UNIVERSIDAD EUROPEA DE MADRID

MÁSTER UNIVERSITARIO EN ANALISIS DE DATOS MASIVOS (BIG DATA)

TRABAJO FIN DE MÁSTER

Estudio comparativo de Modelos de Lenguaje de Gran Escala (LLM) aplicados a la detección de intrusiones en redes 5G

Ing. Kevin Francisco Silva Castro

Dirigido por

Ing. Jesús Gil Ruiz

Madrid, 2025

TÍTULO: **ESTUDIO COMPARATIVO DE MODELOS DE LENGUAJE DE GRAN ESCALA (LLM) APLICADOS A LA DETECCIÓN DE INTRUSIONES EN REDES 5G.**

AUTOR: ING. KEVIN SILVA CASTRO

TITULACIÓN: MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS (BIG DATA)

DIRECTOR/ES DEL PROYECTO: ING. JESÚS GIL RUIZ

FECHA: SEPTIEMBRE DE 2025

RESUMEN

Este Trabajo de Fin de Máster presenta una evaluación comparativa del rendimiento de modelos de lenguaje de gran escala (LLM) aplicados a la detección de intrusiones en redes 5G, un campo donde aún existe poca exploración científica. Utilizando conjunto de datos UNSW-NB15 y técnicas de preprocesamiento orientadas a procesamiento de lenguaje natural (PLN), se entrenaron y analizaron tres modelos representativos: GPT, BERT y RoBERTa.

El proyecto se desarrolló de forma individual, sin colaboración directa con empresa externa, aunque orientado a su aplicabilidad en el sector de la ciberseguridad, particularmente en entornos de telecomunicaciones y redes móviles.

Además, se aplicaron técnicas de Inteligencia Artificial Explicable (XAI), lo que permitió interpretar las decisiones del modelo y validar su confiabilidad, aportando un valor diferencial en contextos donde la transparencia es fundamental. Las conclusiones del estudio sugieren que el uso de modelos LLM en IDS representa una vía prometedora para mejorar la seguridad de redes de nueva generación.

Palabras clave: IDS (Sistema de Detección de Intrusiones), redes 5G, modelos de lenguaje (LLM), Inteligencia Artificial Explicable (XAI), ciberseguridad.

ABSTRACT

This Master's Thesis presents a comparative evaluation of the performance of large-scale language models (LLMs) applied to intrusion detection in 5G networks, a field where there is still little scientific exploration. Using the UNSW-NB15 dataset and preprocessing techniques geared toward natural language processing (NLP), three representative models were trained and analyzed: GPT, BERT, and RoBERTa.

The project was developed individually, without direct collaboration with an external company, although it was oriented towards its applicability in the cybersecurity sector, particularly in telecommunications and mobile network environments.

In addition, Explainable Artificial Intelligence (XAI) techniques were applied, which allowed the model's decisions to be interpreted and its reliability to be validated, providing differential value in contexts where transparency is essential. The study's conclusions suggest that the use of LLM models in IDS represents a promising avenue for improving the security of next-generation networks.

Keywords: IDS (Intrusion Detection System), 5G networks, language models (LLM), Explainable Artificial Intelligence (XAI), cybersecurity.

AGRADECIMIENTOS

En ocasiones se incluye este apartado para agradecer a aquellos que han ofrecido su ayuda en el desarrollo del trabajo, ya sea técnica o de otro tipo.

Cita - frase célebre / Dedicatoria

Esta página es del todo opcional, pero resulta una muy buena forma de presentar el trabajo académico más importante de todo el grado.

TABLA RESUMEN

	DATOS
Nombre y apellidos:	Kevin Francisco Silva Castro
Título del proyecto:	Estudio comparativo de modelos de lenguaje de gran escala (LLM) aplicados a la detección de intrusiones en redes 5G
Directores del proyecto:	ING. JESÚS GIL RUIZ
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa:	NO
El proyecto ha implementado un producto: (esta entrada se puede marcar junto a la siguiente)	SI
El proyecto ha consistido en el desarrollo de una investigación o innovación: (esta entrada se puede marcar junto a la anterior)	SI
Objetivo general del proyecto:	Aplicar LLMs para la detección de intrusiones en redes 5G

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1. RESUMEN DEL PROYECTO

1.1 Contexto y justificación

La quinta generación (5G), tecnología que ha llegado a reducir esa brecha tecnológica que existía o había dejado la conexión 4G (cuarta generación), con ciertas similitudes entre ambas, pero mejorando cada aspecto importante como: la velocidad, latencia, capacidad y conectividad más confiable, pero también esto ha llevado a aumentar o a desafiar la seguridad de estas redes móviles por parte de los ciberdelincuentes. Por otro lado, es inevitable que la IA (Inteligencia Artificial) tenga su protagonismo en estas redes ofreciendo mejoras en muchos aspectos y sobre todo en la seguridad, esto llevando a aplicar técnicas de Modelos de Lenguaje de Gran Escala (LLM), donde han mostrado un potencial creciente en tareas de detección de patrones analizando los tráficos de red para identificar amenazas de seguridad en tiempo real, prevenir ataques y proteger la red y los datos de los usuarios. Esta investigación se enmarca en el contexto de la seguridad informática aplicada a infraestructuras modernas de red, con el objetivo de evaluar la eficacia de estos modelos avanzados para la detección de tráfico malicioso en las redes 5G.

Este trabajo de fin de Máster (TFM) propone un estudio comparativo entre modelos LLM, evaluando su eficacia para la detección de intrusiones en redes 5G mediante el análisis de tráfico de red y su clasificación en tiempo real. En el contexto de la ciberseguridad en redes 5G, caracterizadas por su alto dinamismo y complejidad, las arquitecturas Transformers emergen como una solución prometedora para analizar grandes volúmenes de tráfico y detectar patrones anómalos en tiempo real, aprovechando su capacidad de aprendizaje profundo y paralelización.

Para ello, se emplearán datasets de referencia como UNSW-NB15, sobre los cuales se aplicarán diferentes modelos LLM, evaluando sus resultados mediante métricas estándar de clasificación. Se espera que los hallazgos de esta investigación sirvan como base para guiar futuras implementaciones de sistemas inteligentes de detección de amenazas en entornos 5G.

Como complemento técnico, también se explorarán técnicas básicas de XAI (Inteligencia Artificial Explicable), con el fin de interpretar de forma preliminar las decisiones de los modelos y aportar transparencia al proceso de detección, sin que esto desplace el foco principal del estudio. Esto permitirá no solo evaluar el rendimiento cuantitativo de los modelos, sino también obtener indicadores sobre la confiabilidad y trazabilidad de sus predicciones en contextos reales.

1.2 Planteamiento del problema

Aunque las redes neuronales han mejorado la detección de intrusiones en entornos digitales, todavía no está claro qué arquitectura de Modelos de Lenguaje de Gran Escala (LLM) resulta más eficaz en el contexto particular de las redes 5G, caracterizadas por su complejidad y alto volumen de datos en tiempo real. El estudio busca responder a la siguiente pregunta: **¿Qué Modelo de Lenguaje de Gran Escala ofrece un rendimiento más eficiente y preciso en la detección de**

intrusiones en redes 5G?, el estudio comparativo de diferentes modelos LLM será aplicado sobre conjunto de datos de ciberseguridad de referencia, usando UNSW-NB15 para este trabajo de ámbito científico-técnico, incorporando técnicas de aprendizaje profundo (Deep Learning).

Este estudio no pretende diseñar nuevos modelos LLM desde cero, sino evaluar el desempeño de modelos existentes al ser aplicados a un entorno específico de seguridad en redes 5G, empleando datasets reales de tráfico.

1.3 Objetivos del proyecto

Los objetivos de este proyecto tienen como desafío identificar diferencias significativas entre los modelos evaluados, en términos de precisión, eficiencia y capacidad de generalización según el tipo de ataque y el conjunto de datos utilizado. Además de aplicar técnicas de interpretabilidad que permitan entender el razonamiento de los modelos en la detección de tráfico malicioso.

1.4 Resultados obtenidos

Este proyecto permitirá la comparativa de diferentes modelos LLM aplicados a la detección de intrusiones en redes 5G, se pretende analizar el comportamiento de cada modelo en función de su capacidad para redactar tráfico malicioso, clasificar ataques conocidos y desconocidos con precisión, eficiencia y en tiempo real.

