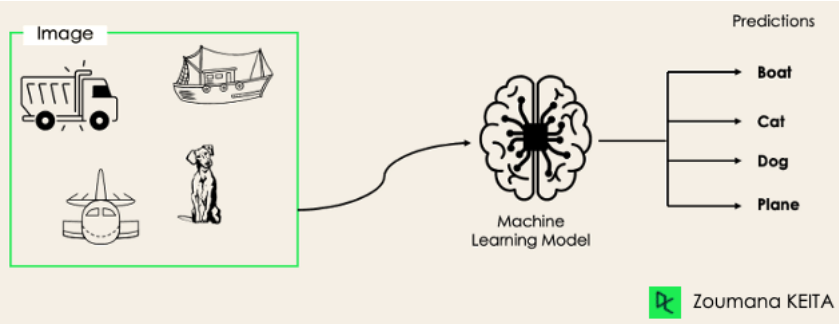
Nota. Recuperado de [35].

La mayoría de los algoritmos de clasificación binaria también se pueden utilizar para la clasificación de clases múltiples. Estos algoritmos incluyen:

- Random Forest
- Naive Bayes
- K-Nearest Neighbors
- Gradient Boosting
- SVM
- Regresión logística

Clasificación multietiqueta: En las tareas de clasificación de etiquetas múltiples, intentamos predecir 0 o más clases para cada ejemplo de entrada. Debido a que el ejemplo de entrada puede tener más de una etiqueta, en este caso no hay exclusión mutua. En la Figura 4, se muestra un ejemplo de clasificación multietiqueta.

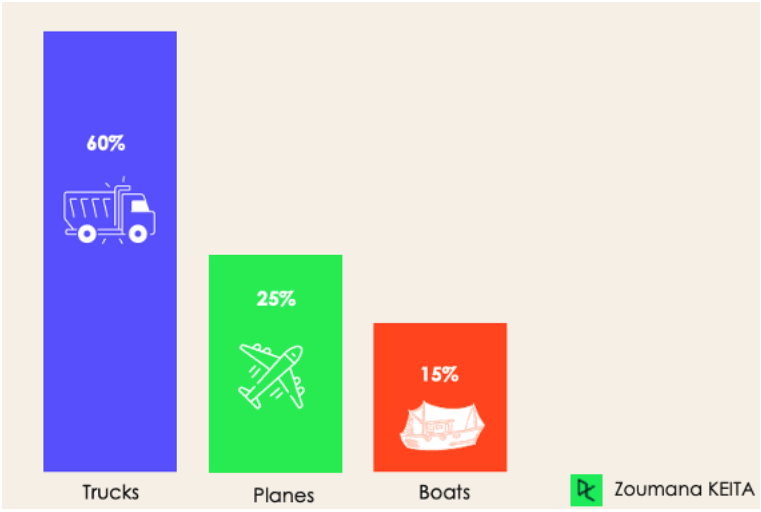
Figura 4: Ejemplo de Clasificación Multietiqueta en Machine Learning



Nota. Recuperado de [35].

Clasificación desbalanceada: La clasificación desequilibrada es cuando la cantidad de ejemplos en cada clase se distribuye de manera desigual [35].

Figura 5: Ejemplo de Clasificación desbalanceado en Machine Learning



Nota. Recuperado de [35].

2.5.1. Aprendizaje supervisado: El paradigma ML es el aprendizaje supervisado, en el que el objetivo principal es aprender un modelo que pueda hacer predicciones precisas a partir de datos previamente etiquetados. Este método se basa en un conjunto de datos de entrenamiento compuesto por pares de entrada-salida que ya tienen una etiqueta o valor de salida conocido

para cada entrada. El proceso implica entrenar un algoritmo para que pueda predecir la salida correspondiente con la mayor precisión posible al recibir nuevas entradas [37].

2.5.2. Aprendizaje no supervisado: El aprendizaje no supervisado, una forma de ML, utiliza datos sin etiquetar como entrada y permite a la máquina estudiar y organizar estos datos sin instrucciones ni formación previa; de esta manera, ayuda a identificar patrones latentes. La máquina no le enseña nada sobre el significado de los datos o cómo encontrarlos; en cambio, uno debe aprender los patrones de la estructura de los datos. Los datos no ofrecen información definitiva sobre las predicciones del modelo, y las respuestas que ofrecen no siempre son precisas. Como resultado, el aprendizaje no supervisado se convierte en un método para explorar las estructuras de información y extraer información útil de los datos sin conjuntos de entrenamiento pre-etiquetados [38]. Aprendizaje no supervisado se puede dividir en tres tareas principales:

- Por agrupamiento
- Reglas de asociación
- Reducción de dimensionalidad

2.5.2a. Por agrupamiento:

Los métodos de agrupación agrupan los datos sin etiquetar según sus similitudes y diferencias. Podemos inferir que dos instancias tienen características distintas cuando aparecen en grupos diferentes. Se dividen en diferentes tipos de agrupación [39].

- **Agrupación exclusivo:** Un solo punto de datos pertenece a un solo grupo cuando se agrupan los datos.
- **Agrupación por superposición:** Un único punto de datos puede pertenecer a varias agrupaciones con diferentes grados de pertenencia.
- **Agrupación por aglomeración:** Es un tipo de agrupación en la que se forman grupos de manera que instancias similares estén dentro del mismo grupo y objetos diferentes estén dentro de otros grupos.
- **Agrupación Probabilística:** Una distribución de probabilidad se utiliza para crear clústeres.

2.5.2b. Reglas de asociación:

La minería de reglas de asociación es una técnica de minería de datos que los minoristas utilizan para obtener una mejor comprensión de los patrones de compra de los clientes en función de las relaciones entre varios productos. Se utiliza con frecuencia en el análisis de la cesta de la compra. El algoritmo Apriori es el algoritmo de aprendizaje de reglas de asociación más utilizado.

Sin embargo, para este tipo de aprendizaje no supervisado, se utilizan otros algoritmos, como Eclat y FP-growth [39].

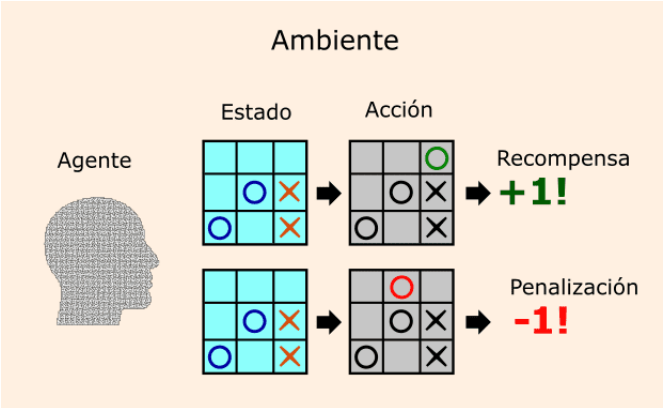
2.5.2c. Reducción de dimensionalidad:

El análisis de componentes principales y la descomposición de valores singulares son dos algoritmos comunes para la reducción de dimensionalidad. Estos algoritmos tienen como objetivo convertir datos de espacios de alta dimensión a espacios de baja dimensión sin afectar las propiedades importantes de los datos iniciales. El análisis de datos exploratorios o el procesamiento de datos para prepararlos para el modelado son dos áreas en las que se utilizan con frecuencia estas técnicas. Durante EDA, es útil reducir la dimensionalidad de un conjunto de datos para ayudar a visualizar los datos. Esto se debe a que es difícil visualizar datos en más de tres dimensiones. Desde el punto de vista del procesamiento de datos, reducir la dimensionalidad de los datos hace que sea más fácil modelar [39].

2.5.2d. Aprendizaje por refuerzo:

El aprendizaje por refuerzo es un método de inteligencia artificial que imita cómo los organismos naturales aprenden de su entorno con el fin de maximizar algunas señales de recompensa[40]. Este proceso implica que un agente, en un ambiente, toma decisiones secuenciales que afectan su estado futuro con el objetivo de acumular la mayor cantidad de recompensas posibles. El aprendizaje por refuerzo se basa en la idea de que las acciones que producen resultados positivos son reforzadas y más propensas a ser elegidas en el futuro [41]. En la Figura 6, se muestra como funciona el aprendizaje por refuerzo

Figura 6: Funcionamiento del aprendizaje por refuerzo



Nota. Recuperado de [42].

El aprendizaje por refuerzo se diferencia de otros tipos de aprendizaje de AI, como el aprendizaje supervisado y no supervisado, porque se enfoca en aprender políticas de control óptimas a través de la experiencia directa y la exploración del entorno, sin necesidad de un conjunto de datos predefinido [43]. En aprendizaje por refuerzo, el agente aprende a realizar tareas a través de pruebas y errores, evaluando las consecuencias de sus acciones mediante recompensas o penalizaciones [44].

2.6. Tensorflow

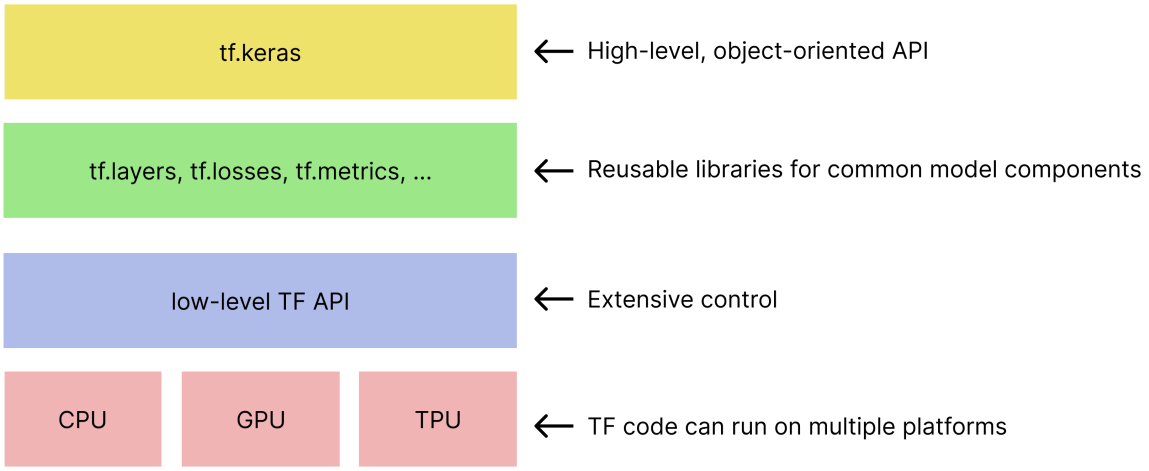
2.6.1. Origen: Google Brain, que creó DistBelief, lanzó la versión inicial de TensorFlow en 2015 para reemplazarlo como su sistema de DL interno [45]. Ha sido utilizado desde entonces por Google para mejorar sus servicios como Google Search, Translate y Gmail, así como por una variedad de empresas y organizaciones globales, incluidos el Departamento de Defensa de los Estados Unidos y PayPal [45]. Esto demuestra que TensorFlow es muy popular y versátil como plataforma de ML. Por lo tanto, el alto nivel de reconocimiento y generalización de TensorFlow indica que es una plataforma de ML valiosa.

2.6.2. Definición: TensorFlow es una plataforma completamente gratuita y de código abierto para el ML. Sin embargo, esta clase se centra en el uso de una API específica de TensorFlow para desarrollar y entrenar modelos de ML, a pesar de que TensorFlow es un sistema completo para administrar todos los aspectos de un sistema de ML [46].

2.6.3. Arquitectura: La característica distintiva de la arquitectura de TensorFlow es su capacidad para representar cálculos a través de gráficos de flujo de datos, lo que permite la ejecución eficiente y adaptable de operaciones matemáticas complejas distribuidas a través de múltiples unidades de procesamiento, como CPU y GPU. En términos de agnosticismo, TensorFlow ayuda en el entrenamiento y la inferencia de modelos de ML y lo hace sin límites. Esta adaptabilidad es crucial para adaptarse a diferentes campos, como la identificación de voz o texto o la detección de objetos para imágenes o vídeos [47]. Además de su diseño modular, la arquitectura de TensorFlow permite a los usuarios crear modelos de ML mediante la composición de funciones y otras construcciones familiares de lenguajes de programación. Las API de TensorFlow están organizadas en jerarquía, con las API de bajo nivel siendo la base de las API de alto nivel. Los investigadores de ML crean y exploran nuevos algoritmos utilizando API de bajo nivel. Para crear, entrenar y predecir modelos de ML en esta clase, usarás una API de alto nivel llamada

tf.keras. La API de Keras de código abierto tf.keras emula TensorFlow [46]. En la Figura 7, se muestra la jerarquía de los conjuntos de herramientas de TensorFlow.

Figura 7: Jerarquía de los conjuntos de herramientas de TensorFlow



Nota. Recuperado de [46].

2.7. Deep Learning

Es un subcampo de la AI que se centra en entrenar RNN para aprender y realizar tareas de procesamiento de datos. Las capas de neuronas artificiales de RNN procesan datos de forma jerárquica y progresiva. DL desempeña un papel importante en una amplia gama de aplicaciones, incluido el procesamiento de imágenes, NLP, visión artificial y reconocimiento de voz. Según el método de aprendizaje y la naturaleza de los datos de entrada, existen varios tipos de clasificación en el campo de DL. El aprendizaje de CNN está dividido en tres categorías principales: supervisado, no supervisado y por refuerzo. El aprendizaje supervisado se basa en un conjunto de datos etiquetados; en este caso, el modelo aprende a predecir la etiqueta de los datos de entrada. Sin embargo, el aprendizaje no supervisado busca patrones o agrupaciones intrínsecas en datos sin referencias previas. El aprendizaje por refuerzo se enfoca en la capacidad de tomar decisiones mediante el uso de recompensas o penalizaciones en función de las acciones realizadas por el modelo [33]. Sin embargo, su naturaleza de caja negra es uno de los desafíos de DL, ya que con frecuencia es difícil comprender cómo el modelo llega a una predicción específica [48].

A pesar de este obstáculo, DL tiene un gran potencial en aplicaciones de clasificación binaria y otras áreas, brindando nuevas oportunidades para mejorar la eficiencia y la precisión en la toma de decisiones basadas en datos. En la Figura 8, se muestra como DL clasifica de manera binaria a una entrada de datos.

Figura 8: Clasificación de una entrada de datos con DL

Nota. Recuperado de [36].

2.8. Redes neuronales recurrentes

Es una red neuronal en la que la salida del paso anterior se introduce como entrada del paso actual. En una red neuronal tradicional, todas las entradas y salidas son independientes entre sí. Sin embargo, ante la tarea de predecir la siguiente palabra de una frase, es necesario tener en cuenta las palabras anteriores y, por tanto, se requiere la capacidad de recordar información contextual. RNN satisfacen esta necesidad incorporando capas ocultas. La característica principal y más destacada de RNN es su estado oculto, que retiene información importante sobre la secuencia [49].

2.9. Transformer

Transformer es un modelo de DL que utiliza un mecanismo de autocontrol, ponderando diferencialmente la importancia de cada dato de entrada. Se utiliza principalmente en los campos de NLP y visión artificial. Al igual que las RNN, los Transformer están diseñados para procesar datos de entrada secuenciales, como los datos del lenguaje natural, y se utilizan en tareas como la traducción y el resumen de textos [31]. El modelo Transformer logra comprender la gramática del lenguaje en expresiones lingüísticas que involucran palabras concretas. Los modelos basados en Transformer han demostrado ser más eficaces en comparación con otros enfoques, como las

CNN [50]. El mecanismo de auto-atención en un modelo permite identificar la relación entre la palabra que se está analizando en un momento determinado y otras palabras presentes en la misma secuencia.

Tabla II: Comparación entre Modelos de DL y Transformers

Característica	**Modelos de Deep Learning**	**Modelos Basados en Transformer**
Arquitectura Base	DNN, CNN, RNN, LSTM, GRU, etc.	Transformer (Auto-atención)
Manejo de Secuencias	Limitado	Efectivo para secuencias largas
Atención a Contexto Global	Local	Global
Paralelismo	Limitado	Altamente paralelizable
Autoregresividad	Sí	Sí, con atención autoregresiva
Captura de Patrones	Dependiente de la arquitectura	Capaz de patrones complejos
Transferencia de Aprendizaje	Menos eficiente	Eficiente
Interpretabilidad	Menos interpretable	Más interpretable con atención
Uso Común	Imágenes, audio, texto corto	Texto, traducción, imágenes, audio

2.9.1. Modelos basados en Transformer: A lo largo del tiempo, numerosas variantes basadas en la arquitectura Transformer original han emergido, comúnmente referidas en conjunto como Transformer [51]. Estos modelos Transformer han demostrado ser superiores a las arquitecturas predecesoras en el campo del NLP como las RNN, las Redes Neuronales Recurrentes de Memoria a Corto y Largo Plazo y las Redes Neuronales Convolucionales [52]. Los modelos Transformer pueden entrenarse en grandes cantidades de datos y se adapta a tareas específicas. Entre los modelos que utiliza la arquitectura Transformer se encuentran BERT, GPT, T5 y XLNet, los cuales son objeto de este estudio.

2.10. BERT

Es un modelo basado en una red neuronal multicapa, que es derivado de un modelo entrenado en 2008 para representar el contexto, y el principal propósito es interpretar el lenguaje de búsqueda de los usuarios de una manera que sea más natural, mediante NLP [51]. Las búsquedas de Google han mejorado 10% con el modelo BERT, según resultados presentados por sus consumidores, de acuerdo con la revista The Verge y Wired [53]. BERT es la última versión del algoritmo de Google, que utiliza una red neuronal de open source para procesar el lenguaje natural

de la búsqueda de entrada. El modelo no reemplaza completamente al anterior algoritmo llamado RankBrain pero sí lo complementa y mejora sus capacidades. RankBrain es el modelo que Google implementa por primera vez AI en su motor de búsqueda para optimizar el procesamiento y la visualización de los resultados de búsqueda de los usuarios. BERT representa una importante innovación en la tecnología de búsqueda y desempeña un papel clave en la interpretación de consultas complejas, especialmente aquellas que requieren comprensión contextual o tonal. BERT mejora la capacidad del algoritmo para discernir el verdadero significado detrás de las búsquedas, proporcionando resultados más acordes con lo que los usuarios realmente buscan [54].

2.10.1. Pre-entrenamiento con doble dirección: BERT se entrena con una gran cantidad de textos no señalados, esto le proporciona la oportunidad de aprender cómo se representan las palabras y su significado se deduce de las circunstancias en torno a ella. Este adiestramiento previo en dos vías le proporciona la capacidad de entender y reproducir el ambiente de manera eficaz.

2.10.2. Preentrenamiento y afinamiento: Luego del pre-entrenamiento, BERT está apto para realizar tareas de NLP determinadas, como la clasificación de escritos, la extracción de datos, la traducción de escritos y más. Después del pre-entrenamiento, el modelo se ªfinaçon datos específicos del dominio o tarea particular. Esto permite ajustar los parámetros del modelo pre-entrenado para que se adapte mejor a las necesidades concretas de una aplicación, como la detección de sentimientos en opiniones de productos o la traducción en un campo técnico específico.

2.10.3. Comprensión contextual profunda: BERT tiene la capacidad de entender las dobles sentidos y tonalidades del idioma debido a su habilidad para tomar el contexto. Esto le proporciona una mayor capacidad para dar contestaciones de mayor precisión y en contexto.

2.11. GPT

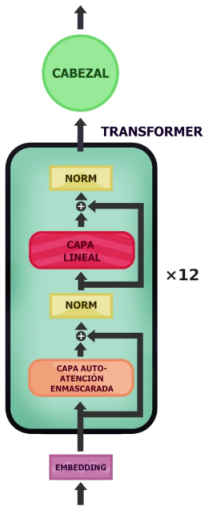
Generative Pre-trained Transformer (GPT), es una descendencia de modelos de lenguaje, desarrollada por OpenAI que se basa en la arquitectura Transformer. La notable capacidad de estos modelos para crear texto y comprender el lenguaje humano de formas singulares lo han convertido en un punto de inflexión en los campos de NLP y la AI en general [55]. Actualmente GPT es una de las grandes familias de Transformer. Entre los modelos del GPT existen GPT-1, GPT-2, GPT-3 y GPT-4.

2.11.1. GPT-1: Es el primer modelo que estableció para el método de pre-entrenamiento. Disminuyó la dependencia del entrenamiento supervisado y esto permitió generar textos cada vez más cercanos al lenguaje humano [56].

2.11.1a. Arquitectura de GPT-1:

Consta de un codificador estructurado en tres áreas distintas: El primero, ubicado en la parte inferior es el codificador de la ficha, que convierte la ficha en vectores en un espacio semántico de 768 dimensional (proceso de incrustación). Cada vector contiene un valor no sólo sobre el significado de palabra codificada, sino también en la posición del texto. El segundo son doce módulos similares con arquitectura Transformer. La última área es la cabecera del modelo, que realiza la función de cambiar la tarea que se le encargó como la clasificación de textos, predicción de palabras y entre otros [56]. En la Figura 9 se muestra la arquitectura de GPT-1.

Figura 9: Arquitectura de GPT-1.



Nota. Recuperado de [56].

2.11.2. GPT-2: Se introdujo el 5 de noviembre 2019 y recibió mucha atención. OpenAI inicialmente dudó en usarlo y gradualmente lanzó el modelo para evitar abusos. Es un modelo basado en Transformer de 1.500 millones de parámetros que logró resultados de última generación en 7 de 8 conjuntos de datos de modelado de lenguaje probados en una configuración ad hoc. Los resultados demostraron que el sistema es capaz de generar texto coherente y realizar tareas como responder preguntas, traducir automáticamente, comprender textos y resumir. Todo

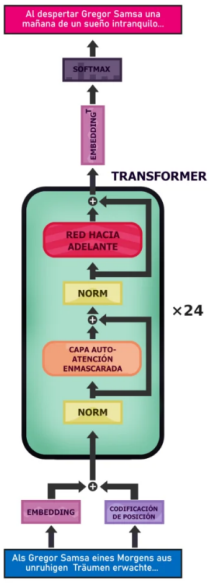
esto se logró en un entorno de çero disparos", es decir, sin necesidad de entrenamiento específico para estas tareas [57].

2.11.2a. Arquitectura de GPT-2:

La arquitectura GPT-2 consta de tres niveles de codificadores: En el primer nivel, los tokens de entrada se someten a doble codificación, posicionamiento e incrustación. GPT-2 a diferencia de lo que ocurre en GPT-1 y otros sistemas de traducción automática, ya no es una codificación léxica palabra por palabra, sino que el sistema utilizado es una codificación de pares de bytes, a medio camino entre una codificación de palabras y una codificación de caracteres/bytes. Son pares de bytes, no palabras, los que inicialmente se convierten en vectores, lo que permite al convertidor identificar características fonéticas y ortográficas de los caracteres e incluso identificar palabras con errores ortográficos.

El segundo nivel tiene 24 módulos Transformer idénticos al GPT-1. Sin embargo, en este caso las subcapas de normalización se encuentran antes de la unidad de atención y de la minired de avance (capa de líneas). En el último nivel se coloca un sistema lineal que consta de la matriz transpuesta de la matriz de entrada seguida de una función softmax [57]. El trabajo de este nivel es imprimir la cadena de salida como se muestra en la Figura 10.

Figura 10: Arquitectura de GPT-2



Nota. Recuperado de [57].

2.11.3. GPT-3: Es autorregresivo, y al igual que su predecesor, utiliza DL para generar texto similar al humano. Está abierto al público y permite el acceso a desarrolladores con proyectos prometedores o científicos que buscan constantemente mejorar sus modelos [58]. Es el modelo más grande y moderno de la línea hasta enero de 2022. Con 175 mil millones de parámetros, es extremadamente eficaz para generar texto y comprender el lenguaje humano. Puede realizar tareas de NLP como traducción, resúmenes, respuesta a preguntas, y conversaciones, entre otras.

2.11.3a. Arquitectura de GPT-3:

Es una red neuronal profunda que utiliza el mecanismo de atención para predecir la palabra que seguirá en una oración. Se ha entrenado con un corpus de más de mil millones de palabras y puede crear texto con una precisión de caracteres. Un codificador y un decodificador son los dos componentes principales de la arquitectura GPT-3. El codificador toma la primera palabra de la oración como entrada y crea una representación de vectores de la palabra, que luego se pasa a través de un mecanismo de atención para predecir la siguiente palabra. El decodificador recibe la palabra anterior y su representación de vectores como entrada, y luego crea una distribución de probabilidad de todas las palabras posibles que se pueden dar. El desempeño de GPT-3 es significativamente superior a los modelos anteriores de GPT en cuanto a la generación de texto. En un conjunto de pruebas con artículos de Wikipedia, el modelo de Turning NLG de Microsoft puede generar texto con precisión en el nivel de caracteres, pero requiere una gran cantidad de datos de entrenamiento. Según OpenAI, GPT-3 puede alcanzar este nivel de rendimiento sin información de capacitación adicional después de su período inicial de capacitación. Además, los modelos anteriores, como BERT de Google y el transformador de NLP de Stanford, no pueden producir párrafos y oraciones más largos que GPT-3 [59].

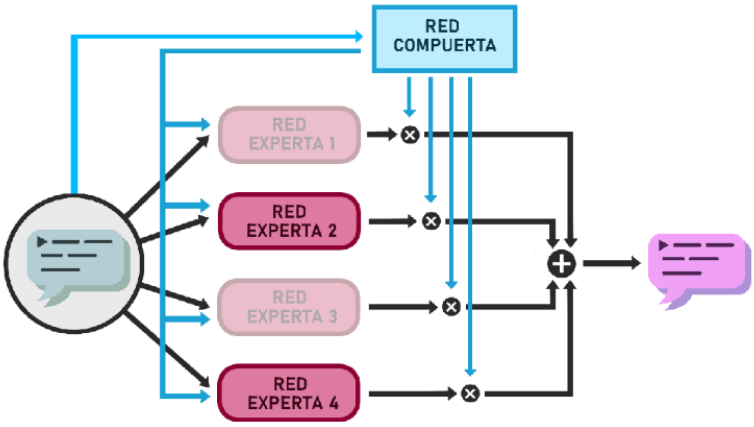
2.11.4. GPT-4: Es más creativo y colaborativo que nunca. Puedes crear, editar e iterar con los usuarios en tareas de escritura creativa y técnica, como componer canciones, escribir guiones o aprender el estilo de escritura de un usuario [60]. Bing Chat de Microsoft y la versión paga más avanzada ChatGPT, en la actualidad tienen un modelo subyacente de convertidor GPT-4, que a diferencia de su predecesor, puede aceptar no solo entrada de texto, sino también entrada de imágenes [61].

2.11.4a. Arquitectura de GPT-4:

De acuerdo con las mencionadas diseminaciones, el novedoso modelo posee nada menos que 1.800 mil millones características, en comparación a los 175 mil millones de GPT-3 (alrededor de diez veces más). Sin embargo, esto no se traduce en unas mayores exigencias de capacidad

computacional, en la medida en que GPT-4 utiliza el sistema de mezcla de expertos (Mezcla de Experts, MoE), el cual posee 16 redes paralelas, denominadas expertos, cada una de las cuales está dotada de 110 mil millones parámetros y se especializa en una labor determinada, como por ejemplo, la generación de texto, la traducción a máquina o la programación. Una especie de puertas (gate) establece por cuál de los catorce especialistas deberá pasar el conocimiento que proviene de la entrada. Los límites de las categorías de atención se elevan a 55 mil millones de parámetros, y son compartidos por todos los especialistas. Además, los números de grupos de transformación pasa de 96 a 120. En la Figura 11 se muestra la Arquitectura de GPT-4, que es un sistema de peritaje mixto, varias redes, que funcionan en paralelo, se especializan en la realización de una tarea concreta (generación de prosa, programación de código, composición de poesía, traducción automática, entre otros Un mecanismo de pasarela (red de pasarela) determina por cuál de ellas pasará la información [61].

Figura 11: Arquitectura de GPT-4.



Nota. Recuperado de [61].

2.12. T5

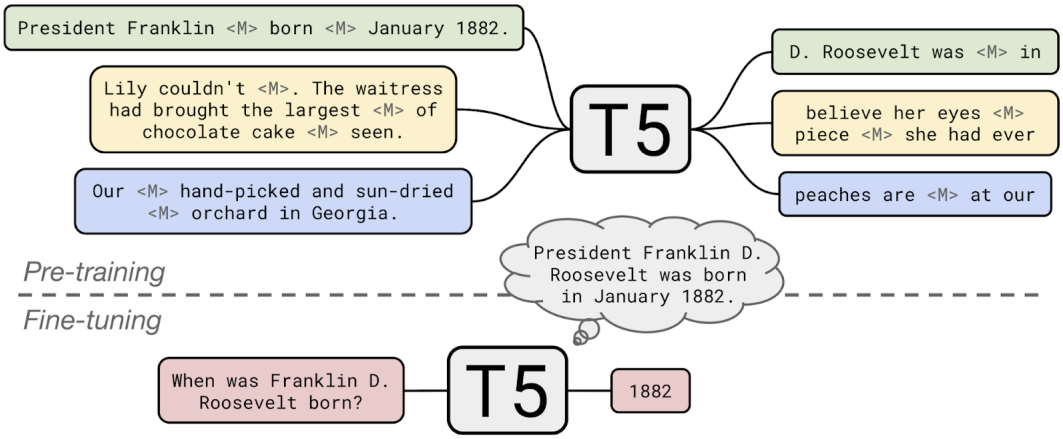
El Transformer de Transferencia de Texto a Texto (T5, por su siglas en inglés, Text-to-Text Transfer Transformer) es un modelo de lenguaje desarrollado por Google AI basado en la arquitectura Transformer, y este modelo fue introducido en el artículo académico publicado en 2019 [62]. El objetivo principal del T5 es abordar distintas tareas en el campo del NLP, formulando tanto los inputs como los outputs en un formato de texto a texto [62].

2.12.1. Unificación de Tareas de NLP: T5 aborda todas las tareas de NLP como un problema de conversión de texto a texto, donde la entrada y la salida siempre son secuencias de texto. Esto facilita el entrenamiento y permite al modelo realizar una variedad de tareas sin realizar cambios estructurales significativos [63].