1.5 Estructura de la memoria

La memoria del proyecto se organiza en los siguientes capítulos:

- **Capítulo 1 – Introducción:** Se presenta el contexto del problema, los objetivos del proyecto y la motivación para abordar la detección de intrusiones en redes 5G utilizando modelos LLM.
- **Capítulo 2 – Estado del Arte:** Se describen los fundamentos de las redes 5G, el aprendizaje profundo, los modelos LLM y las técnicas de IA explicable (XAI).
- **Capítulo 3 – Objetivos:** Se detalla el proceso de selección y adaptación de datasets, los modelos implementados, el entorno de pruebas y las métricas utilizadas para la evaluación.
- **Capítulo 4 – Desarrollo del proyecto:** Se presentan los resultados obtenidos con cada modelo, comparaciones entre ellos y el análisis de los comportamientos observados, incluyendo visualizaciones.
- **Capítulo 5 – Discusión:** Se discuten las principales conclusiones del estudio, se identifican las limitaciones encontradas y se proponen líneas futuras de investigación.
- **Capítulo 6 – Conclusiones:** Se discuten las principales conclusiones del estudio, se identifican las limitaciones encontradas y se proponen líneas futuras de investigación.
- **Capítulo 7 – Futuros trabajos:** Se discuten las principales conclusiones del estudio, se identifican las limitaciones encontradas y se proponen líneas futuras de investigación.

- **Capítulo 8 – Referencias:** Se incluyen todas las fuentes bibliográficas citadas a lo largo del trabajo.
- **Capítulo 9 – Anexos:** Se adjunta el material adicional relevante que complementa el proyecto.

Capítulo 2. ANTECEDENTES / ESTADO DEL ARTE

Para iniciar este estudio se empezó a investigar el problema con ayuda de proyectos similares que aporten comprensión al contexto de este TFM.

2.1 Estado del arte

Estudios recientes han comenzado a aplicar arquitecturas Transformer, aplicando modelos para diversas tareas de ciberseguridad, pero su aplicación especifica en entornos 5G aún se encuentra en etapas exploratorias, siendo escasos los trabajos comparativos entre modelos LLM en esta área. Se analiza tanto los desafíos que presenta esta nueva generación de redes como la arquitectura de modelos que han sido utilizadas en tareas similares. Esta revisión permite sustentar la selección de modelos a comparar, así como las decisiones metodológicas adoptadas en el presente trabajo.

2.1.1 Redes 5G

Existe muchos conceptos acerca de las redes 5G, en la página oficial de Fortinet (Fortinet Inc, s.f.) indican un significado especifico de lo que son y lo que esta tecnología ofrece:

5G es la red móvil de 5.a generación. Es un nuevo estándar inalámbrico global después de las redes 4G. La red 5G permite un nuevo tipo de red que está diseñada para conectar prácticamente a todos y a todo lo que hay entre sí, incluidas las máquinas, los objetos y los dispositivos.
La tecnología inalámbrica 5G está diseñada para ofrecer mayores velocidades de datos pico de múltiples Gbps (Gigabits por segundos), latencia ultra baja, más confiabilidad, capacidad de red masiva, mayor disponibilidad y agilidad. Estos tienen el potencial de ofrecer nuevas experiencias de usuario y conectar nuevas industrias.
El mismo Fortinet indica los desafíos que esta tecnología presenta, específicamente en seguridad y que requiere que se tome en consideración lo siguiente:

- Superficie de ataque expandida: El aumento masivo de los dispositivos conectados y la naturaleza distribuida de las redes 5G crean una superficie de ataque más grande para que los cibercriminales la aprovechen. Cada dispositivo conectado, desde teléfonos inteligentes hasta sensores de IoT, representa un punto de entrada potencial para actores maliciosos.
- Seguridad de dispositivos IoT: La proliferación de dispositivos de IoT, a menudo con capacidades de seguridad limitadas, representa un riesgo significativo. Los dispositivos de IoT comprometidos se pueden utilizar para lanzar ataques, interrumpir redes o robar datos sensibles.
- Riesgos del corte de red: La segmentación de red en 5G ofrece personalización y aísla los recursos de red. Sin embargo, la implementación incorrecta puede crear vulnerabilidades, lo que potencialmente permite el acceso no autorizado o ataques de corte cruzado.

Por otro lado, Cisco System (Cisco System, s.f.) siendo una potencia en tecnologías de equipos de telecomunicaciones, indica un concepto más técnico acerca de las redes 5G:

La tecnología 5G tiene una velocidad máxima teórica de 20 Gbps, mientras que la velocidad máxima de 4G es de solo 1 Gbps. 5G también promete una latencia más baja, lo que puede mejorar el rendimiento de las aplicaciones comerciales, así como otras experiencias digitales (como juegos en línea, videoconferencias y autos autónomos).

Mientras que las generaciones anteriores de tecnología celular (como 4G LTE) se centraban en garantizar la conectividad, 5G lleva la conectividad al siguiente nivel al ofrecer experiencias conectadas desde la nube a los clientes. Las redes 5G están virtualizadas y controladas por software, y explotan las tecnologías de la nube.

La red 5G también simplificará la movilidad, con interrupciones entre el acceso celular y el wifi. Los usuarios móviles pueden mantenerse conectados mientras se desplazan entre conexiones inalámbricas exteriores y redes inalámbricas dentro de edificios sin intervención del usuario ni necesidad de volver a autenticarse.

El nuevo estándar inalámbrico Wi-Fi 6 (también conocido como 802.11ax) comparte características con el 5G, incluyendo un rendimiento mejorado. Las radios Wi-Fi 6 pueden ubicarse donde los usuarios las necesiten para ofrecer una mejor cobertura geográfica y un menor coste. Estas radios Wi-Fi 6 se basan en una red basada en software con automatización avanzada.

La tecnología 5G debería mejorar la conectividad en zonas rurales desatendidas y en ciudades donde la demanda puede superar la capacidad actual con la tecnología 4G. Las nuevas redes 5G también contarán con una arquitectura densa y de acceso distribuido y acercarán el procesamiento de datos al borde y a los usuarios para permitir un procesamiento más rápido.

2.1.2 Modelos de lenguaje de gran escala (LLM)

Existen diferentes términos que definen un LLM, lo llaman modelos o arquitecturas. En Amazon (AWS Amazon, 2024) lo describen como:

Los modelos de lenguaje de gran tamaño, también conocidos como LLM, son modelos de aprendizaje profundo muy grandes que se pre-entrenan con grandes cantidades de datos. El transformador subyacente es un conjunto de redes neuronales que consta de un codificador y un decodificador con capacidades de autoatención. El codificador y el decodificador extraen significados de una secuencia de texto y comprenden las relaciones entre las palabras y las frases que contiene.

Los transformadores LLM son capaces de entrenarse sin supervisión, aunque una explicación más precisa es que los transformadores llevan a cabo un autoaprendizaje. Es a través de este

proceso que los transformadores aprenden a entender la gramática, los idiomas y los conocimientos básicos.

Además de la definición técnica ofrecida por Amazon, otras compañías como Google Cloud (Google Cloud, s.f.) proporcionan una descripción más general:

Un modelo de lenguaje extenso (LLM) es un modelo estadístico de lenguaje, entrenado con una gran cantidad de datos, que se puede usar para generar y traducir texto y otros contenidos, así como realizar otras tareas de procesamiento de lenguaje natural (PLN).

Por lo general, los LLM se basan en arquitecturas de aprendizaje profundo, como Transformer, desarrollado por Google en 2017, y se pueden entrenar con miles de millones de textos y otros tipos de contenido.

2.1.3 Modelos LLM seleccionados para el estudio comparativo

Los modelos seleccionados han sido explorados en investigaciones prácticas por su capacidad para procesar secuencias de datos y extraer patrones relevantes en contextos complejos, estos estudios se han visto relacionados con redes de comunicación, ciberseguridad y análisis de tráfico. La selección de estos modelos para el presente trabajo se basa en su uso documentado en escenarios similares, así como su potencial para ofrecer un rendimiento competitivo en métricas claves.

2.1.3.1 BERT

Bert (Bidirectional Encoder Representations from Transformers) (Canales luna, 2024), este modelo de código abierto que desarrolló Google en 2018 fue y sigue siendo una de las arquitecturas de redes neuronales más utilizadas, aunque hoy en día con sus variantes y adaptaciones contamos con versiones de BERT más avanzadas y en tareas específicas. Con BERT se puede hacer uso de la arquitectura Transformer para el análisis de secuencias y tareas de clasificación.