2.12.2. Pre-entrenamiento y Fine-tuning: Al igual que muchos otros modelos de Transformer, T5 se entrena inicialmente en un gran corpus de texto en una tarea de modelado de lenguaje y luego se adapta a tareas específicas con conjuntos de datos más pequeños [63]. En la Figura 12, se detalla el proceso de pre-entrenamiento y ajuste fino del modelo T5.

2.12.3. Escalabilidad: T5 está disponible en una variedad de tamaños, desde muy pequeños hasta muy grandes, lo que le permite ser utilizado en entornos con una variedad de recursos computacionales [63].

Figura 12: Proceso de Pre-entrenamiento y Ajuste Fino del Modelo T5 en Tareas de NLP



Nota. Recuperado de [62].

2.13. XLNet

Es un modelo de NLP que incorpora un mecanismo de atención bidireccional generalizado y supera las limitaciones de modelos anteriores como BERT. XLNet aprovecha todas las permutaciones posibles de una secuencia de palabras, lo que le permite capturar un contexto más rico y comprender mejor las relaciones entre palabras, a diferencia de BERT, que utiliza una técnica de "mascaramiento" para predecir palabras ocultas dentro de una oración. Esto se logra mediante el uso de un método de entrenamiento autoregresivo que predice cada palabra basándose en todas

las palabras anteriores en todas las permutaciones posibles, lo que mejora significativamente la comprensión del contexto y la coherencia del texto [64].

2.13.1. Características clave de XLNet:

2.13.1a. Permutación autorregresiva:

En lugar de utilizar un modelo de lenguaje autorregresivo como GPT, que se entrena para predecir la siguiente palabra en una secuencia, XLNet utiliza una variante de preentrenamiento que permite permutaciones de palabras dentro de una secuencia de oración. Este enfoque permite capturar relaciones bidireccionales entre palabras, como BERT, manteniendo la coherencia y estructura del texto [64].

2.13.1b. Entrenamiento multidireccional:

XLNet entrena un modelo para predecir la probabilidad de una palabra en función de todas las demás palabras de la oración, independientemente de su ubicación. Esto permite capturar las relaciones entre palabras sin restricciones direccionales, lo que lo hace más poderoso en un contexto de modelado [64]. En la Figura 18 se muestra la Arquitectura de Permutaciones de Aprendizaje de XLNet.

2.13.1c. Ajuste para tareas específicas:

Al igual que BERT, XLNet se puede ajustar para tareas específicas de NLP, como clasificación de texto, extracción de información y traducción automática. El pre-entrenamiento bidireccional de XLNet lo hace eficiente para muchas tareas [64].

2.13.1d. Integración de Transformer-XL:

El modelo puede manejar secuencias de texto más largas que sus predecesores gracias a XLNet, lo que mejora significativamente la capacidad de comprender y generar texto coherente. Esta habilidad lo hace especialmente efectivo en una variedad de tareas de procesamiento natural de la lengua, como la comprensión de lectura, el resumen de textos y la traducción automática, entre otras [64].

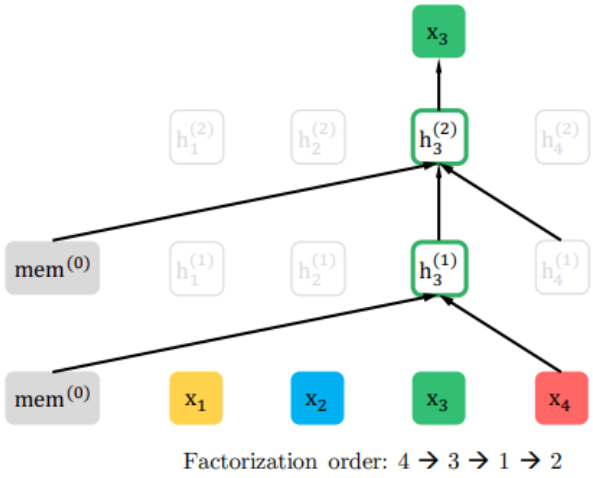
2.13.1e. Rendimiento de XLNet:

XLNet ha superado a BERT y otros modelos lingüísticos en una variedad de actividades de procesamiento del lenguaje, generando nuevos referentes de excelencia en este campo. Se ha convertido en un recurso esencial tanto para investigadores como para expertos en la materia gracias a su enfoque revolucionario y su habilidad para analizar y comprender grandes volúmenes de texto [64].

2.13.1f. Flexibilidad y aplicabilidad:

La arquitectura de XLNet es ideal para una amplia gama de aplicaciones porque puede manejar contextos extensos y comprender el lenguaje de manera más avanzada. XLNet proporciona una herramienta útil para desarrolladores y científicos de datos, que incluye la generación automática de texto y la ayuda en tareas específicas como la traducción automática, el resumen de textos y la detección de sentimientos.

Figura 13: Arquitectura de Aprendizaje de XLNet



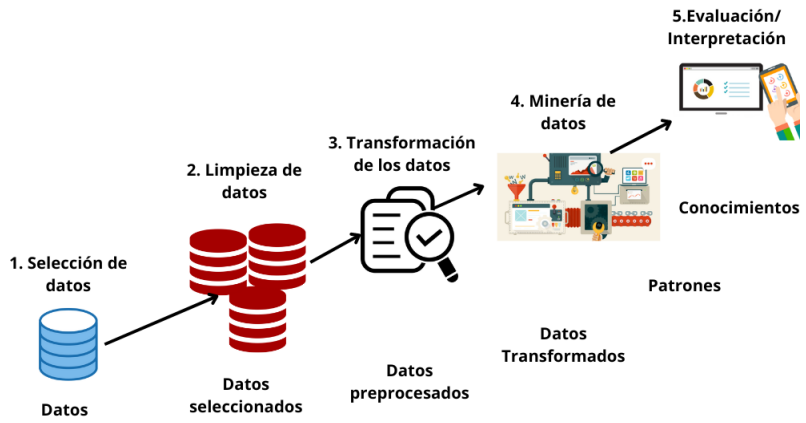
Nota. Recuperado de [64].

2.14. KDD

La metodología basada KDD es un método iterativo y cíclico de minería de datos para extraer conocimientos valiosos de grandes conjuntos de datos. Este enfoque sigue un proceso sistemático que comienza con la comprensión del problema en cuestión y luego pasa al procesamiento de los datos disponibles. El proceso KDD cubre varias etapas donde se puede obtener información valiosa que ayuda a comprender mejor los datos y respaldar la toma de decisiones informada [65]. La Figura 14 destaca estas etapas y se describe en detalle a continuación.

2.14.1. Fase de selección: Se identifican conjuntos de datos relevantes para el proceso de descubrimiento de conocimiento. Se establecen criterios para determinar qué conjuntos de datos deben incluirse en el análisis, y la selección debe realizarse cuidadosamente para garantizar que se proporcione de manera efectiva la información relevante necesaria.

Figura 14: Visión general de los pasos que componen el proceso KDD.



2.14.2. Fase de Pre-procesamiento: En la etapa de pre-procesamiento, los datos se limpian y transforman para prepararlos para el análisis. Esto implica eliminar datos irrelevantes y manejar valores faltantes, incorrectos o nulos para garantizar la integridad de los datos. El pre-procesamiento es crucial para garantizar la calidad y adaptabilidad de los datos, aumentando así la precisión y validez de los resultados.

2.14.3. Fase de Transformación: Los datos se preparan para facilitar las fases posteriores del proceso de minería de datos. Aplicar diversas técnicas para transformar los datos de la forma más adecuada. Se pueden utilizar técnicas de selección de características para identificar y retener sólo las variables más relevantes, reduciendo así la complejidad y aumentando la eficiencia.

2.14.4. Fase de Minería de datos: Se utilizan diversas técnicas de minería de datos para su análisis y exploración. Los algoritmos de ML se utilizan para descubrir patrones o relaciones en los datos, proporcionando conocimiento y comprensión. Esta tecnología se centra en explorar y analizar datos para extraer conocimientos útiles.

2.14.5. Fase de Evaluación: La calidad se analiza utilizando varias métricas de desempeño (por ejemplo, exactitud, precisión, sensibilidad, especificidad, etc.). La eficacia de un modelo de evaluación depende de su capacidad para proporcionar información útil, asegurando la calidad y validez de los resultados obtenidos.

2.14.6. Fase de Interpretación: Finalmente, en la fase de interpretación, los resultados se derivan y presentan de manera clara y comprensible para su uso efectivo en la toma de decisiones [65].

3. METODOLOGÍA/TÉCNICAS/DISEÑO

En este capítulo, se detalla la metodología empleada para llevar a cabo la investigación, centrándose en el proceso de KDD.

El proyecto se ha desarrollado utilizando la metodología KDD, una elección estratégica que concuerda de manera ideal con nuestro objetivo principal. Este objetivo se centra en la análisis y comparación de diversos modelos que utilizan la arquitectura Transformer en conjunto con NLP, con el propósito de detectar ataques de phishing en el contenido textual de dichos ataques. El método KDD y representa un proceso sistemático para extraer conocimiento significativo de grandes conjuntos de datos. Primero se seleccionan los datos relevantes, luego se preprocesan y se transforman para prepararlos para el análisis. La etapa clave de la minería de datos utiliza algoritmos para descubrir patrones y relaciones en los datos. La evaluación de patrones garantiza que los resultados sean relevantes y útiles, y que la presentación del conocimiento se realice de manera comprensible para el usuario final. La retroalimentación y el refinamiento continúan iterándose, lo que permite una mejora continua del proceso en función de las necesidades y expectativas del usuario.

3.1. Recursos

En la Tabla III, se muestra los recursos utilizados para la ejecucción de los modelos basados en la Arquitectura Transformer.

Tabla III: Recursos utilizados para el proyecto

categoría	**recurso**	**Uso**
Hardware	MSI con Intel i7 (8va generación), 16GB	Herramienta de trabajo
Sistema operativo	Windows 11 pro	Entorno de desarrollo
Lenguaje de programación	Python 3.10.2	Para ejecutar los códigos de los modelos
Librerias principales	Tensorflow, Transformers y Pandas	Librerías para los modelos Transformer.

3.2. Fase de selección

Selección del conjunto de datos: Los conjuntos de datos son fundamentales en este proceso, para obtener un conjunto de datos se vuelve imperativo. Para crear un conjunto de datos significativo, optamos por utilizar muestras de ataques de phishing como referencia. Este enfoque

proporciona análisis detallados e información sobre el comportamiento de los ataques de ciberseguridad, lo que ayuda a identificar patrones, tendencias y otros aspectos relevantes relacionados con los ataques de phishing. Para lograr esta tarea se llevaron a cabo investigaciones exhaustivas en bases científicas reconocidas como Scopus, Scielo, Redalyc.org, IEEE y WebOfScience. Una revisión de artículos identificó conjuntos de datos que involucran ataques de phishing que brindan una representación realista de las amenazas en línea.

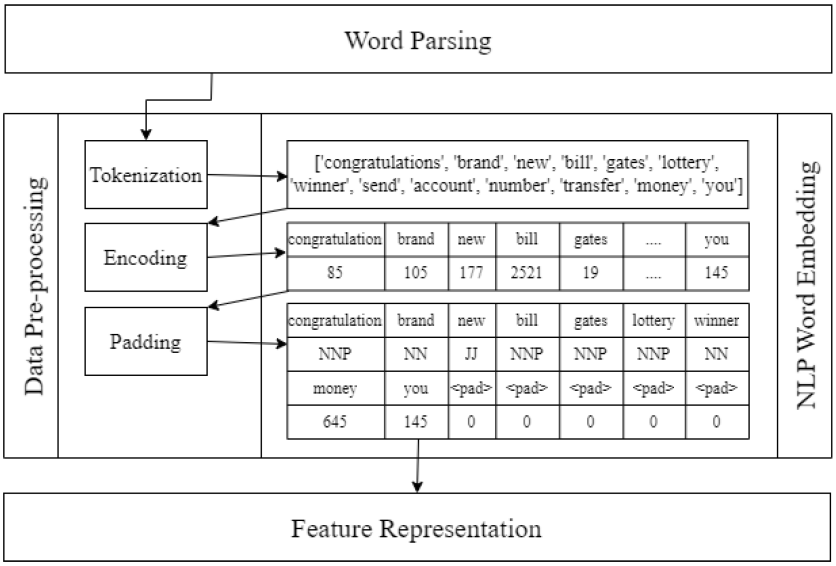
Origen del conjunto de datos: Utilizamos el conjunto de datos de libre acceso Phishload [66], que luego descomprimimos en un archivo de tipo SQL, abrimos en HeidySQL y exportamos a un formato CSV, más fácil de usar en Python. Este conjunto de datos se compone de tres tablas, de las cuales utilizamos la tabla websites, de la que extrajimos dos columnas. La columna context, que contiene el código HTML de todas las páginas web, y la columna phishing que indica si una página web es phishing o ham. Este conjunto de datos consta de 10.488 filas, pero tras eliminar las filas que contenían campos nulos, el conjunto de datos se redujo a 9761 en total, de las que al final se obtuvieron 9761 filas de phishing y 9761 filas de ham.

3.3. Fase de Preprocesamiento

Preprocesamiento de datos: Una vez capturado todo el contenido de la página web, es necesario realizar un preprocesamiento de datos para que los datos estén listos para ser incorporados en la representación de características utilizando Keras Embedding con GloVe. Para lograrlo, se realizaron tres pasos sucesivos: tokenización, codificación y relleno, como se muestra en la Figura 15. El conjunto de datos fue reducido a un tamaño de 200 palabras por entrada para optimizar recursos y tiempo de ejecución, para lo cual se implemento un script en el lenguaje de programación Python, se detallan en el Algoritmo 1.

```
import pandas as pd

# Ruta al archivo CSV
archivo_csv = 'Preprocessed_dataset.csv'

# Cargar el dataset en un DataFrame
df = pd.read_csv(archivo_csv)

# Ver las primeras filas del DataFrame para verificar su correcta carga
print(df.head())

```

Figura 15: Representación de características



Nota. Recuperado de [21].

```python
# Definir el numero maximo de palabras permitidas por entrada
max_words = 200

# Funcion para truncar las entradas de texto a un numero maximo de palabras
def truncate_text(text, limit=max_words):
    words = text.split()  # Dividir el texto en palabras
    if len(words) > limit:
        return ' '.join(words[:limit])  # Unir las primeras 'limit' palabras
    else:
        return text

# Asumiendo que la columna que quieres truncar se llama 'texto'
df['contexts'] = df['contexts'].apply(truncate_text)

# Ver las primeras filas del DataFrame modificado
print(df.head())

# Ruta al archivo CSV de salida
archivo_csv_salida = 'phishing_dataset.csv'

# Guardar el DataFrame modificado en un nuevo archivo CSV
```

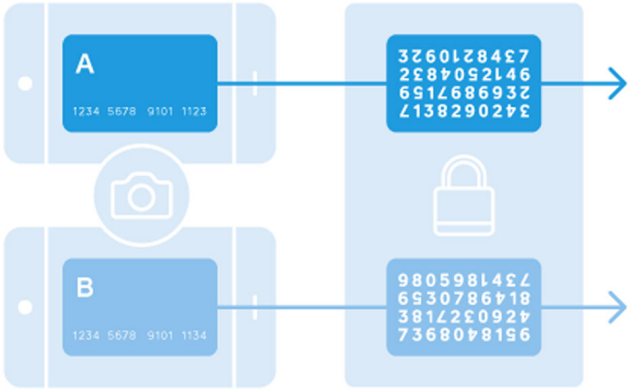
```
df.to_csv(archivo_csv_salida, index=False)

print("Dataset modificado guardado con exito.")
```

Algoritmo 1: Convertidor de palabras por entrada

3.4. Fase de Transformación

3.4.1. Tokenización: Cualquier tarea relacionada con NLP requiere la tokenización. Consiste principalmente en vectorizar cada palabra de un texto anterior, creando una secuencia de palabras con significado. Para convertir la cadena de entrada en una secuencia de enteros, hemos utilizado el método de tokenización texts_to_sequences. En la Figura 16, muestra el proceso de tokenización.