2.1.3.2 Aplicaciones de BERT en detección de intrusiones en redes 5G

Uno de los trabajos realizados bajo el modelo BERT, fue el de “Efficient Federated Intrusion Detection in 5G ecosystem using optimized BERT-based model” por (Adjewa, Esseghir, & Merghem-Boulahia, 2024), donde desarrollaron un IDS (Sistema de Detección de intrusiones). Su propuesta, securityBERT, optimizó las capas inferiores de BERT y aplicó técnicas de cuantificación para reducir el tamaño del modelo en un 28,74% sin comprometer el rendimiento.

Implementan un modelo de aprendizaje federado para entrenar de forma distribuida sin necesidad de centralizar los datos, preservando la privacidad de los usuarios. En la Figura 4- 1 muestra el flujo de aprendizaje federado utilizado para entrenar su modelo securityBERT, donde los modelos locales se entrenan en los dispositivos y solo comparten parámetros con el servidor local.

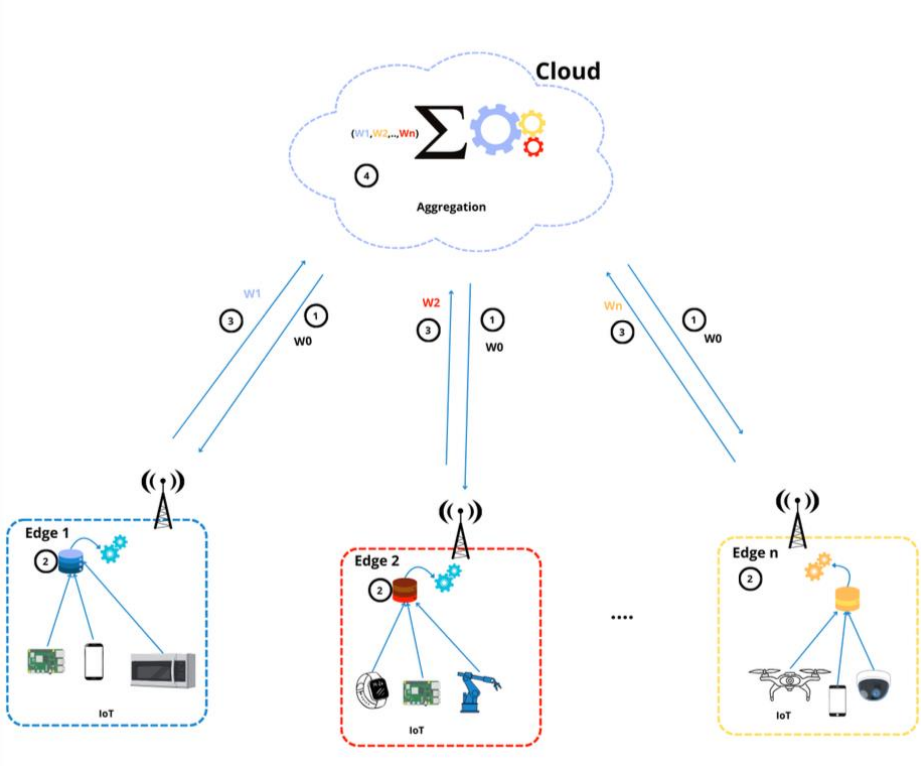


Figura 4- 1 Flujo de aprendizaje federado propuesto por (Adjewa, Esseghir, & Merghem-Boulahia, 2024)

En sus resultados demostrarón que su modelo alcanzó una precisión del 97.79% en un entorno centralizado y 97.12% en un entorno federado con datos IID (Distribuidos de forma homogénea). Incluso en escenarios màs realistas, con datos no-IDD (Heterogéneos), obtuvo una precisión del 96.66%, lo que demuestra que BERT puede funcionar en entornos 5G.

Tomando de referencia la Tabla III de esta investigación (ver Figura 4- 2), donde utilizaron diferentes datasets para la simulación, UNSW-NB15 es el dataset que por su amplia variedad de ataques, su composición equilibrada de tráfico normal y malicioso es el que refleja entornos más cercanos a las redes 5G y lo hacen el candidato adecuado para este TFM al reflejar escenarios más cercanos a un tráfico real de las redes 5G.

TABLE III: Datasets available for simulating IDS

Dataset	Year	Number of Features	Number of Attacks	IoT	Label	Algorithm	Reference
NSL-KDD	2009	41	4	✗	✓	DNN, ANN, RF, KNN, SVM	[19]
UNSW-NB15	2015	49	9	✗	✓	DNN, ensemble methods, LSTM, SVM	[15]
CIC-IDS2017	2017	80	8	✗	✓	KNN, RF, MLP, AdaBoost, Naive Bayes, ID3, QDA	[18]
BOT-IoT	2019	46	8	✓	✓	SVM, LSTM, RNN	[7]
Edge-IIoTset	2022	61	14	✓	✓	DT, RF, KNN, SVM	[7]

Figura 4- 2 Datasets utilizados en la simulación de securityBERT por (Adjewa, Esseghir, & Merghem-Boulahia, 2024)

Con respecto a su algoritmo, en GitHub esta su código fuente de securityBERT (Adjewa F. y.-B., 2024), lo que constituye un valioso recurso para esta investigación, dicho repositorio servirá como base de referencia y guía para la implementación de los experimentos de este TFM.

2.1.3.3 RoBERTa

RoBERTa (Robustly Optimized BERT Pretraining Approach) es una mejora sobre BERT (FAIRSEQ, s.f.), este modelo elimina la tarea de predicción en la siguiente oración y se entrena con mayor cantidad de datos, su arquitectura ha sido utilizada en tareas como clasificación de eventos en ciberseguridad y detección de phishing que dan resultados mejores que el BERT tradicional, en métricas de precisión y F1-score.

La relevancia de RoBERTa en el contexto de este TFM radica en su capacidad de capturar dependencias complejas en datos secuenciales, lo cual es útil en el análisis de patrones anómalos de tráfico de red. Esto lo convierten en un candidato fuerte para ser evaluado en este estudio comparativo.

Existe un ejemplo representativo de este modelo, en el artículo "Detección de Noticias Falsas (Fake News) en Internet usando Deep Learning" (Jóvenes en la Ciencia, 2022), este estudio, aunque su aplicación no sea a redes informáticas, su metodología implementada incluye el preprocesamiento de texto, la vectorización y la evaluación de métricas como exactitud, precisión y F1-score resulta útil como referente metodológico para el presente TFM.

La Figura 4- 3 muestra el dominio que RoBERTa obtuvo de manera consistente ante las otras variantes de LLM evaluadas en el trabajo antes mencionado, alcanzando métricas (accuracy y F1) de 0.981 en el conjunto de prueba utilizado.

Modelo	Subconjunto de Validación				Subconjunto de Prueba			
	Accuracy	Precisión	Recall	F1	Accuracy	Precisión	Recall	F1
AlBERT (base, v2)	0.965	0.965	0.965	0.965	0.965	0.965	0.965	0.965
BERT (base, uncased)	0.972	0.973	0.971	0.972	0.971	0.972	0.971	0.971
ELECTRA (base)	0.970	0.970	0.969	0.970	0.973	0.974	0.973	0.973
RoBERTa (base)	**0.978**	**0.978**	**0.977**	**0.978**	**0.981**	**0.981**	**0.980**	**0.981**
XLM-RoBERTa (base)	0.972	0.973	0.972	0.972	0.962	0.962	0.962	0.962
LR	0.891	0.891	0.890	0.890	0.886	0.891	0.890	0.890

Figura 4- 3 Resultados de evaluación con los diferentes modelos de clasificación y sus métricas por (Jóvenes en la Ciencia, 2022)

Este resultado evidencia que la arquitectura RoBERTa, posee una capacidad superior para extraer patrones contextuales complejos y robustos frente a ruido y desbalanceo de clases. Dichas características son directamente relevantes para el análisis de flujos de tráfico en 5G, donde se requiere identificar anomalías en secuencias con gran variabilidad y entornos ruidosos. Así, aunque el estudio original se centró en noticias falsas sobre COVID-19, es posible sostener académicamente que RoBERTa puede ser perfectamente adaptado a la detección de intrusiones y clasificación de tráfico en 5G.

2.1.3.4 GPT-2

GPT-2 (Generative Pre-trained Transformer 2) (OpenAI, 2019) es un modelo de lenguaje basado en la arquitectura Transformer y entrenado con grandes volumenes de texto de internet para predecir la siguiente palabra en una secuencia. Aunque esta sea su propósito inicial, su metodología lo convierte en un candidato versátil para tareas más allá del texto convencional, como la detección de intrusiones en redes.