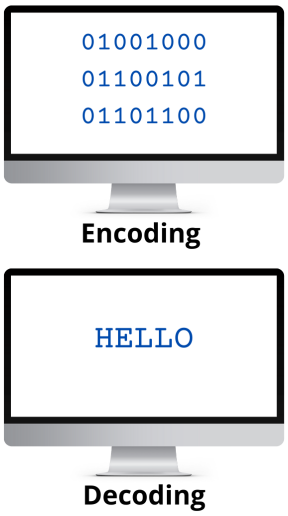
Figura 16: Proceso de tokenización



Nota. Recuperado de [67].

3.4.2. Encoding: La función texts_to_sequences se utiliza para codificar las palabras de una cadena de texto, que asigna un número entero a cada palabra a medida que aparece. Como resultado, convertimos los datos de texto en un formato numérico que los modelos DL pueden procesar (como se mira en la Figura 17).

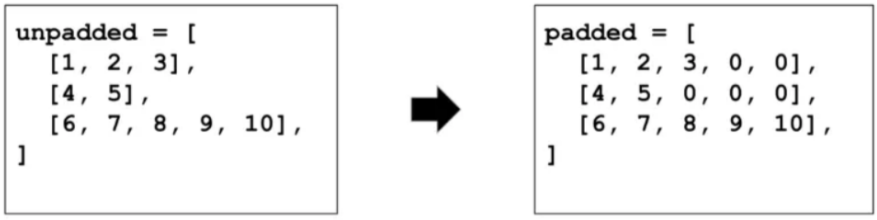
Figura 17: Funcionamiento del encoding



Nota. Recuperado de [68].

3.4.3. Padding: La longitud máxima maxlen que debe tener la cadena introducida en el algoritmo DL se determina mediante el relleno. Esta cadena se truncará hasta que la longitud de las palabras de la página web sea mayor que el máximo. Por otro lado, si la longitud de la página web es menor que el máximo, los espacios sin palabras se rellenarán con una etiqueta PAD hasta que se complete el máximo. En la Figura 18, muestra el funcionamiento del padding en el preprocesamiento de texto.

Figura 18: Funcionamiento del padding



Nota. Recuperado de [69].

3.4.4. Representación de características con Keras Embedding y GloVe: Keras Embedding Layer with pre-trained GloVe funciona mapeando cada palabra del flujo de entrada a una representación vectorial previamente entrenada, que se aprende basándose en las características distribucionales de las palabras en un corpus de texto significativo. El objetivo principal de nuestro modelo es determinar la importancia semántica y sintáctica del texto de las páginas web antes de que los algoritmos DL lo analicen.

3.5. Fase de Minería de datos

3.5.1. Ejecución de algoritmos Transformer: Se implementaron y entrenaron cuatros modelos Transformer (GPT, T5, BERT y XLnet) para el elegir el mejor para la detección de phishing. Seleccionamos BERT como el modelo más adecuado para nuestra aplicación después de evaluar minuciosamente entre los demás modelo, este mostró mejor resultado. Para la implementación de los modelos Transformer, se lo desarrollaron en script con el lenguaje Python, los cuales se detallan en los Algoritmos 2, 3, 4 y 5.

```
#Importacion de las librerias
import shutil
import pandas as pd
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
from official.nlp import optimization
# para crear el optimizador AdamW
from sklearn.model_selection import train_test_split

#Cargar dataset
# Ruta al archivo CSV
file_path = 'phishing_dataset.csv'

# Cargar el dataset
df = pd.read_csv(file_path)

# Dividir el dataset en conjuntos de entrenamiento, validacion y prueba
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
train_df, val_df = train_test_split(train_df, test_size=0.2, random_state=42)

# Constantes para el entrenamiento
AUTOTUNE = tf.data.AUTOTUNE
```

```
batch_size = 32
seed = 42
epochs = 3

# Crear datasets de TensorFlow
def convert_df_to_tf_dataset(df, shuffle=True, batch_size=32):
    ds = tf.data.Dataset.from_tensor_slices((df['contexts'].values, df['phishing'].
    values))
    if shuffle:
        ds = ds.shuffle(buffer_size=len(df))
    ds = ds.batch(batch_size)
    ds = ds.cache().prefetch(buffer_size=AUTOTUNE)
    return ds

train_ds = convert_df_to_tf_dataset(train_df, shuffle=True, batch_size=batch_size)
val_ds = convert_df_to_tf_dataset(val_df, shuffle=False, batch_size=batch_size)
test_ds = convert_df_to_tf_dataset(test_df, shuffle=False, batch_size=batch_size)
# Construir el modelo
def build_classifier_model():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing'
    )
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder'
    )
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']
    net = tf.keras.layers.Dropout(0.1)(net)
    net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)
    return tf.keras.Model(text_input, net)

model = build_classifier_model()
# Compilar el modelo
loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
metrics = tf.metrics.BinaryAccuracy()

epochs = 5
steps_per_epoch = tf.data.experimental.cardinality(train_ds).numpy()
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
```

```
optimizer = optimization.create_optimizer(init_lr=init_lr, num_train_steps=
    num_train_steps, num_warmup_steps=num_warmup_steps, optimizer_type='adamw')

model.compile(optimizer=optimizer, loss=loss, metrics=[metrics])

# Entrenar el modelo
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)

model.evaluate(val_ds)
```

Algoritmo 2: BERT

```
#Librerias
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
from transformers import XLNetTokenizer, TFXLNetModel
from tensorflow.keras.optimizers import Adam
# Cargar el conjunto de datos
df = pd.read_csv('phishing_dataset.csv')
df

#Dividir el dataset
train, test = train_test_split(df, test_size=0.2, random_state=42)

#Construir el modelo
xlnet_model = 'xlnet-base-cased'
tokenizer = XLNetTokenizer.from_pretrained(xlnet_model)

def encode_reviews(tokenizer, reviews, max_length=120):
    return tokenizer(reviews, padding='max_length', truncation=True, max_length=
    max_length, return_tensors='tf')

train_encodings = encode_reviews(tokenizer, train['contexts'].tolist())
test_encodings = encode_reviews(tokenizer, test['contexts'].tolist())
def create_model_xlnet(xlnet_model):
    word_inputs = tf.keras.Input(shape=(120,), dtype='int32', name='word_inputs')
    #Ajusta el tamano segun tu configuracion
    xlnet = TFXLNetModel.from_pretrained(xlnet_model)
    xlnet_encodings = xlnet(word_inputs)[0]
    doc_encoding = tf.squeeze(xlnet_encodings[:, -1:, :], axis=1)
```

```
    doc_encoding = tf.keras.layers.Dropout(0.1)(doc_encoding)
    outputs = tf.keras.layers.Dense(1, activation='sigmoid', name='outputs')(
    doc_encoding)
    model = tf.keras.Model(inputs=[word_inputs], outputs=[outputs])
    model.compile(optimizer=Adam(learning_rate=2e-5), loss='binary_crossentropy',
    metrics=['accuracy'])
    return model
train_dataset = tf.data.Dataset.from_tensor_slices(({
    'word_inputs': train_encodings['input_ids']
}, train['phishing'].values)).batch(32)

test_dataset = tf.data.Dataset.from_tensor_slices(({
    'word_inputs': test_encodings['input_ids']
}, test['phishing'].values)).batch(32)

#compilar el modelo
model = create_model_xlnet(xlnet_model)
history = model.fit(train_dataset, validation_data=test_dataset, epochs=5)

#Evaluamos el modelo
model.evaluate(test_dataset)
```

Algoritmo 3: XLNet

Como trabajo adicional, GPT tiene dos modelos entre ellos davinci y babbage que son para clasificar texto. Por lo cual se realizó una comparación entre estos modelos. En comparación con babbage, davinci se utiliza para conjuntos de datos más grandes y tareas de mayor complejidad; se observó un mayor consumo de tokens, lo que resultó en un costo mayor. Para lo cual se implementó un script en el lenguaje de programación Python, se detallan los Algoritmos 4, 5.

```
import pandas as pd
import openai

# Configura tu clave de API de OpenAI
openai.api_key = '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'

# Funcion para clasificar un texto usando GPT-3.5 Turbo
def clasificar_texto_con_gpt(texto):
    response = openai.Completion.create(
        model="babbage-002",
        prompt=f": '{texto}'. Es phishing o no?",
        temperature=0,
```

```
        max_tokens=60,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0
    )
    respuesta = response.choices[0].text.strip().lower()
    # Mapea la respuesta a un valor binario
    if 'phishing' in respuesta:
        return 1
    else:
        return 0

# Carga el dataset
df = pd.read_csv('phishing_dataset.csv')

# Clasificar cada texto en el DataFrame (esto puede tardar dependiendo del tamano del dataset)
df['clasificacion_gpt'] = df['contexts'].apply(clasificar_texto_con_gpt)

from sklearn.metrics import accuracy_score

# Asumiendo que 'phsihing' es la columna con las etiquetas verdaderas
y_true = df['phishing']
y_pred = df['clasificacion_gpt']

# Calcula el accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy}")
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Divide el dataset en entrenamiento y prueba
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

# Clasifica los textos para el conjunto de entrenamiento
train_df['clasificacion_gpt'] = train_df['contexts'].apply(clasificar_texto_con_gpt)

# Clasifica los textos para el conjunto de prueba
test_df['clasificacion_gpt'] = test_df['contexts'].apply(clasificar_texto_con_gpt)

# Calcula la precision para el conjunto de entrenamiento
train_accuracy = accuracy_score(train_df['phishing'], train_df['clasificacion_gpt'])
```

```
54
55 # Calcula la precision para el conjunto de prueba
56 test_accuracy = accuracy_score(test_df['phishing'], test_df['clasificacion_gpt'])
57
58 # Calcula la matriz de confusion para obtener mas informacion sobre el rendimiento
59 train_confusion = confusion_matrix(train_df['phishing'], train_df['clasificacion_gpt'
       ])
60 test_confusion = confusion_matrix(test_df['phishing'], test_df['clasificacion_gpt'])
61
62 print(f"Training Accuracy: {train_accuracy}")
63 print(f"Test Accuracy: {test_accuracy}")
64 print(f"Training Confusion Matrix:\n{train_confusion}")
65 print(f"Test Confusion Matrix:\n{test_confusion}")
```

Algoritmo 4: GPT-davinci

```
1  import pandas as pd
2  import openai
3
4  # Configura tu clave de API de OpenAI
5  openai.api_key = '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'
6
7  # Funcion para clasificar un texto usando GPT-3.5 Turbo
8  def clasificar_texto_con_gpt(texto):
9      response = openai.Completion.create(
10         model="babbage-002",
11         prompt=f": '{texto}'. Es phishing o no?",
12         temperature=0,
13         max_tokens=60,
14         top_p=1,
15         frequency_penalty=0,
16         presence_penalty=0
17     )
18     respuesta = response.choices[0].text.strip().lower()
19     # Mapea la respuesta a un valor binario
20     if 'phishing' in respuesta:
21         return 1
22     else:
23         return 0
24
25 # Carga el dataset
26 df = pd.read_csv('phishing_dataset.csv')
```

```
27
28 # Clasificar cada texto en el DataFrame (esto puede tardar dependiendo del tamano del
       dataset)
29 df['clasificacion_gpt'] = df['contexts'].apply(clasificar_texto_con_gpt)
30
31 from sklearn.metrics import accuracy_score
32
33 # Asumiendo que 'phsihing' es la columna con las etiquetas verdaderas
34 y_true = df['phishing']
35 y_pred = df['clasificacion_gpt']
36
37 # Calcula el accuracy
38 accuracy = accuracy_score(y_true, y_pred)
39 print(f"Accuracy: {accuracy}")
40 from sklearn.model_selection import train_test_split
41 from sklearn.metrics import accuracy_score, confusion_matrix
42
43 # Divide el dataset en entrenamiento y prueba
44 train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
45
46 # Clasifica los textos para el conjunto de entrenamiento
47 train_df['clasificacion_gpt'] = train_df['contexts'].apply(clasificar_texto_con_gpt)
48
49 # Clasifica los textos para el conjunto de prueba
50 test_df['clasificacion_gpt'] = test_df['contexts'].apply(clasificar_texto_con_gpt)
51
52 # Calcula la precision para el conjunto de entrenamiento
53 train_accuracy = accuracy_score(train_df['phishing'], train_df['clasificacion_gpt'])
54
55 # Calcula la precision para el conjunto de prueba
56 test_accuracy = accuracy_score(test_df['phishing'], test_df['clasificacion_gpt'])
57
58 # Calcula la matriz de confusion para obtener mas informacion sobre el rendimiento
59 train_confusion = confusion_matrix(train_df['phishing'], train_df['clasificacion_gpt'
       ])
60 test_confusion = confusion_matrix(test_df['phishing'], test_df['clasificacion_gpt'])
61
62 print(f"Training Accuracy: {train_accuracy}")
63 print(f"Test Accuracy: {test_accuracy}")
64 print(f"Training Confusion Matrix:\n{train_confusion}")
```

```
print(f"Test Confusion Matrix:\n{test_confusion}")
```

Algoritmo 5: GPT-babbage

```
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import T5Tokenizer, TFT5ForConditionalGeneration
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy

# Load the dataset
df = pd.read_csv('phishing_dataset.csv')

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['contexts'], df['phishing'],
    test_size=0.2, stratify=df['phishing'])

# Tokenize the data using the T5 tokenizer
tokenizer = T5Tokenizer.from_pretrained('t5-small')
# Tokenize the data using the T5 tokenizer with truncation and padding
train_encodings = tokenizer(list(X_train), truncation=True, padding="max_length",
    max_length=512, return_tensors='tf')
test_encodings = tokenizer(list(X_test), truncation=True, padding="max_length",
    max_length=512, return_tensors='tf')
import tensorflow as tf

train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    y_train
)).shuffle(1000).batch(32)

test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    y_test
)).batch(32)


from transformers import TFT5ForConditionalGeneration

```

```
class T5ClassificationModel(tf.keras.Model):
    def __init__(self, model_name, num_labels):
        super().__init__()
        self.t5 = TFT5ForConditionalGeneration.from_pretrained(model_name)
        self.classifier = tf.keras.layers.Dense(num_labels, activation='softmax')

    def call(self, input_ids, attention_mask=None):
        outputs = self.t5(input_ids=input_ids, attention_mask=attention_mask,
    decoder_input_ids=input_ids)
        hidden_states = outputs.encoder_last_hidden_state[:, 0, :]
        logits = self.classifier(hidden_states)
        return logits

model = T5ClassificationModel('t5-small', num_labels=2)
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_dataset, validation_data=test_dataset, epochs=5)
model.evaluate(test_dataset)
```

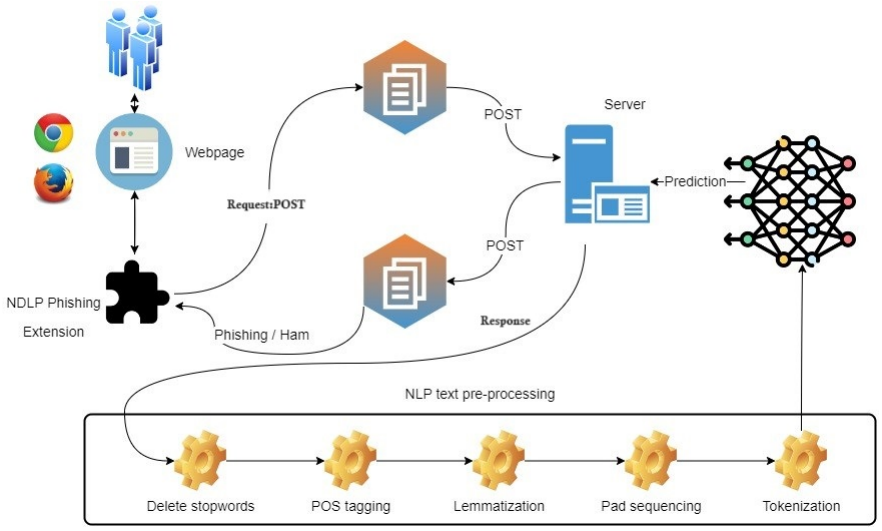
Algoritmo 6: T5

3.6. Desarrollo de la extensión de navegador

La extensión creada envía los datos del contenido textual de las páginas web visitadas a un servidor, donde se procesa y compara con el modelo BERT que se seleccionó previamente. Los resultados de la predicción se devuelven al navegador y muestran usuario como un porcentaje de probabilidad de amenaza que sea phishing. En el Algoritmo 7, se implementa el script del servidor flask. En la Figura 19, se muestra la arquitectura del aplicativo.

```
from flask import Flask, request
from numpy import array
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Activation, Dropout, Dense
from tensorflow.keras.layers import Flatten, LSTM, GRU, Bidirectional,
    GlobalAveragePooling1D
from tensorflow.keras.layers import GlobalMaxPooling1D, SpatialDropout1D,
    GlobalMaxPool1D, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Embedding
```

Figura 19: Arquitectura del aplicativo




```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Concatenate
from nltk.stem.wordnet import WordNetLemmatizer
from nltk import pos_tag
from nltk.corpus import wordnet
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
import nltk
import pandas as pd
import numpy as np
import re

nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')

model = load_model('C:/Users/Bryan/codigo/Codigo_Transformer/Aplicativo/BERT_model.h5'
    )

app = Flask(__name__)

```

```
@app.route('/predict', methods=['POST'])
def areceibe_text():
    webpage_text  = [clean_text(request.get_data(as_text=True))]

    tokenizer = Tokenizer(num_words=3000)
    tokenizer.fit_on_texts(webpage_text)
    webpage_text = tokenizer.texts_to_sequences(webpage_text)
    vocab_size = len(tokenizer.word_index) + 1
    webpage_text = pad_sequences(webpage_text, padding='post', maxlen=200)

    print(webpage_text)

    results = model.predict(webpage_text)
    results= results[0][0]
    results = round(results,3)
    print (results)

    return str(results)
    #if results[0][0] > results[0][1]:
    #    return "Pishing"
    #else:
    #    return "Normal"


#Lemmatize Words

def fetch_pos_tag(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        # As default pos in lemmatization is Noun
        return wordnet.NOUN


lemmatizer = WordNetLemmatizer()

```

```
#cleaning the data now

regex = [
    r'<[^>]+>', #HTML tags
    r'@(\w+)', # @-mentions
    r"#(\w+)", # hashtags
    r'http[s]?://(?:[a-z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-f][0-9a-f]))+', # URLs
    r'[^0-9a-z #+_\\r\\n\\t]', #BAD SYMBOLS
]

REPLACE_URLS = re.compile(r'http[s]?://(?:[a-z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-f
    ][0-9a-f]))+')
REPLACE_HASH = re.compile(r'#(\w+)')
REPLACE_AT = re.compile(r'@(\w+)')
REPLACE_HTML_TAGS = re.compile(r'<[^>]+>')
REPLACE_DIGITS = re.compile(r'\d+')
#REPLACE_BY = re.compile(r"[/(){}\[\]\|,;.:?\-\'\"$]")
REPLACE_BY = re.compile(r"[^a-z0-9\-]")

STOPWORDS = set(stopwords.words('english'))

#tokens_re = re.compile(r'('+'|'.join(regex)+')', re.VERBOSE | re.IGNORECASE)

# sentences = [] #for Word2Vec model

def clean_text(text,**args):
    text = text.lower()
    text = REPLACE_HTML_TAGS.sub(' ', text)
    text = REPLACE_URLS.sub('', text)
    text = REPLACE_HASH.sub('', text)
    text = REPLACE_AT.sub('', text)
    text = REPLACE_DIGITS.sub(' ', text)
    text = REPLACE_BY.sub(' ', text)
    text = " ".join(lemmatizer.lemmatize(word.strip(), fetch_pos_tag(pos_tag([word.
    strip()])[0][1])) for word in text.split() if word not in STOPWORDS and len(word)
    >3)

    #sentences.append(text.split())
    return text


```

```
114 if __name__ == '__main__':
115     app.run(host='0.0.0.0', port=5000, debug=True)
```

Algoritmo 7: Servidor de la extensión

3.6.1. Estructura del aplicativo: La creación del archivo manifest.json fue el primer paso en el desarrollo de la extensión. Este archivo es el motor de la extensión y define todos los recursos necesarios para que funcione. Los detalles como la versión de la extensión, los permisos necesarios y los archivos de script que se ejecutarán se encuentran en este archivo. En el Algoritmo 8 se puede observar el script del archivo manifest.json. Posteriormente, se creó el archivo HTML (como se mira en el Algoritmo 10) que forma la interfaz gráfica de la extensión. Esta contiene todos los elementos gráficos como campos de texto y botones, para que el usuario interactué de una forma mas amigable. El funcionamiento de la extensión se implementó a través de varios archivos JavaScript, como background.js (como se mira en el Algoritmo **??**) que su función es realizar peticiones al servidor. Este script escucha los eventos del botón HTML y, cuando se activa, envía una petición al servidor. El servidor devuelve una alerta con el porcentaje de phishing de la página web. El content.js es para chrome envié el contenido de la pagina web. El script de content.js se puede mirar en el Algoritmo 11.

```
1  {
2    "manifest_version": 2,
3    "name": "NDLP Phishing",
4    "version": "2.0",
5    "description": "Check if it's a phishing attempt.",
6    "permissions": [
7      "activeTab",
8      "http://127.0.0.1:5000/*",
9      "notifications",
10     "declarativeContent",
11     "storage",
12     "webNavigation",
13     "tabs"
14   ],
15   "background": {
16     "scripts": ["background.js"],
17     "persistent": true
18   },
19   "browser_action": {
20     "default_popup": "popup.html"
```

```
  },
  "icons": {
    "48": "icon.png"
  },
  "content_security_policy": "script-src 'self' https://example.com; object-src 'self'
    ",
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": [
        "content.js",
        "lib/jquery/dist/jquery.min.js",
        "lib/bootstrap/dist/js/bootstrap.min.js"
      ]
    }
  ],
  "web_accessible_resources": [
    "sweetalert2.all.min.js",
    "sweetalert2.min.css"
  ]
}
```

Algoritmo 8: Manifest.json

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NDLP Phishing</title>


  <style>
    body {
      width: 480px;
      margin: 20px auto;
      padding: 10px;
      background: #2b2b2b;
      text-align: center;
    }

```

```
.container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 350px;
}

.button-container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(48%, 1fr));
  grid-gap: 10px;
}

.rounded-circle {
  /* background: linear-gradient(45deg, #00db2f, #06678b); */
  background: linear-gradient(45deg, #23572e, #06678b);
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.2), 0 1px 2px rgba(0, 0, 0, 0.24);
  transform: translateZ(25px);


  width: 10rem;
  height: 10rem;
  padding: 10px;
  clear: both;
  align-content: center;
  margin: 0 auto;

}

html {
  height: 490px;
}

.button-85:after {
  z-index: -1;
  content: "";
  position: absolute;
  width: 100%;
  height: 100%;
  background: linear-gradient(45deg, #300073e1, #0b9797);
  left: 0;
```

```
61      top: 0;
62      border-radius: 50%;
63    }
64
65    h1 {
66      font-family: 'Russo One', sans-serif;
67      color: white;
68      font-size: 25px;
69      padding: 10px;
70      text-align: center;
71    }
72
73    h1 {
74      margin-top: 0;
75    }
76
77    .button-85 {
78      padding: 0em 1em;
79      justify-content: baseline;
80      border-radius: 50%;
81      margin-right: 24px;
82      margin-left: 25px;
83      border: none;
84      outline: none;
85      color: white;
86      background: linear-gradient(45deg, #00db2f, #06678b);
87      cursor: pointer;
88      position: relative;
89      z-index: 0;
90      user-select: none;
91      -webkit-user-select: none;
92      touch-action: manipulation;
93      font-size: 20px;
94      padding: 15px 30px;
95    }
96
97    .button-85:before {
98      content: "";
99      background: linear-gradient(45deg,
100         black);
101     position: absolute;
102
```

```
103       top: -2px;
104       left: -2px;
105       background-size: 500%;
106       z-index: -1;
107       filter: blur(5px);
108       -webkit-filter: blur(5px);
109       width: calc(100% + 4px);
110       height: calc(100% + 4px);
111       animation: glowing-button-85 20s linear infinite;
112       transition: opacity 0.3s ease-in-out;
113       border-radius: 50%;
114     }
115
116     #sendButton {
117       font-family: 'Montserrat', sans-serif;
118       font-weight: 500;
119       background: linear-gradient(45deg, #00db2f, #06678b);
120       box-shadow: 0 1px 3px rgba(0, 0, 0, 0.2), 0 1px 2px rgba(0, 0, 0, 0.24);
121       transform: translateZ(25px);
122       width: 10rem;
123       height: 10rem;
124       padding: 10px;
125       clear: both;
126       align-content: center;
127       margin: 0 auto;
128       background-size: 500%;
129       color: white;
130       border-radius: 50%;
131     }
132     #siteStatus {
133       font-weight: bold;
134     }
135     footer {
136       margin-top: 30px;
137       font-size: 0.8rem;
138     }
139     #siteStatus{
140       color: #00db2f;
141     }
142     p{
143       color: #95d2e9;
144     }
```

```
145
146
147   </style>
148 </head>
149   <div class="container">
150     <h1>Detector de Phishing</h1>
151     <div id="status">
152       <p>Estado: <span id="siteStatus">Analizando...</span></p>
153     </div>
154     <div id="rounded-circle">
155       <button class="button-85" id="sendButton">Comprobar Sitio</button>
156     </div>
157     <footer>
158       <p>Reportar un error</p>
159     </footer>
160     <script src="popup.js"></script>
161   </div>
162 </body>
163 </html>
```

Algoritmo 9: Archivo popup.html

```
1  chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
2      if (request.action === "sendContent") {
3        var bodyContent = request.body;
4        sendContentToServer(bodyContent);
5      }
6    });
7
8      function sendContentToServer(content) {
9          fetch('http://127.0.0.1:5000/predict', {
10           method: 'POST',
11           headers: {
12             'Content-Type': 'text/plain'
13           },
14           body: content
15         })
16         .then(response => {
17           if (response.ok) {
18               return response.text();
19
20           } else {
```

```
            console.log('Failed to send information to the server.');
          }
        })

        .then(result => {

            var resultNum = parseFloat(result);

            if (!isNaN(resultNum)) {
              var resultMult = resultNum * 100;
              var resultString = resultMult.toFixed(2); // Convierte a string con dos decimales
              alert('This page has a ' + resultString + '% probability of being Phishing.');
          }
        })
        .catch(error => {
                console.log('Error in the request:', error);
        });
      }
```

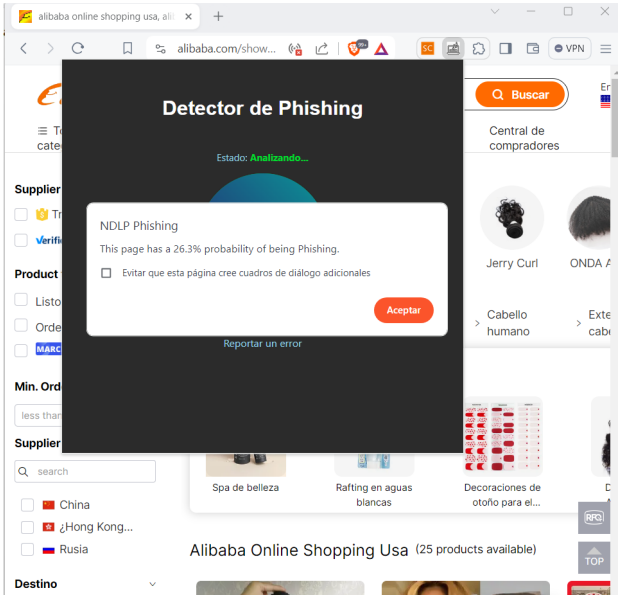
Algoritmo 10: background.js

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
    if (request.action === "getTextFromPage") {
      var bodyContent = document.body.innerText;
      chrome.runtime.sendMessage({ action: "sendContent", body: bodyContent });
    }
  });
```

Algoritmo 11: Content.js

3.6.2. Funcionamiento del aplicativo: Primero se abre la página web, y después se da click en la extensión. La extensión tiene una interfaz, que muestra un botón llamado çomprobar sitio", que la función del botón es escanear el contenido de la página web. El funcionamiento de la extensión es analizar la probabilidad de phishing del sitio web escaneado, esta envía la probabilidad en forma de alerta en el sitio web. En la Figura 20, muestra el funcionamiento la extensión.

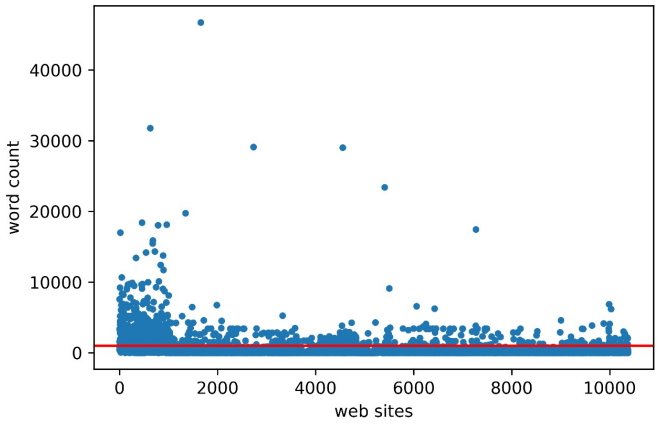
Figura 20: Funcionamiento de la extensión



3.7. *Fase de Evaluación*

3.7.1. Análisis del número de palabras para ingresar en el algoritmo: Es fundamental determinar la longitud del texto a introducir en los algoritmos, para lo cual se analizó el conjunto de datos. Por lo tanto, con la finalidad de mejorar y optimizar el rendimiento del sistema, en el análisis de NLP se limitó la longitud del conjunto de datos a 200 palabras. En la Figura 21, se dibuja una línea roja, correspondiente a la media de la longitud L= 200 palabras por sitio web.