2.1.3.5 Aplicaciones de GPT en detección de intrusiones en redes 5G

Existen varios proyectos como antecedentes, donde se demuestra la aplicabilidad de GPT-2 en entornos de ciberseguridad, uno de estos antecedentes es el proyecto “NIDS-GPT” (Huang, 2024), se utilizó GPT-2 para procesar tráfico de red representando cada número de un paquete como una palabra, esto significó alcanzar grandes niveles de precisión en detección de intrusiones y usaron un conjunto de datos experimental como es CICIDS2017 y CAN-intrusion.

En la Figura 4- 4 se aprecia los resultados que obtuvo el estudio NIDS-GPT, se evaluó el uso del modelo para detección de intrusiones mostrando resultados sobresalientes. En el dataset utilizado (CICIDS-2017), alcanzó precisión, recall y F1 de 1.00, superando ampliamente a modelos tradicionales.

method	avg	Precision	Recall	F1-score
ET	macro avg	0.87	0.70	0.76
RandomForest	macro avg	0.87	0.62	0.76
optimized_RF [29]	macro avg	0.76	0.54	0.60
CNN_BiLSTM [30]	macro avg	0.09	0.12	0.09
Ours	macro avg	**1.00**	**1.00**	**1.00**

Figura 4- 4 Resultados en comparación con otros modelos o métodos (NIDS-GPT) por (Huang, 2024)

Con esta base técnica, se tomará como referencia académica y práctica el repositorio oficial de NIDS-GPT, para que sea adaptado dentro del análisis comparativo usando en este caso el dataset de tráfico de red 5G (UNSW-NB15). Se aprovechará su esquema de tokenización de paquetes, entrenamiento y ciertos hiperparámetros con el fin de asegurar una evaluación homogénea y fortalecer la validez de los resultados experimentales.

2.1.3.6 LLaMA

LlaMA (Large Language Model Meta AI) este modelo diseñado por Meta AI (Meta, 2023) trae diferentes versiones, para este caso de TFM interesa mucho la versión de LLaMA 2 porque esta entrenada especificamente para tareas de análisis en redes 5G, así lo detalla el artículo "Mobile-LLaMA: Instruction Fine-Tuning Open-Source LLM for Network Analysis in 5G Networks" (Kan, Mun, Cao, & Lee, 2024), en donde este modelo demuestra un rendimiento significativo en la tarea de análisis de enrutamiento IP.

2.1.3.7 Aplicaciones de LLaMA en detección de intrusiones en redes 5G

Mobile-LLaMA es una versión especializada del modelo LLaMA 2 (13B parámetros), diseñada específicamente para el análisis de redes móviles 5G mediante la técnica de *instruction fine-tuning*. Su desarrollo surge como respuesta a la necesidad de herramientas personalizadas que permitan realizar análisis de tráfico, enrutamiento y rendimiento en redes 5G, sin comprometer la privacidad de los datos, algo especialmente relevante en entornos sensibles como los de telecomunicaciones. Este modelo se construyó utilizando datasets públicos reales, como tablas BGP (para enrutamiento), archivos PCAP (para tráfico) y métricas de red estructuradas en CSV (para evaluación de rendimiento), incorporando librerías como PyBGPStream, Scapy, Pandas y Matplotlib en el entrenamiento para una generación de código precisa y funcional.

Para dimensionar la magnitud y complejidad del proyecto Mobile-LLaMA, en la Figura 4- 5 se presenta la arquitectura propuesta en el proyecto Mobile-LLaMA. Esta integra desde la generación de instrucciones con modelos previos (OpenAI), hasta el ajuste fino de LLaMA con más de 15.000 instrucciones específicas de análisis de redes móviles, abarcando funciones de enrutamiento IP, análisis de paquetes y evaluación de rendimiento.

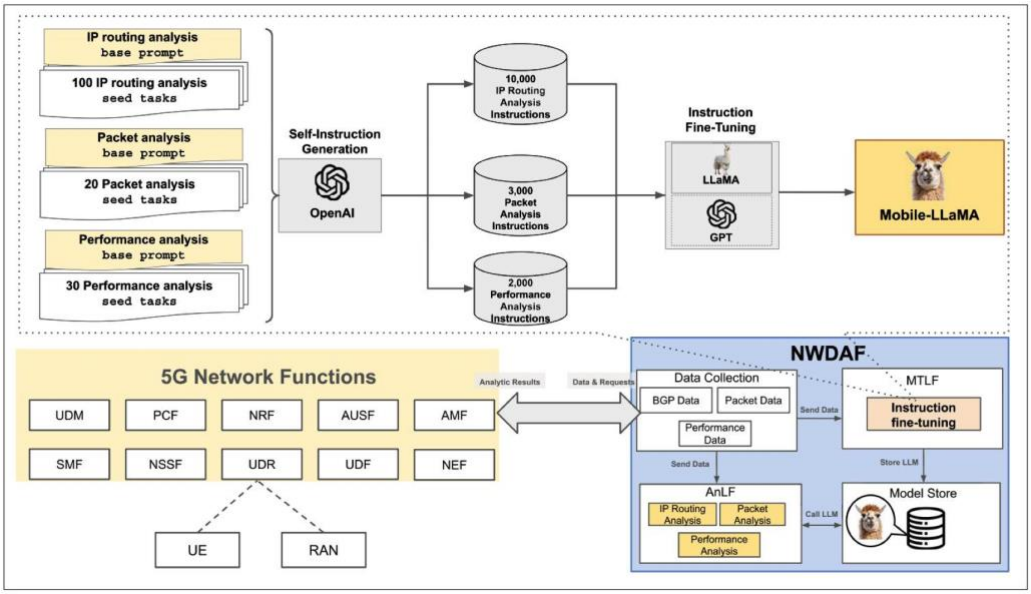


Figura 4- 5 Arquitectura NWDAF implementada por (Kan, Mun, Cao, & Lee, 2024)

Aunque el trabajo de (Kan, Mun, Cao, & Lee, 2024) propongan técnicas de ajuste fino más eficiente, como LoRA (Low-Rank-Adaptation), que permiten reducir los requerimientos de memoria y posibilitan la ejecución de modelos LLaMA en hardware más accesible, la implementación práctica aún implica una demanda significativa de recursos computacionales.

Pese a la relevancia de estos hallazgos, en el presente estudio no se incluirá la ejecución experimental de este modelo. La decisión es por las limitaciones prácticas de infraestructura computacional y configuración del entorno, dado que el entrenamiento e inferencia de un LLM requieren recursos de GPU (Unidad de Procesamiento Gráfico) de alto rendimiento y un stack (conjunto de herramientas, librerías, etc.) optimizado que exceden el alcance de este estudio.

Por otro lado, se puede tomar de referencia los resultados del proyecto mobile-LLaMA, ya que este fue comparado directamente con modelos GPT y otras variantes de este LLM, se lo puede observar en la Figura 4- 6 que obtuvo un desempeño notablemente superior para tareas de análisis de redes (IP routing, análisis de paquetes y análisis de rendimiento), reportando métricas en cada una de estas categorías.

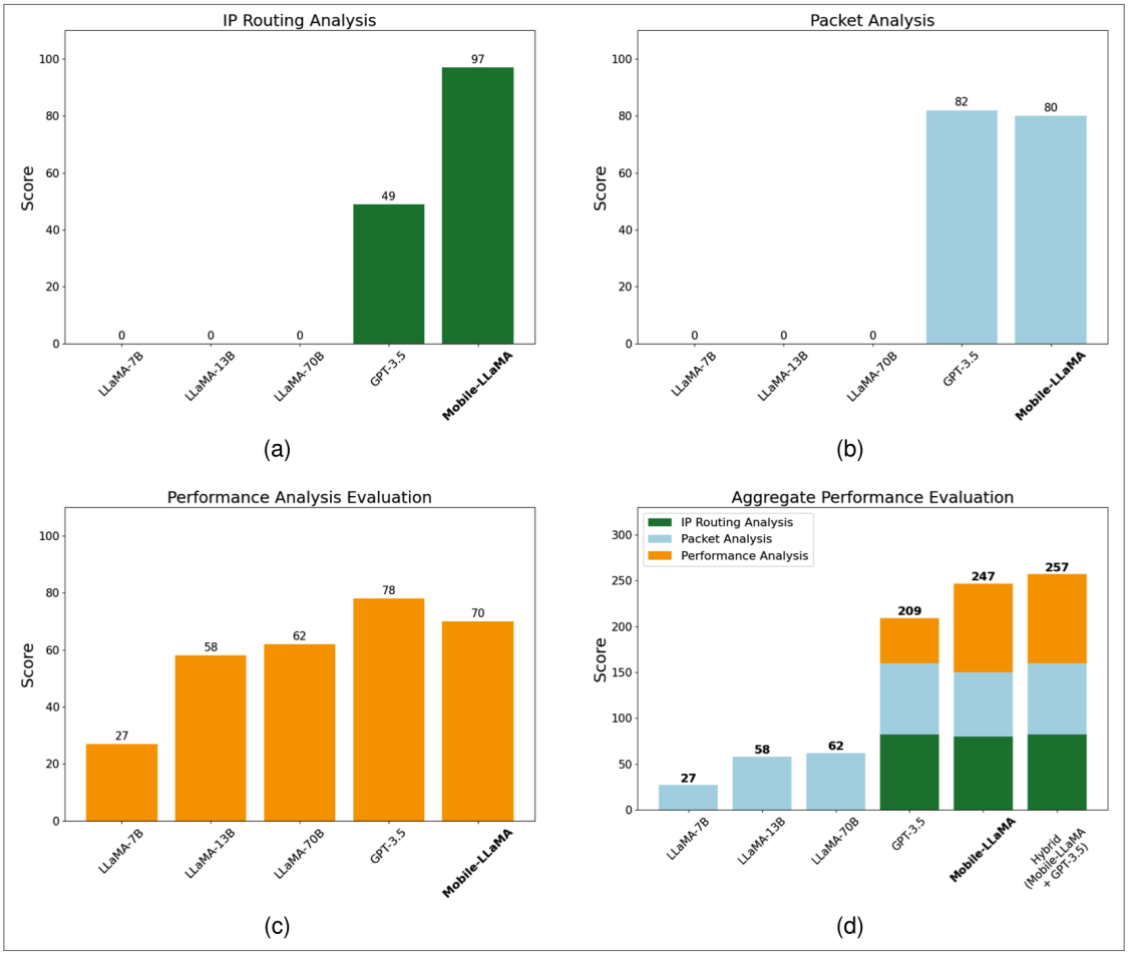


Figura 4- 6 Análisis de resultados realizado en el proyecto Mobile-LLaMA por (Kan, Mun, Cao, & Lee, 2024)

2.2 Contexto y justificación

En la actualidad, las redes 5G representan una evolución significativa en el ámbito de las telecomunicaciones, con características significativas como alta velocidad, baja latencia, y capacidad para conectar un número masivo de dispositivos. Estas ventajas han facilitado el despliegue de soluciones como ciudades inteligentes, vehículos autónomos y aplicaciones críticas de salud, pero también han abierto la puerta a nuevos desafíos de seguridad. Las arquitecturas distribuidas, la virtualización de red (NFV), y la exposición de interfaces abiertas (APIs) hacen que estas redes sean vulnerables a amenazas emergentes difíciles de detectar con mecanismos tradicionales.

El presente trabajo propone una comparativa en el rendimiento de distintos modelos LLM en escenarios de ciberseguridad aplicados específicamente a redes 5G. Aunque existen estudios previos que aplican a modelos como BERT o RoBERTa en entornos de red, aún es limitada la evidencia que contraste su eficacia con otros modelos recientes como LLaMA en datasets representativos del contexto 5G. Además, resulta necesario no solo evaluar métricas como precisión, sino también aspectos como la trazabilidad de sus predicciones, el tiempo de respuesta y su capacidad de generalización.

Por lo tanto, este estudio pretende aportar conocimientos valiosos tanto a nivel académico como práctico, proporcionando una comparación rigurosa entre modelos seleccionados bajo un enfoque experimental. Más allá de obtener un ranking de rendimiento de modelos, se propone también construcción de un entorno adaptable, capaz de soportar la integración de múltiples LLM en procesos de evaluación de comparativa. Este enfoque no solo permite analizar métricas tradicionales, sino que también habilita futuras extensiones del trabajo, como el análisis de nuevos modelos o combinaciones híbridas, sin necesidad de reconstruir la arquitectura base. En contextos dinámicos como las redes 5G, contar con una herramienta flexible de benchmarking (estudio o análisis de competidores) puede representar un aporte significativo a la toma de decisiones en seguridad informática y arquitectura de red.

2.3 Planteamiento del problema

El análisis del estado del arte revela una creciente aplicación de modelos basados en arquitecturas Transformer, como GPT, BERT o RoBERTa, en el ámbito de la ciberseguridad. Sin embargo, la mayoría de los estudios se centran en entornos genéricos o en redes previas al 5G, lo cual limita su aplicabilidad a contextos con latencias ultra bajas, alta densidad de dispositivos y entornos distribuidos como lo que platea el 5G.

La falta de estudios comparativos rigurosos que evalúen el comportamiento de múltiples LLM sobre datasets realistas de ciberseguridad, como el UNSW-NB15 utilizado en este estudio, dificulta la identificación de modelos óptimos para la detección de intrusiones en estas redes móviles modernas. Además, existe una escasa integración de técnicas de XAI que permitan interpretar las decisiones de dichos modelos, lo cual reduce su aplicabilidad en entornos críticos donde la auditabilidad de las decisiones automatizadas es clave.

En consecuencia, el problema central que aborda este proyecto puede formularse como:

> ¿Cómo es el desempeño comparativo de distintos Modelos de Lenguaje de Gran Escala (LLM) aplicados a la detección de intrusiones en redes 5G, considerando tanto métricas de clasificación como criterios de interpretabilidad y eficiencia computacional?

Esto permitirá no solo avanzar en la selección de modelos adecuados para la detección inteligente de intrusiones, sino también establecer criterios metodológicos replicables para futuras investigaciones en el cruce entre PLN (Procesamiento de Lenguaje Natural), ciberseguridad e infraestructura de red avanzada.

Capítulo 3. OBJETIVOS

3.1 Objetivos generales

El objetivo general del presente TFM es evaluar y comparar el rendimiento de diferentes modelos de lenguaje de gran escala (LLM) aplicados a la detección de intrusiones en redes 5G, mediante el análisis de sus métricas de clasificación sobre un conjunto de datos de ciberseguridad previamente estructurado.

3.2 Objetivos específicos

- Seleccionar modelos LLM representativos mediante el análisis del estado del arte que puedan ser adaptados a tareas de detección de intrusiones.
- Preparar el conjunto de datos de ciberseguridad mediante técnicas de procesamiento y adaptación, con el fin de garantizar su compatibilidad y utilidad en la evaluación comparativa de los modelos LLM seleccionados.
- Evaluar el rendimiento de los modelos LLM usando métricas estándar de clasificación, aplicándolos en tareas de detección de intrusiones sobre un conjunto de datos de ciberseguridad.
- Comparar los resultados obtenidos mediante el análisis de sus resultados experimentales, con el fin de identificar cuál ofrece una mayor eficacia en la detección de intrusiones en redes 5G.

Capítulo 4. DESARROLLO DEL PROYECTO

4.1 Planificación del proyecto

El presente proyecto constara de seis fases:

FASE 1: Revisión del estado del arte (Meses 1-2)

Objetivo: Consolidar el conocimiento teórico y técnico del uso de LLMs en ciberseguridad, específicamente en entornos 5G.

- Revisión bibliográfica sobre detección de intrusiones.
- Selección de modelos como BERT, RoBERTa, LLaMa aplicados a seguridad.
- Descarga del dataset UNSW-NB15 o similares.

FASE 2: Preprocesamiento del dataset (Meses 2-3)

Objetivo: Preparar los datos relevantes para alimentar los modelos seleccionados.

- Selección del dataset UNSW-NB15.
- Limpieza y normalización de los datos.
- Transformación de datos en secuencia de texto.

FASE 3: Implementación de los modelos (Meses 3-4)

Objetivo: Adaptar y entrenar los modelos seleccionados para tareas de clasificación de tráfico.

- Fine-tuning de modelos para un entrenamiento supervisado. (Transformers con PyTorch o TensorFlow)
- Evaluación de rendimiento (métricas claves).

FASE 4: Evaluación y Comparativa de resultados (Meses 4-5)

Objetivo: Analizar el rendimiento de los modelos en base a métricas estándar.

- Evaluación con precisión, recall, F1-score y exactitud.
- Visualización de resultados generando gráficas comparativas.