Figura 21: Distribución de la longitud de las palabras en todo el conjunto de datos.



Nota. Recuperado de [70].

3.7.2. Implementación de Modelos: Cada uno de los modelos Transformer recibe entrenamiento y validación a través de un conjunto de datos estandarizado, lo que garantiza que los resultados sean comparables.

3.7.3. Evaluación de Costo-Beneficio: Es transcendental realizar una evaluación de costo-beneficio entre modelos utilizados en el proyecto, para saber cual nos beneficia y es mas eficiente al momento de clasificar texto. Se realiza una tabla comparativa sobre los modelos Transformer con su costo en utilizar su modelo, costo en infraestructura y sus beneficios.

3.7.4. Recolección de Métricas: En este estudio, es importante sacar las métricas para medir el rendimiento de los algoritmos. Lo que se quiere es tomar el mejor modelo, por cual se debe sacar las métricas de cada modelo. Dentro de las métricas tenemos accuracy, loss, val_accuracy y val_loss, esta se evalúan a lo largo del entrenamiento y la validación de cada modelo para supervisar su desempeño y convergencia. Después del entrenamiento se calculan las métricas de precisión, F1-score, y Recall. Para determinar el modelo más eficiente, se compararán las métricas de rendimiento mediante una tabla.

3.7.5. Análisis de sobreajuste y subajuste: Se pueden encontrar casos de sobreajuste o subajuste, que indican la necesidad de ajustar la complejidad del modelo o el proceso de entrenamiento, comparando la precisión de entrenamiento con la precisión de prueba. Esto se lo representa mediante gráficos en la sección de resultados.

3.7.6. Evaluación de rendimiento de los modelos: Para evaluar la eficacia de los modelos GPT, T5, XLNet y BERT. Se evalúan los modelos mediante la curva ROC, que permite medir el rendimiento de cada modelo.

4. RESULTADOS

En este capítulo, se muestran los resultados obtenidos de la investigación, principalmente en el análisis y comparación de diversos modelos Transformer combinados con técnicas de NLP para detectar ataques de phishing en el contenido textual. La metodología de KDD permitió la extracción de grandes conjuntos de datos, evaluando la efectividad de cada modelo en la detección de phishing.

4.1. Fase de Interpretación

En Tabla IV, se muestra la comparación de los modelos Transformer sobre su rendimiento utilizados dentro de nuestro estudio.

Tabla IV: Comparativa de Rendimiento de Modelos Transformer

Modelo	**Costo de Acceso**	**Costo de Infraestructura**	**Costo de Operación**	**Observaciones**
GPT	Pago	Alto	Moderado	Requiere GPUs
BERT	Gratis	Moderado	Bajo	Optimizable
T5	Gratis	Alto	Moderado	Alta precisión
XLNet	Gratis	Alto	Moderado	Buen rendimiento en tareas específicas

4.1.0a. Evaluación del Consumo de Tokens:

Es necesario entender que OpenAI cobra por el uso de estos modelos, y esta a su vez cobra en base al número de tokens procesados, considera el token de entrada, entrenamiento y salida generada por el modelo creado en Python. Davinci es utilizado para conjuntos de datos más grandes y tareas de mayor complejidad, lo que resultó en un mayor uso de tokens. Esto se reflejó en un aumento en los costos en comparación con Babbage, que se utilizó para tareas más precisas. En la Tabla V, muestra los costos por la utilización de los tokens. En la Figura 22, se representa de manera gráfica los números tokens utilizados en los submodelos de GPT para la detección de phishing. De acuerdo con los datos presentados en la Figura 23, el análisis del conjunto de tokens resultó en un gasto total de $25.65 dólares.

Tabla V: Costos de los modelos por la utilización de los tokens

Modelo	Uso
davinci-002	$2.00 / 1M tokens
babbage-002	$0.40 / 1M tokens

Figura 22: Comparación del Uso Mensual de los Modelos de GPT Babbage y Davinci: Solicitudes de API y Tokens Procesados

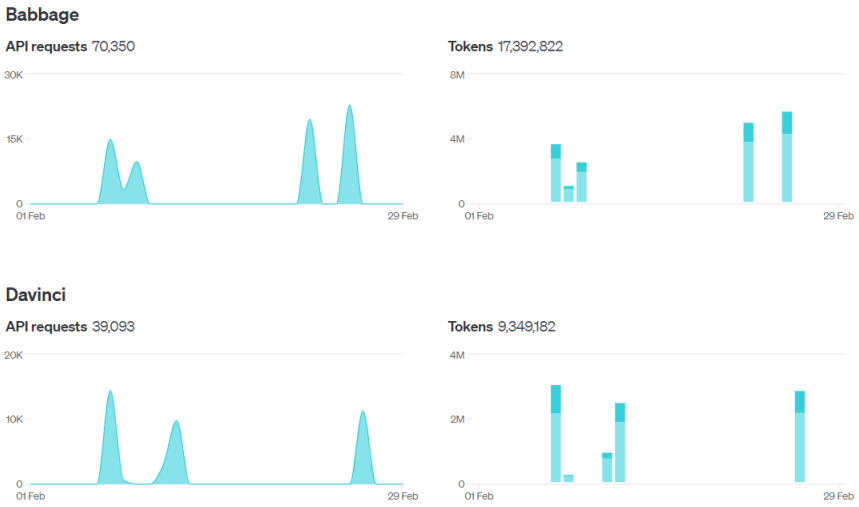
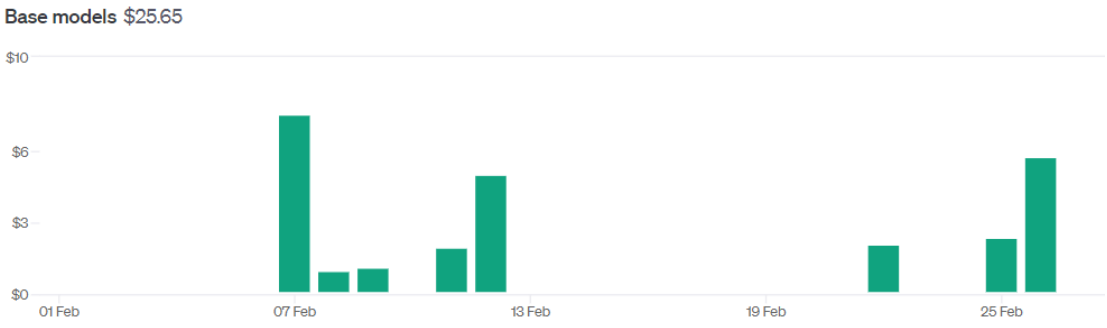


Figura 23: Gastos de los modelos de GPT utilizados en nuestro estudio



En la Tabla VI, se muestra el contexto de optimización obtenidas de los modelos Transformer como el acurracy, loss, val_loss y val_accuracy. Se observó que BERT y XLnet siempre dieron mejor precisión de prueba que los otros algoritmos. T5 mostró valores similares a BERT y XLnet; sin embargo, GPT dio los peores resultados para clasificar texto. GPT es un modelo que genera texto y afina texto, por lo que no se le puede añadir epochs, ni evaluar las métricas.

Tabla VI: loss, acurracy, val_loss y val_accuracy con L = 200 y con 5 epochs

Algoritmo	**L**	**epoch**	**loss**	**accuracy**	**val-loss**	**val-accuracy**	**Tiempo de ejecución**
BERT	200	5	0.08	0.97	0.19	0.93	1 hora y 19 minutos
GPT	200	N/A	N/A	0.81	N/A	0.81	1 hora
T5	200	5	0.19	0.92	0.18	0.92	2 horas 68 minutos
XLNet	200	5	0.16	0.93	0.19	0.92	10 horas 30 minutos

En la Tabla VII, se muestra la métricas de los modelos Transformer implementados. Sirven para evaluar el rendimiento de los modelos.

Tabla VII: Métricas de los modelos Transformer

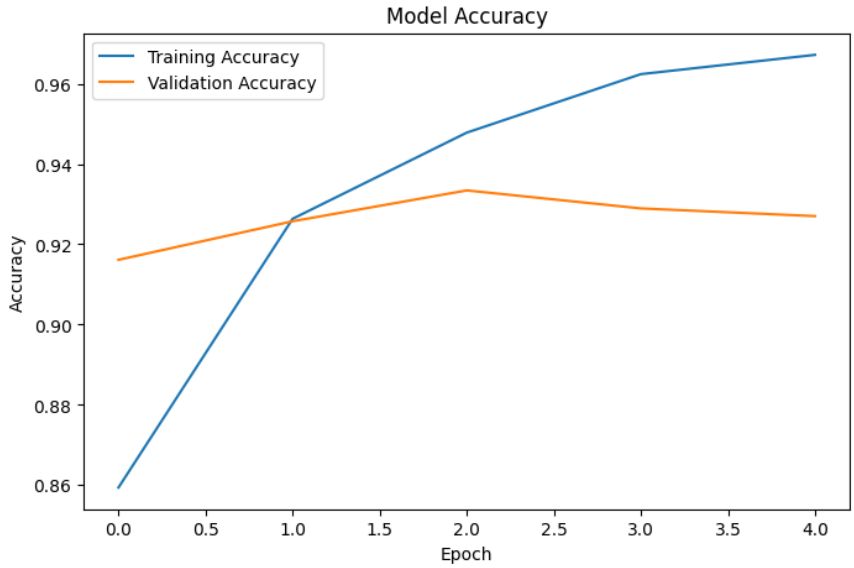
Algoritmo	**Precisión**	**Recall**	**F1 score**
BERT	0.96	0.95	0.96
GPT	0.86	0.91	0.89
T5	0.94	0.96	0.95
XLNet	0.95	0.97	0.96

En las Figuras 14, 15, y 16, se presentan los análisis gráficos de la ejecución de los cuatros modelos. En este sentido, los valores de las precisiones de entrenamiento obtenidos en cada modelo se acercaban a los de la validación.

4.2. NLP-BERT

Como se puede observar en la Figura 24, que el acurracy es de 97.15% y la de validación 93.34% obtenidas con BERT fueron aceptables. Nuevamente, se puede comprobarse que la línea de precisión de la validación se aproximó mucho a la precisión de entrenamiento. De los cuatros modelos DL, BERT fue el modelo que mejores métricas dio como resultado.

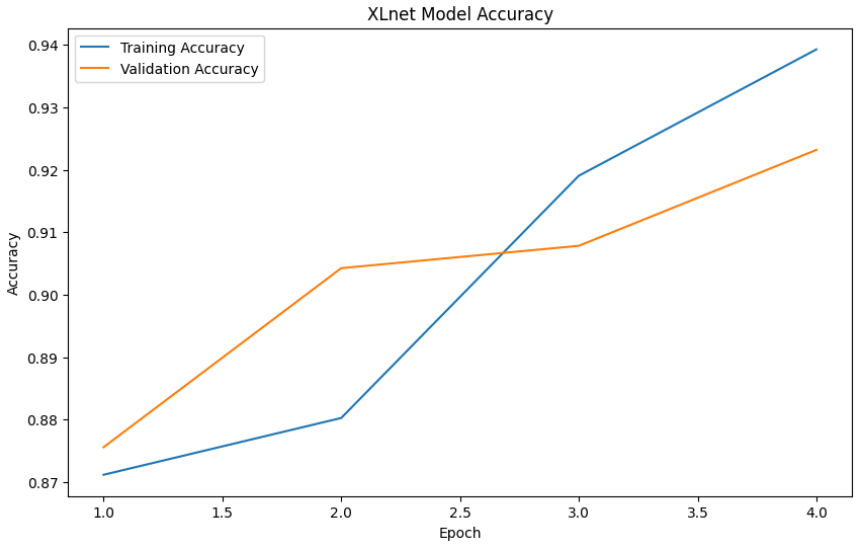
Figura 24: Sobreajuste y Subajuste de BERT



4.3. NLP-XLNet

Como se puede observar en la Figura 25, la precisión de prueba 93.93% y la de validación 92.32% obtenidas con XLNet fueron aceptables. Nuevamente, se puede comprobarse que la línea de precisión de la validación se aproximó mucho a la precisión de entrenamiento.

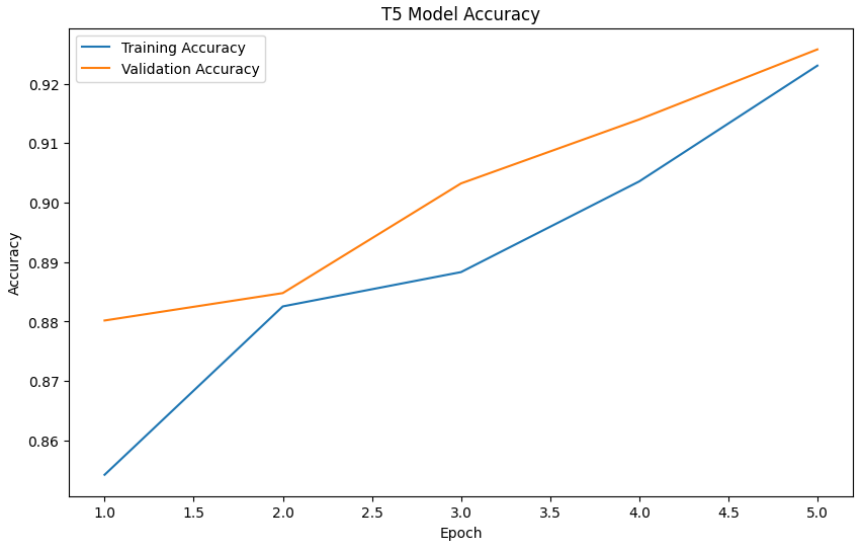
Figura 25: Sobreajuste y Subajuste de XLNet



4.4. NLP-T5

Como se puede observar en la Figura 26, la precisión de prueba 92.30% y la de validación 92.58% obtenidas con T5. En el cual se puede comprobar que la línea de precisión de la validación se aproximó mucho a la precisión de entrenamiento.

Figura 26: Sobreajuste y Subajuste de T5



4.5. NLP-GPT

Se analizó dos modelos de GPT que son para clasificar texto, donde se tomó el que mejor resultado obtuvo. En la Tabla VIII muestra la precisión de los modelos babbage y davinci.

Tabla VIII: Acurracy de los modelos de GPT babbage y davinci comparados

Modelo	**Acurracy**	**Tiempo de ejecución**
davinci-002	51.35 %	90 minutos
babbage-002	80.76%	60 minutos

En las Figuras 27, 28, 29 y 30, se muestra la matriz de confusión de los cuatros modelos Transformer utilizados en el estudio comparativo de detección de phishing. Estas matrices muestran detalladamente la precisión de clasificación de cada modelo en relación con las clases objetivo, lo que permite una evaluación completa de sus capacidades de predicción en el conjunto de datos de prueba.

Figura 27: Matriz de confusión del modelo BERT

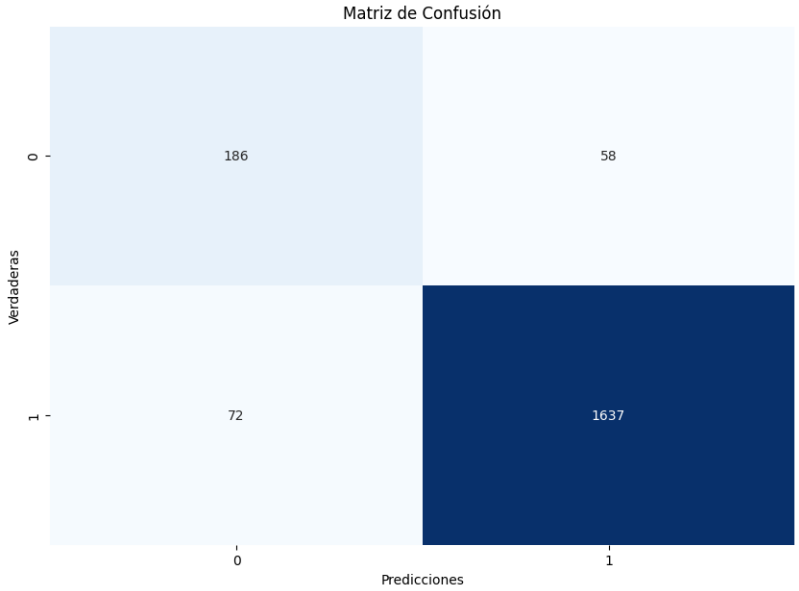


Figura 28: Matriz de confusión del modelo XlNet

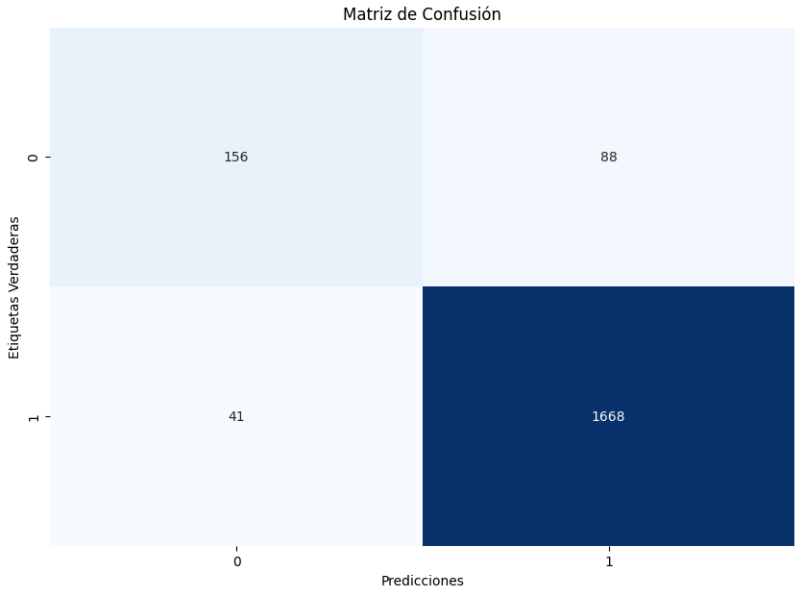


Figura 29: Matriz de confusión del modelo T5

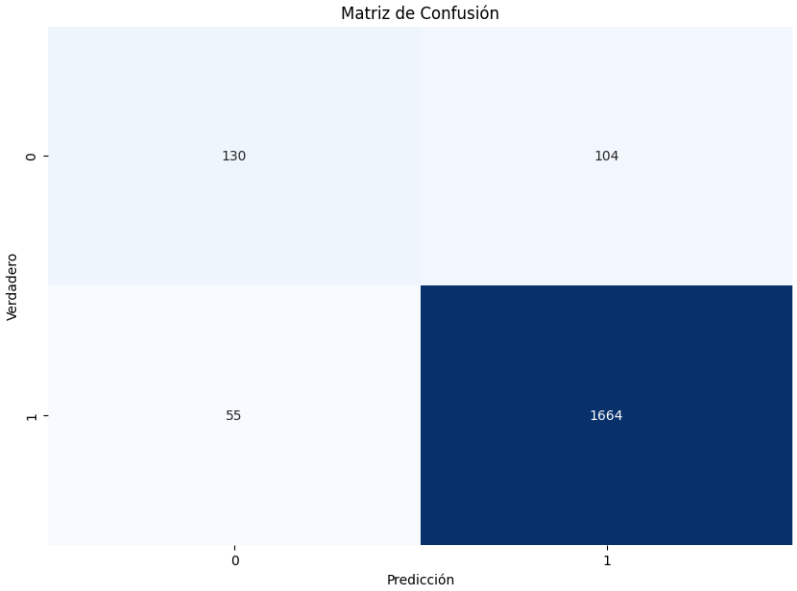
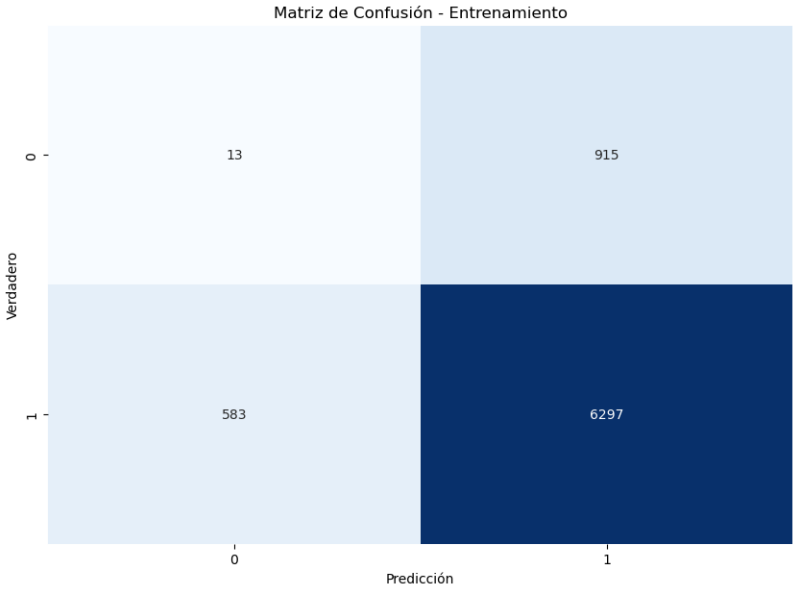


Figura 30: Matriz de confusión del modelo GPT



La curva de ROC en los modelos Transformer de DL, permite analizar su rendimiento en tareas de clasificación. En la Figura 31, 32, 33 y 34, muestra la relación entre la tasa de verdaderos positivos (sensibilidad) y la tasa de falsos positivos (especificidad) para diferentes puntos de corte o umbrales de clasificación. Para los modelos evaluados, los cuatro gráficos de la Curva ROC muestran un rendimiento de clasificación de los modelos T5, BERT y XLNet, como buena. Sin embargo en el caso de GPT está debajo de la línea con un AUC de 0.85, y los demás modelos cuenta con valores de AUC de 0.92 a 0.96, lo que indica una buena la diferencia entre clases positivas y negativas.Los modelos BERT, T5 y XLNet demuestra ser eficaz en la tarea de clasificación, a pesar de las pequeñas variaciones en el AUC, y aquel con el AUC de 0.96 es el más preciso.

Figura 31: Curva ROC BERT

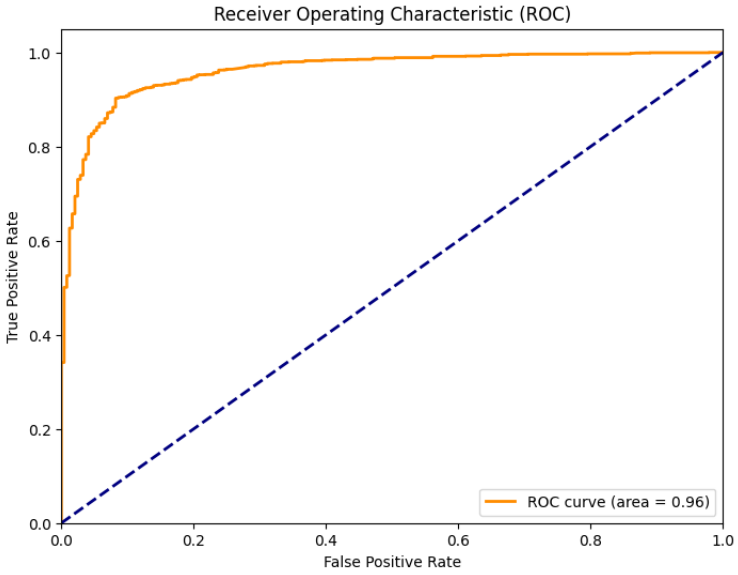


Figura 32: Curva ROC XLNet

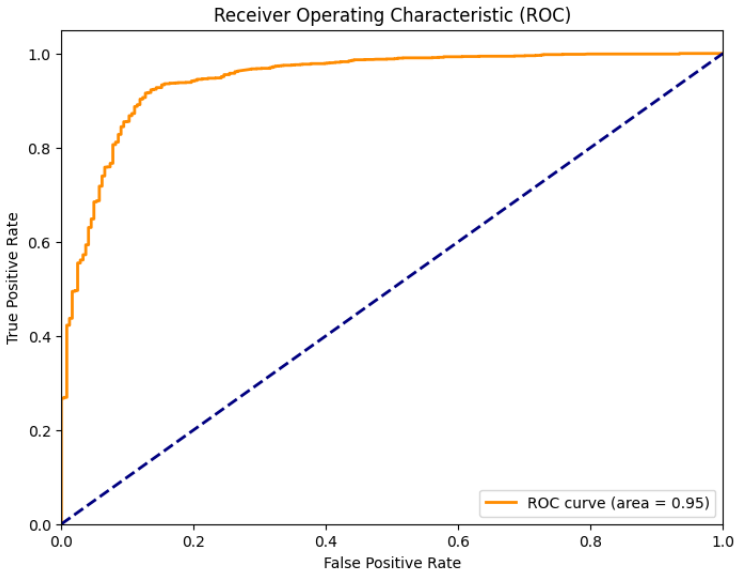


Figura 33: Curva ROC T5

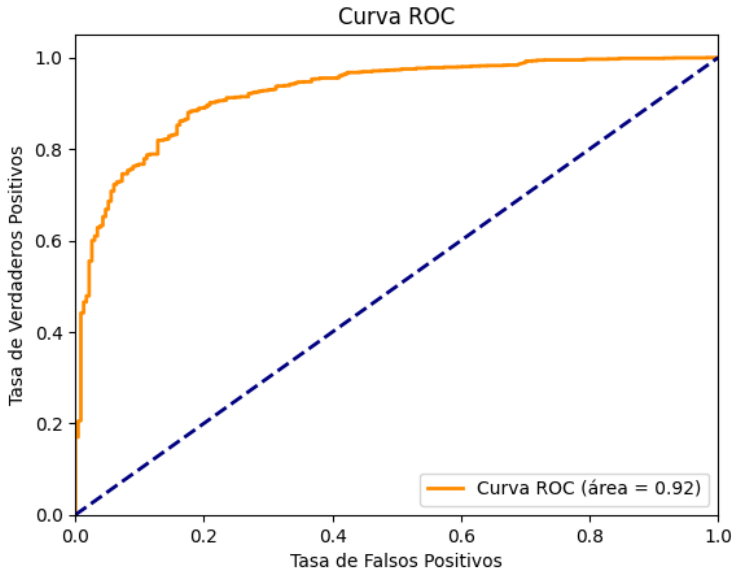
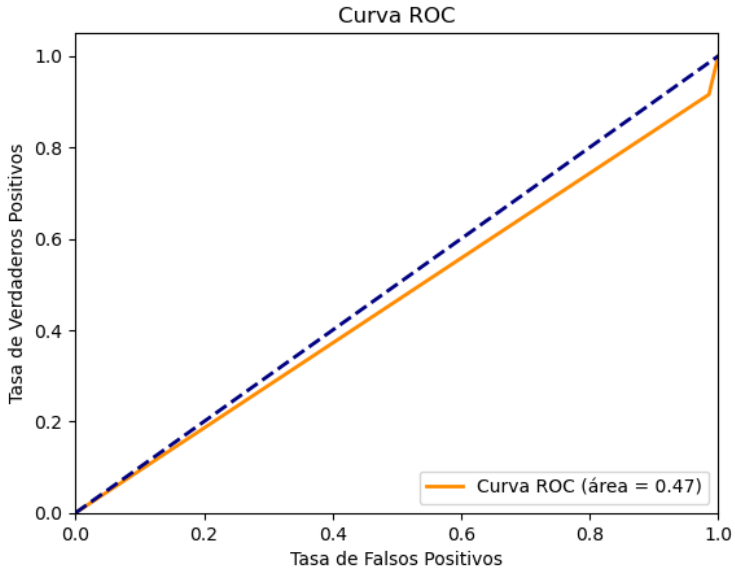


Figura 34: Curva ROC GPT



5. Conclusiones y recomendaciones

5.1. Conclusiones

En este trabajo se realizó un análisis comparativo de los modelos basados en la arquitectura Transformer GPT, T5, BERT y XLNet, para la detección de paginas web de phishing. Según nuestro estudio, el modelo BERT obtuvo un acurracy del 93.34%, en un segundo lugar XLnet obtuvo un 92.88%, T5 obtuvo un acurracy 92.58% y como último lugar GPT obtuvo un acurracy 80.76%.

Un aporte significativo de nuestro estudio en comparación con otros trabajos del estado del arte analizados, es la capacidad para detectar phishing mediante el análisis del contenido textual de las páginas web. Para esto, primeramente se hizo un pre procesamiento mediante NLP al texto HTML, resultando en un texto limpio sin código HTML. Luego se usó la técnica de word embedding con GloVe para incrustar de palabras, y aprovechar así las conexiones semánticas entre ellas.

Usamos dentro del modelo GPT la versión 3.5, debido a su avanzada capacidad para interpretar y analizar el significado subyacente de los textos de manera más profunda y precisa. Dentro de GPT-3.5 se realizó una comparación entre modelos como babbage-002, davinci-002, que son modelos para clasificar textos. Davinci es para clasificación texto que requieren análisis de sentimientos complejos. Babbage es utilizado más para las tareas especificas de clasificación de textos definidas por etiquetas.

Además de los modelos que se evaluaron, también se creó una extensión de navegador web para detectar ataques de phishing en base al algoritmo BERT, que fue el algoritmo mejor evaluado entre los cuatros modelos analizados. La extensión proporciona a los usuarios una herramienta fácil de usar y efectiva para la seguridad en línea utilizando las últimas tecnologías de detección.

5.2. Trabajo futuro

Para futuras mejoras, se contempla ampliar las capacidades de nuestra aplicación, incorporando una función que permita a los usuarios ingresar URLs directamente para su análisis, mejorando así su versatilidad y facilidad de uso. Esta adición promovería una detección proactiva de phishing, permitiendo evaluar la seguridad de sitios web antes de acceder a ellos.

5.3. Recomendaciones

Debido a la complejidad y la demanda de recursos de los modelos Transformer, la recomendación principal es disponer de una computadora altamente equipada.

Limitar el tamaño de los conjuntos de datos a un máximo de 200 palabras por entrada es una recomendación práctica para optimizar el uso de recursos y tiempo en el entrenamiento y ejecución de modelos Transformer, especialmente en tareas de detección de phishing. Esta técnica permite un procesamiento más eficiente sin comprometer significativamente la efectividad o la precisión de la detección. Se puede mantener un equilibrio entre el rendimiento y los recursos enfocándose en segmentos de texto más concisos, lo que facilita una implementación más ágil y sostenible.