FASE 5: Análisis de resultados y redacción (Meses 5-6)

Objetivo: Sintetizar los hallazgos y generar conclusiones del estudio.

- Reflexión sobre limitaciones, impactos y futuros trabajos.

- Prepara presentación, elaborar visualizaciones comparativas.

FASE 6: Validación, Revisión y entrega (Mes 6)

Objetivo: Garantizar la calidad y coherencia del trabajo final.

- Revisión ortográfica, técnica y formal.
- Integración de referencias
- Publicación en revistas de artículos científicos.

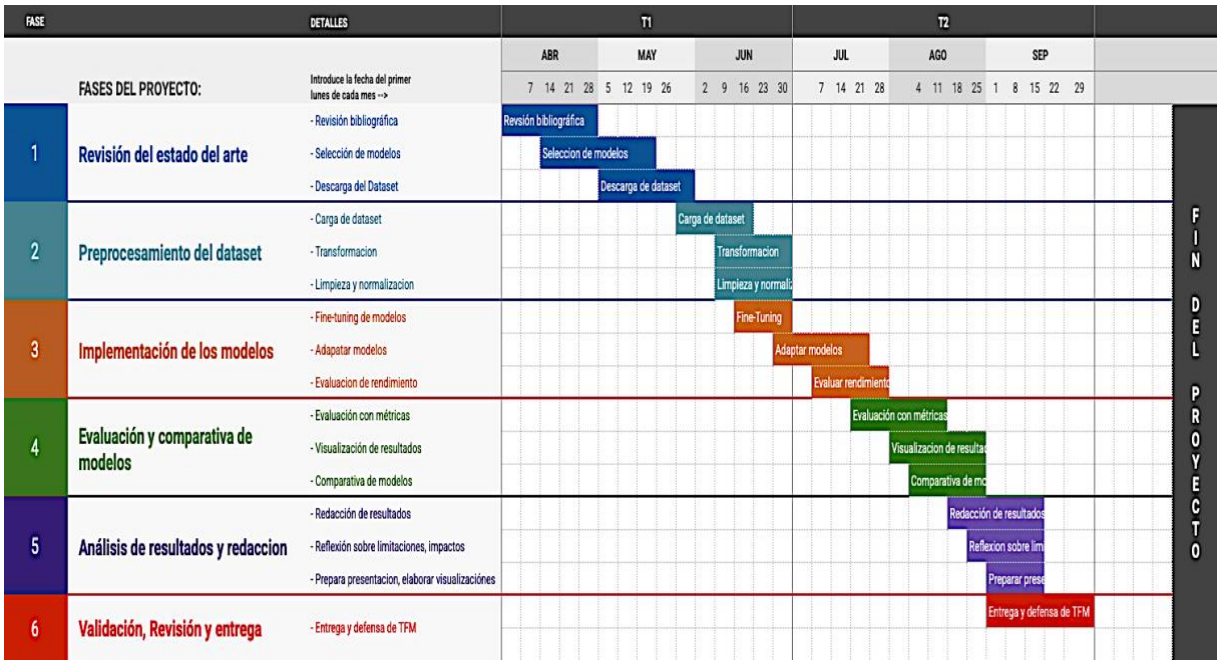


Figura 4- 7 Planificación del proyecto

4.2 Descripción de la solución, metodologías y herramientas empleadas

4.2.1 Enfoque metodológico

Este trabajo de fin de máster se enmarca en un enfoque cuantitativo y experimental. Se sigue una metodología de análisis comparativo entre distintos Modelos de Lenguaje de Gran Escala (LLM) preentrenados, aplicados a la detección de intrusiones en redes 5G. La selección de modelos se fundamenta en el estado del arte, priorizando arquitecturas ampliamente utilizadas en tareas de clasificación.

El diseño metodológico contempla el uso de fine-tuning sobre datasets específicos del ámbito de la ciberseguridad, aplicando criterios consistentes de evaluación (precisión, recall, F1-score, exactitud), para comparar el rendimiento de cada modelo en entornos de red altamente dinámicos como el 5G.

4.2.2 Metodología aplicada

El proyecto se ha estructurado en las siguientes fases:

- **Obtención de datos:** Se emplea el conjunto de datos UNSW-NB15 (University of New South Wales, 2015), este es un dataset público y con fines académicos generado por la University Of New South Wales (UNSW) en colaboración con el Austrialian Centre For Cyber Security (ACCS). Este dataset cumple con los principios de gobernanza de datos, tales como transparencia, trazabilidad y accesibilidad controlada.
- **Obtención de LLM:** Mediante el estado del arte y en base a otros trabajos relacionados se seleccionó los modelos BERT (Canales luna, 2024), RoBERTa (FAIRSEQ, s.f.), GPT-2 (OpenAI, 2019) y LlaMA (Meta, 2023), estos modelos gracias a su disponibilidad de implementación y código abierto en plataformas como GitHub, permitirá extraer fragmentos de códigos de estos repositorios oficiales y notebooks de referencia para adaptarlos a la detección de tráfico maliciosos en la redes 5G.
  Aunque se lo menciona en el estado del arte y que existen trabajos relacionados del modelo LlaMA, este demanda un alto coste de recursos y por esta razon en la fase de experimentación no se la considera, pero si se deja constancia de que este modelo trabaja muy bien para analisis de tráfico de red.
- **Preprocesamiento de datos**: Limpieza y normalización del dataset UNSW-NB15, conversión de registros a formato textual, tokenización con bibliotecas como transformers de Hugging Face.
- **Entrenamiento supervisado**: Adaptación de modelos LLM mediante técnicas de *transfer learning* y *fine-tuning*, entrenando sobre el dataset preparado mediante frameworks como PyTorch y TensorFlow.
- **Evaluación experimental**: Aplicación de métricas estándar de clasificación para comparar el rendimiento entre modelos. Se incluyen análisis estadísticos descriptivos y visualización de resultados mediante gráficos y tablas.
- **Interpretabilidad**: Se contemplan técnicas de interpretación como análisis de atención de los modelos, permitiendo entender las decisiones tomadas por la red neuronal en casos de detección de tráfico malicioso.

4.2.2.1 Obtención de datos

Una vez seleccionados los modelos adecuados para este tipo de estudio, era necesario identificar el dataset que permitiera disponer de datos que se relacionen a un escenario real como el tráfico de las redes 5G.

Para este estudio se decidió utilizar varios conjuntos de datos provenientes de diferentes fuentes, UNSW-NB15 para el dataset de tráfico de redes 5G (University of New South Wales, 2015), esta fuente permite descargar desde OneDrive los datos en formato csv.

Antes de cargarlos al modelo, aunque este dataset viene listo para entrenamiento, se decidió revisarlo para conocer o familiarizarse con los datos y sus variables. Se utilizó código en Python

(Python Software Foundation, 2024) y en la plataforma Google Colab (Google, 2025) para la ejecución de los modelos seleccionados y carga de dataset (Ver Figura 4- 8).

```
import pandas as pd

train_nb15= pd.read_csv('/content/UNSW_NB15_training-set.csv')
test_nb15 = pd.read_csv('/content/UNSW_NB15_testing-set.csv')

# Verificar Registros
train_nb15['label'].value_counts()
train_nb15['attack_cat'].value_counts()
```

attack_cat	count
Normal	56000
Generic	40000
Exploits	33393
Fuzzers	18184
DoS	12264
Reconnaissance	10491
Analysis	2000
Backdoor	1746
Shellcode	1133
Worms	130

Figura 4- 8 Código de Carga de dataset UNSW-NB15

Esta será nuestra base para la implementación de los diferentes modelos seleccionados, cada uno tendrá la misma carga de datos.

4.2.2.2 Obtención de los modelos LLM

La obtención del primer modelo LLM fue BERT, la implementación de este modelo fue guiado por el trabajo relacionado con securityBERT (Adjewa F. y.-B., 2024), sirvió de base y guía para la adaptación de este modelo al conjunto de datos seleccionado.

En la Figura 4- 9 se realiza la instalación de librerías necesarias para ejecutar BERT, librerías básicas para análisis y visualización.