Realizar evaluaciones regulares del modelo para garantizar su eficacia y precisión, y modificarlo según sea necesario para adaptarlo a las nuevas tendencias y estrategias de phishing.

Se recomienda la utilización de Python 3.10.2 para facilitar el desarrollo efectivo de algoritmos utilizando modelos Transformer de DL. La compatibilidad comprobada con TensorFlow, que es esencial para las operaciones de clasificación de texto, respalda esta recomendación. Durante el transcurso de este proyecto, se descubrió que otras versiones de Python tenían incompatibilidades significativas con TensorFlow.

6. REFERENCIAS

[1] E. Benavides-Astudillo, N. Tipan-Guerrero, G. Castillo-Zambrano et al., "A framework based on personality traits to identify vulnerabilities to social engineering attacks", en *International Conference on Applied Technologies*, Springer, 2021, págs. 381-394.

[2] M. Macas, C. Wu y W. Fuertes, "A survey on deep learning for cybersecurity: Progress, challenges, and opportunities", *Computer Networks*, vol. 212, pág. 109 032, 2022.

[3] "APWG — Phishing Activity Trends Reports". (), dirección: https://apwg.org/trendsreports/ (visitado 25-02-2024).

[4] S. Balasubaramanian, P. Ganesan y J. Rajasekaran, "Weighted ensemble classifier for malicious link detection using natural language processing", *International Journal of Pervasive Computing and Communications*, 2023.

[5] E. E. Lastdrager, "Achieving a consensual definition of phishing based on a systematic review of the literature", *Crime Science*, vol. 3, nº 1, págs. 1-10, 2014.

[6] S. Bagui, D. Nandi, S. Bagui y R. J. White, "Machine learning and deep learning for phishing email classification using one-hot encoding", *Journal of Computer Science*, vol. 17, págs. 610-623, 2021.

[7] T. Sutter, A. S. Bozkir, B. Gehring y P. Berlich, "Avoiding the hook: influential factors of phishing awareness training on click-rates and a data-driven approach to predict email difficulty perception", *IEEE Access*, vol. 10, págs. 100 540-100 565, 2022.

[8] X. Zhang, Y. Zeng, X.-B. Jin, Z.-W. Yan y G.-G. Geng, "Boosting the phishing detection performance by semantic analysis", en *2017 ieee international conference on big data (big data)*, IEEE, 2017, págs. 1063-1070.

[9] K. S. Mandapati, S. Meesala, D. Maddela, K. Ponnada, H. Neyyala y E. A. Shaik, "A Hybrid Transformer Ensemble Approach for Phishing Website Detection", en *2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, IEEE, 2023, págs. 1-8.

[10] B. Abdeen, E. Al-Shaer y W. Shadid, "VeriActor: Dynamic Generation of Challenge-Response Questions for Enhanced Email Sender Verification", en *2023 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2023, págs. 1-9.

[11] A. Almutairi, B. Kang y N. Fadhel, "The Effectiveness of Transformer-Based Models for BEC Attack Detection", en *International Conference on Network and System Security*, Springer, 2023, págs. 77-90.

[12] M. Elsadig, A. O. Ibrahim, S. Basheer et al., "Intelligent Deep Machine Learning Cyber Phishing URL Detection Based on BERT Features Extraction", *Electronics*, vol. 11, nº 22, pág. 3647, 2022.

[13] E. S. Aung y H. Yamana, "Segmentation-based phishing URL detection", en *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2021, págs. 550-556.

[14] P. Bountakas, K. Koutroumpouchos y C. Xenakis, "A comparison of natural language processing and machine learning methods for phishing email detection", en *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021, págs. 1-12.

[15] P. Maneriker, J. W. Stokes, E. G. Lazo, D. Carutasu, F. Tajaddodianfar y A. Gururajan, "URLTran: Improving phishing URL detection using transformers", en *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, IEEE, 2021, págs. 197-204.

[16] M. Pundir, J. K. Sandhu et al., "Spam Email Detection using Deep Learning Techniques", en *2023 IEEE North Karnataka Subsection Flagship International Conference (NKCon)*, IEEE, 2023, págs. 1-6.

[17] Q. Yaseen et al., "Spam email detection using deep learning techniques", *Procedia Computer Science*, vol. 184, págs. 853-858, 2021.

[18] S. Atawneh y H. Aljehani, "Phishing Email Detection Model Using Deep Learning", *Electronics*, vol. 12, nº 20, pág. 4261, 2023.

[19] M. Salavarría, J. Alberto y K. J. Monteros González, "Detección de ataques de Phishing utilizando Procesamiento de Lenguaje Natural y Modelo Oculto de Markov", 2022.

[20] J. Jiménez, "Tipos de ataques de ingeniería social y cómo evitarlos", ago. de 2023. dirección: https://www.redeszone.net/tutoriales/seguridad/tipos-ataques-ingenieria-social-consejos/.

[21] E. Benavides, W. Fuertes, S. Sanchez y D. Nuñez-Agurto, "Caracterización de los ataques de phishing y técnicas para mitigarlos. Ataques: una revisión sistemática de la literatura", *Ciencia y Tecnología*, vol. 13, nº 1, págs. 97-104, 2020.

[22] P. L. Indrasiri, M. N. Halgamuge y A. Mohammad, "Robust ensemble machine learning model for filtering phishing URLs: Expandable random gradient stacked voting classifier (ERG-SVC)", *IEEE Access*, vol. 9, págs. 150 142-150 161, 2021.

[23] P. G. Llorente. "Los 8 tipos de ataque phishing más utilizados". (16 de oct. de 2023), dirección: https://sellolegal.com/blog/los-8-tipos-de-ataque-phishing-mas-utilizados/.

[24] B. M. Losada, "Procesamiento de lenguaje natural para adquisición de conocimiento: aproximaciones desde la ingeniería de requisitos", *QUID: Investigación, Ciencia y Tecnología*, nº 24, págs. 69-78, 2015.

[25] A. C. Vásquez, J. P. Quispe, A. M. Huayna et al., "Procesamiento de lenguaje natural", *Revista de investigación de Sistemas e Informática*, vol. 6, nº 2, págs. 45-54, 2009.

[26] M. Vicente, C. Barros, F. S. Peregrino, F. Agulló y E. Lloret, "La generación de lenguaje natural: análisis del estado actual", *Computación y Sistemas*, vol. 19, nº 4, págs. 721-756, 2015.

[27] I. G. Leiva y J. V. R. Muñoz, "El procesamiento del lenguajes natural aplicado al análisis del contenido de los documentos", *Revista general de Información y Documentación*, vol. 6, nº 2, págs. 205-218, 1996.

[28] L. Rouhiainen, "Inteligencia artificial", *Madrid: Alienta Editorial*, 2018.

[29] S. J. Russell y P. Norvig, *Artificial intelligence a modern approach*. London, 2010.

[30] N. Beltrán y E. Rodríguez Mojica, *Procesamiento del lenguaje natural (PLN)-GPT-3.: Aplicación en la Ingeniería de Software. Tecnología Investigación y Academia, 8 (1), 18–37*, 2021.

[31] S. Karita, N. Chen, T. Hayashi et al., "A comparative study on transformer vs rnn in speech applications", en *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, IEEE, 2019, págs. 449-456.

[32] P. Therapeutics y P. Therapeutics, *AI: ¿Por qué es el futuro de la medicina reproductiva? — Pronacera showing RD results*, feb. de 2023. dirección: https://www.pronacera.com/es/ai-por-que-es-el-futuro-de-la-medicina-reproductiva/.

[33] L. A. B. Benavides y L. F. M. Fernandez, "Machine Learning", *Revista Sistemas*, 2022. dirección: https://api.semanticscholar.org/CorpusID:255049852.

[34] C. Janiesch, P. Zschech y K. Heinrich, "Machine learning and deep learning", *Electronic Markets*, vol. 31, págs. 685-695, 2021. dirección: https://api.semanticscholar.org/CorpusID:233210772.

[35] Z. Keita, *Classification in Machine Learning: An Introduction*, sep. de 2022. dirección: https://www.datacamp.com/blog/classification-machine-learning.

[36] E. Sperling, *Deep learning spreads*, feb. de 2018. dirección: https://semiengineering.com/deep-learning-spreads/.

[37] L. E. C. Bravo, H. J. F. lopez y E. R. Trujilllo, "Análisis del rendimiento académico mediante técnicas de aprendizaje automático con métodos de ensamble", *Revista Boletín Redipe*, 2022. dirección: https://api.semanticscholar.org/CorpusID:248041031.

[38] M. Quiroa, *Aprendizaje no supervisado*, jun. de 2023. dirección: https://economipedia.com/definiciones/aprendizaje-no-supervisado.html.

[39] K. Pykes, *Introduction to Unsupervised Learning*, ene. de 2024. dirección: https://www.datacamp.com/blog/introduction-to-unsupervised-learning.

[40] A. R. A. Notari, L. M. G. Raffi, J. M. C. Rodríguez y E. A. S. Pérez, "Enseñanza del aprendizaje por refuerzo con un sencillo ejemplo de minimización de funciones", *In-Red 2023 - IX Congreso Nacional de Innovación Educativa y Docencia en Red*, 2023. dirección: https://api.semanticscholar.org/CorpusID:263807314.

[41] J. E. Sierra-García y M. Santos, "Redes neuronales y aprendizaje por refuerzo en el control de turbinas eólicas", *Revista Iberoamericana de Automática e Informática industrial*, 2021. dirección: https://api.semanticscholar.org/CorpusID:244209214.

[42] B. De Ceupe, *Ceupe*, abr. de 2022. dirección: https://www.ceupe.com/blog/aprendizaje-por-refuerzo.html.

[43] E. A. Schab, M. F. Piccoli y C. A. C. Pietroboni, "Analítica Prescriptiva en VRP mediante Aprendizaje por Refuerzo y Flujos de Eventos", *AJEA*, 2022. dirección: https://api.semanticscholar.org/CorpusID:252708536.

[44] M. J. I. Cornejo, "Aplicación de aprendizaje por refuerzo para el estacionamiento automático de un automóvil en un ambiente simulado", *Actas del Congreso Internacional de Ingeniería de Sistemas*, 2021. dirección: https://api.semanticscholar.org/CorpusID:247823290.

[45] M. Hoijtink y A. Planqué-van Hardeveld, "Machine learning and the platformization of the military: A study of google's machine learning platform TensorFlow", *International Political Sociology*, vol. 16, nº 2, olab036, 2022.

[46] *Introduction to TensorFlow*. dirección: https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit.

[47] M. Abadi, M. Isard y D. G. Murray, "A computational model for TensorFlow: an introduction", en *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, ép. MAPL 2017, Barcelona, Spain: Association for Computing Machinery, 2017, págs. 1-7, ISBN: 9781450350716. DOI: 10.1145/3088525.3088527. dirección: https://doi.org/10.1145/3088525.3088527.

[48] I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions", *Sn Computer Science*, vol. 2, 2021. dirección: https://api.semanticscholar.org/CorpusID:237241776.

[49] Z. Ghazanfar, *How Transformers work in Deep Learning and NLP: an intuitive introduction?*, ene. de 2023. dirección: https://www.linkedin.com/pulse/how-transformers-work-deep-learning-nlp-intuitive-zoya-ghazanfar/.

[50] J. Gehring, M. Auli, D. Grangier, D. Yarats e Y. N. Dauphin, "Convolutional sequence to sequence learning", en *International conference on machine learning*, PMLR, 2017, págs. 1243-1252.

[51] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", en *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran y T. Solorio, eds., Minneapolis, Minnesota: Association for Computational Linguistics, jun. de 2019, págs. 4171-4186. DOI: 10.18653/v1/N19-1423. dirección: https://aclanthology.org/N19-1423.

[52] T. Wolf, L. Debut, V. Sanh et al., "Transformers: State-of-the-art natural language processing", en *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, págs. 38-45.

[53] P. Nayak, "Understanding searches better than ever before", oct. de 2019. dirección: https://blog.google/products/search/search-language-understanding-bert/.

[54] D. De Llanos, *¿Qué es BERT y cómo funciona? - DXMedia*, jun. de 2020. dirección: https://dxmedia.net/algoritmo-bert-google/.

[55] A. Hendy, M. Abdelrehim, A. Sharaf et al., "How good are gpt models at machine translation? a comprehensive evaluation", *arXiv preprint arXiv:2302.09210*, 2023.

[56] A. Radford y K. Narasimhan, "Improving Language Understanding by Generative Pre-Training", 2018. dirección: https://api.semanticscholar.org/CorpusID:49313245.

[57] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever et al., "Language models are unsupervised multitask learners", *OpenAI blog*, vol. 1, nº 8, pág. 9, 2019.

[58] L. Floridi y M. Chiriatti, "GPT-3: Its nature, scope, limits, and consequences", *Minds and Machines*, vol. 30, págs. 681-694, 2020.

[59] "La guía máxima para el modelo de lenguaje GPT-3 de OpenAI". (25 de ago. de 2020), dirección: https://www.twilio.com/es-mx/blog/la-guia-maxima-para-el-modelo-de-lenguaje-gpt-3-de-openai (visitado 26-02-2024).

[60] openAi, *GPT-4*, 2023. dirección: https://openai.com/gpt-4.

[61] N. Shazeer, A. Mirhoseini, K. Maziarz et al., "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer", *arXiv preprint arXiv:1701.06538*, 2017.

[62] C. Raffel, N. Shazeer, A. Roberts et al., "Exploring the limits of transfer learning with a unified text-to-text transformer", *The Journal of Machine Learning Research*, vol. 21, nº 1, págs. 5485-5551, 2020.

[63] S. CM, J. Prakash y V. S. Alaparthi, "Predicting semantic category of answers for question answering systems using transformers: a transfer learning approach", *Multimedia Tools and Applications*, págs. 1-21, 2024.

[64] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov y Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding", *Advances in neural information processing systems*, vol. 32, 2019.

[65] U. Fayyad, G. Piatetsky-Shapiro y P. Smyth, "From data mining to knowledge discovery in databases", *AI magazine*, vol. 17, nº 3, págs. 37-37, 1996.

[66] M.-E. Maurer, *Phishload*, 2012. dirección: https://www.medien.ifi.lmu.de/team/max.maurer/files/phishload/.

[67] Square, *Payment tokenization explained*, mar. de 2024. dirección: https://squareup.com/us/en/the-bottom-line/managing-your-finances/what-does-tokenization-actually-mean.

[68] Techslang, *What is Encoding?*, sep. de 2022. dirección: https://techslang.com/definition/what-is-encoding/.

[69] G. Lokare, "Preparing Text Data for Transformers: Tokenization, Mapping and Padding", feb. de 2023. dirección: https://medium.com/@lokaregns/preparing-text-data-for-transformers-tokenization-mapping-and-padding-9fbfbce28028#:~:text=Padding%20and%20truncation%20are%20preprocessing,all%20have%20the%20same%20length..

[70] E. Benavides-Astudillo, W. Fuertes, S. Sanchez-Gordon, D. Nuñez-Agurto y G. Rodríguez-Galán, “A Phishing-Attack-Detection Model Using Natural Language Processing and Deep Learning”, *Applied Sciences*, vol. 13, nº 9, pág. 5275, 2023.