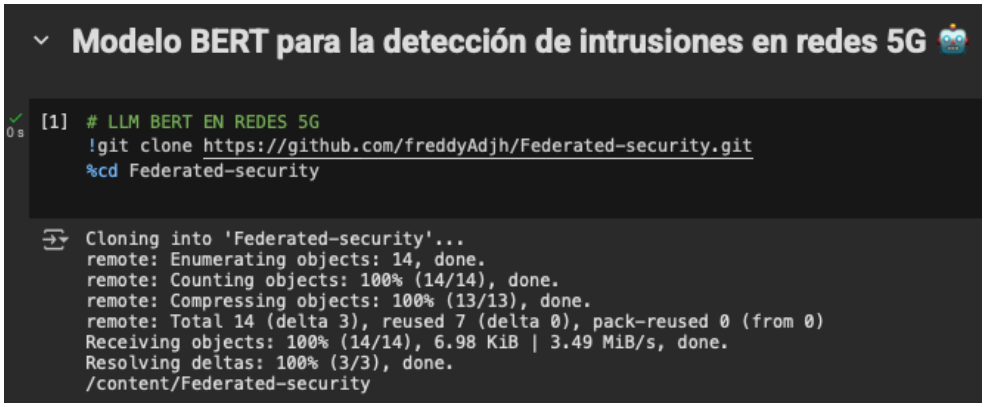


Figura 4- 9 Código de Librerías básicas para la ejecución de BERT por (Adjewa F. y.-B., 2024)

Implementación y pruebas con BERT

Uno de los aportes de SecurityBERT (Adjewa F. y.-B., 2024), es adaptar la tokenización a datos de red, ya que BERT originalmente fue entrenado con texto en lenguaje natural (NLP) (Stryker & Holdsworth, 2024). Para convertir los textos en secuencias de tokens, se utilizó el tokenizador oficial *"bert-base-uncased"* de la librería Transformers de (Hugging Face , s.f.). Este proceso transforma las cadenas de texto en vectores numéricos que sirven como entrada no solo para BERT, sino para cualquier modelo LLM.

4.2.2.3 Preprocesamiento de los datos

Antes de entrenar los modelos de lenguaje seleccionados, se llevó a cabo un proceso de preprocesamiento de datos que garantizará la calidad, homogeneidad y compatibilidad de la información con las arquitecturas de los modelos. Esta fase es fundamental para reducir ruido, eliminar inconsistencias y transformar los datos a un formato interpretable por los modelos.

En la Figura 4- 10 se crea una columna de texto, Bert y otros modelos, para realizar su análisis requieren los datos en formato texto, es por eso por lo que se construye una columna "text" que concatena varias características (protocolo, servicio y estado) para que el modelo pueda procesarlos como tokens.

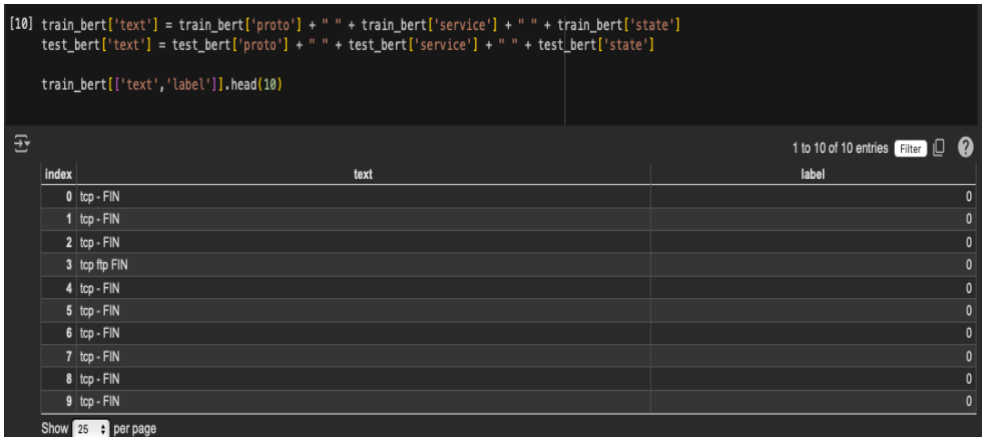


Figura 4- 10 Código de creación de la columna de texto – Preprocesamiento

Se seleccionaron estas características porque estas variables son altamente informativas en la detección de intrusiones, por ejemplo:

- 'proto' (protocolo de red): un protocolo (Cloudflare, s.f.) es un conjunto de reglas para formatear y procesar datos. Los protocolos de red son una lengua franca para los ordenadores. En el dataset esta variable indica que protocolos ha usado cada conexión.
- 'service' (servicio): Representa el servicio o aplicación que corre en el puerto (por ejemplo, HTTP (Microsoft, 2024), FTTP (Lenovo, s.f.), DNS (IBM, 2024)).
- 'state' (estado de la conexión): Define el estado final de la sesión de red, importantísimo en la detección de intrusiones para conocer el estado, si este terminó de forma normal o existió interrupción.

Tokenización

La Figura 4- 11 muestra el código de tokenización que BERT necesita para el análisis de patrones, es decir los registros de red o características seleccionadas en "frases" (por ejemplo: tcp ftp SF), en tokens numéricos que el modelo pueda procesar.

Figura 4- 11 Código de Tokenización – BERT

Esta conversión de tráfico de red a secuencias de tokens constituye la base para que el modelo aprenda patrones y realice la clasificación de tráfico normal o intrusivo.

El tokenizador descompone cada frase en tokens que pertenecen al vocabulario de BERT, añade tokens especiales ([CLS] al inicio y [SEP] al final) y garantiza que todas las secuencias tengan la misma longitud mediante padding.

Una vez completada la tokenización, se estructura los datos en un Dataset personalizado para PyTorch (NVIDIA Corporation, s.f.). Este paso es fundamental, ya que BERT no puede procesar directamente las salidas del tokenizador si no se organizan en el formato correcto. En esta etapa se implementó la clase *BertDataset* (ver Figura 4- 12*),* que toma los tokens generados y las etiquetas de cada muestra y los empaqueta en diccionarios compatibles con el entrenamiento.

Figura 4- 12 Código de clase BertDataset – BERT

Finalizada la tokenización, empieza la etapa de la carga del modelo BERT preentrenado (ver Figura 4- 13) a través de la librería Transformers, configurado específicamente para una tarea de clasificación binaria (tráfico normal vs intrusión), es decir, mediante la ejecución inicial con [num:labels=2], se indica que la capa de salida generará dos posibilidades: una para clase "normal" y otra para clase "ataque".

```
from transformers import BertForSequenceClassification

model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
```

Figura 4- 13 Código de carga del modelo BERT por (Hugging Face , s.f.)

Para garantizar la reproducibilidad de las pruebas, se fijaron semillas en los principales componentes aleatorios del entorno: Python, Numpy y Pytorch. Lo cual segura que el barajado de los lotes sea consistente entre ejecuciones. La Figura 4- 14 muestra el fragmento de código que ayudará que este procedimiento estandarice la iniciación de pesos, la división de datos y el orden del muestreo, reduciendo la variabilidad no controlada en los resultados, mismo que será utilizado en los demás modelos evaluados.

```
import os, random, numpy as np, torch
from transformers import set_seed

SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(SEED)

# para que cuDNN no cambie heurísticas entre ejecuciones
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# utilidad de transformers (afecta generadores internos)
set_seed(SEED)
# ====================================

print("✅ Datos cargados")
print("Entrenamiento:", train_nb15.shape)
print("Pruebas:", test_nb15.shape)

train_nb15.head()
```

Figura 4- 14 Código de configuración de semillas

4.2.2.4 Entrenamiento Supervisado

Entrenamiento del modelo BERT

En esta fase, el modelo BERT se entrena con el dataset UNSW-NB15 utilizando el optimizador AdamW (Gugger & Howard, 2018), recomendado por los desarrolladores de BERT porque maneja de manera más eficiente el ajuste de pesos al aplicar la regularización.

El entrenamiento se realiza en lotes (batches) de 16 muestras (ver Figura 4- 15), esto permite procesar los datos de forma más equilibrada en memoria y mejora la estabilidad del aprendizaje.

```
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
```

Figura 4- 15 Fragmento del código de entrenamiento BERT

El uso de este entrenamiento controlado sigue la metodología observada en el trabajo relacionado de SecurityBERT de (Adjewa F. y.-B., 2024), asegurando que los resultados obtenidos sean reproducibles y comparables. Posteriormente, este mismo proceso se aplicará con los demás modelos seleccionados para el análisis comparativo que se plantea en este estudio.

Para el entrenamiento y debido a la cantidad de datos del dataset, se optó por realizar una prueba inicial del modelo BERT utilizando un subconjunto reducido de este dataset (ver Figura 4-16). Esto se realizó para validar que el pipeline funcionará correctamente, antes de ejecutar el entrenamiento completo.

```
# Reducimos el tamaño del dataset para pruebas rápidas
train_df_small = train_nb15.sample(5000, random_state=42).copy()
test_df_small = test_nb15.sample(5000, random_state=42).copy()

train_df_small['text'] = train_df_small['proto'] + " " + train_df_small['service'] + " " + train_df_small['state']
test_df_small['text'] = test_df_small['proto'] + " " + test_df_small['service'] + " " + test_df_small['state']
```

Figura 4- 16 Código para una muestra reducida del dataset

Tras una espera de algunos minutos (18 minutos) en ejecución, el modelo ejecutó un “epoch” (Lenovo, 2024) de entrenamiento con un lote de 16 muestras [batch_size=16], el modelo alcanzó una función de pérdida “loss” (Omar Elharrouss, 2025) de 0.2741, lo que indica que comenzó a ajustar sus parámetros y a aprender patrones básicos de los datos de red.

```python
from torch.optim import AdamW
import torch
import torch.nn.utils as nn_utils

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

optimizer = AdamW(model.parameters(), lr=5e-5)
MAX_GRAD_NORM = 1.0

model.train()
for epoch in range(1):  # smoke test (1 época)
    for batch in train_loader:
        optimizer.zero_grad()
        outputs = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device),
            labels=batch['labels'].to(device)
        )
        loss = outputs.loss
        loss.backward()
        nn_utils.clip_grad_norm_(model.parameters(), max_norm=MAX_GRAD_NORM)
        optimizer.step()
    print(f"✅ Época {epoch+1} completada — Loss: {loss.item():.4f}")
```

```
✅ Época 1 completada — Loss: 0.2741
```

Figura 4- 17 Código de entrenamiento

La Figura 4- 17 es el código utilizado para la preparación del Dataset para PyTorch, gracias a esta estructuración y siguiendo la metodología planteada en SecurityBERT (Adjewa F. y.-B., 2024), el modelo puede recibir datos uniformes durante el entrenamiento, optimizando el proceso de aprendizaje y permitiendo la replicación de resultados bajo un enfoque reproducible y escalable.

4.2.2.5 Evaluación experimental

Métricas para la evaluación del modelo

Las métricas como: Accuracy, precisión, recall y F1-score (Evidently AI, 2025), permitirán analizar el equilibrio entre falsos positivos y falsos negativos, crucial en un IDS (Sistema de Detección de Intrusiones). Estas métricas permiten cuantificar no solo la cantidad de aciertos globales, sino también la capacidad del modelo para identificar correctamente ataques, cómo se lograría esto, minimizando los (falsos negativos) y evitar alarmas innecesarias reduciendo los (falsos positivos), lo cual es crítico en redes 5G.

En la Figura 4- 18 se muestra el código para un reporte de estas métricas, además se generó una matriz de confusión para visualizar entre clase normal vs ataque. Esta matriz ofrece una representación visual del rendimiento del modelo (clave para comparar con los modelos seleccionados en este estudio).

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Recolectar predicciones y etiquetas reales
all_preds, all_labels = [], []
with torch.no_grad():
    for batch in test_loader:
        outputs = model(batch['input_ids'], attention_mask=batch['attention_mask'])
        preds = torch.argmax(outputs.logits, dim=-1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(batch['labels'].cpu().numpy())

# Reporte detallado
print(classification_report(all_labels, all_preds, target_names=['Normal','Ataque']))

# Matriz de confusión
cm = confusion_matrix(all_labels, all_preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal','Ataque'], yticklabels=['Normal','Ataque'])
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión - BERT')
plt.show()
```

Figura 4- 18 Código para el cálculo de métricas avanzadas

Análisis de BERT

Clase	Precision	Recall	F1-score	Support
Normal	0.74	0.77	0.75	2226
Ataque	0.81	0.79	0.80	2774
Accuracy			0.78	5000
Macro avg	0.77	0.78	0.77	5000
Weighted avg	0.78	0.78	0.78	5000

Tabla 1 Evaluación de modelo BERT con una muestra de 5k registros

La Tabla 1 presenta las métricas clave obtenidas tras evaluar el modelo BERT sobre una muestra representativa de mil registros del conjunto de datos UNSW-NB15. Este análisis exploratorio permitió validar la capacidad inicial del modelo para clasificar correctamente entre tráfico normal y ataques en una red 5G simulada.

Los valores alcanzados de precisión, recall y F1-score superan el 0.75 en ambas clases (normal y ataque), evidenciando un comportamiento robusto incluso con una cantidad reducida de datos.

La clase “ataque” muestra una excelente recuperación (recall) de 0.79 y una F1-score de 0.80, lo cual es prometedor para aplicaciones en sistemas IDS, donde minimizar falsos negativos es esencial.

La consistencia entre los promedios “Micro avg, Macro avg y Weighted avg” (Leung, 2023), todos en 0.79 indican una buena generalización del modelo en esta fase inicial. Aunque las métricas obtenidas son favorables, para validar el comportamiento del modelo en un entorno crítico como el de ciberseguridad en redes 5G, se utilizarán técnicas de Inteligencia Artificial Explicables (XAI) en etapas posteriores.

4.2.2.6 Interpretabilidad

En el contexto de análisis de tráfico 5G, es esencial no solo medir el rendimiento mediante métricas globales como las mostradas en la Tabla 1, sino también observar cómo el modelo clasifica casos específicos. La matriz de confusión (ver Figura 4- 19) obtenida con una muestra de cinco mil registros, permite evidenciar patrones de error que, posteriormente, serán analizados con técnicas XAI para comprender qué características del tráfico influyen en las predicciones erróneas.

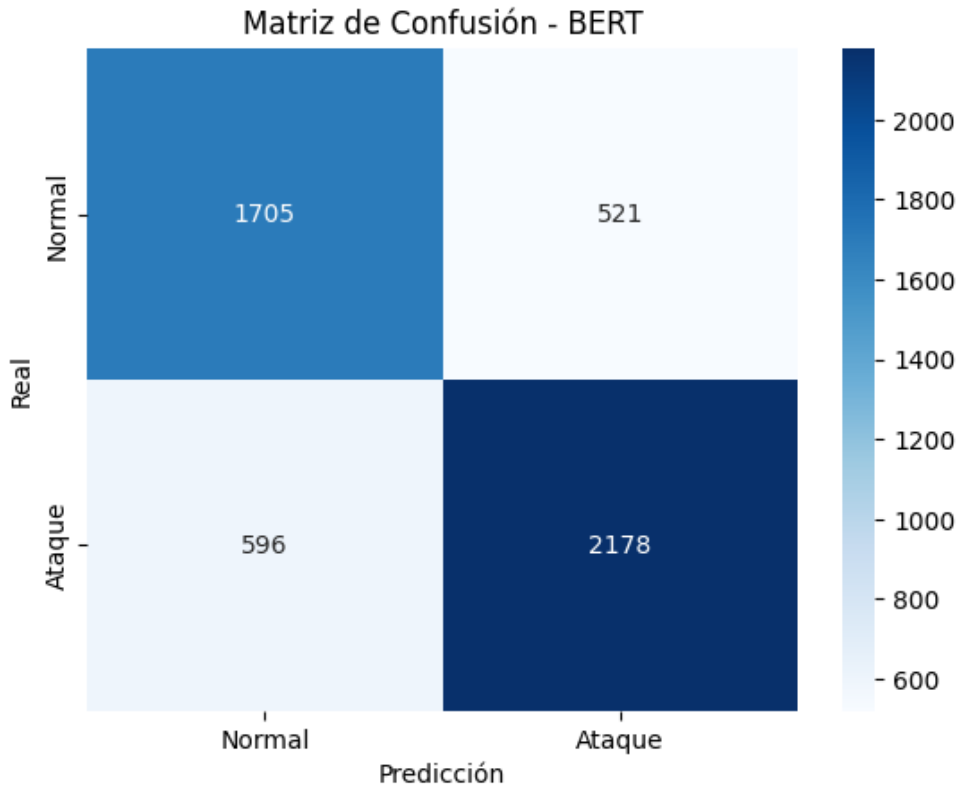


Figura 4- 19 Matriz de confusión sobre una muestra de 5000 registros.

En esta matriz se logra observar en el eje X las predicciones y en el eje Y los valores reales, donde la mayoría de los casos "normales" fueron clasificados correctamente. A continuación, la interpretación de esta matriz:

- **Verdaderos Positivos "TP" (1705):** Casos donde la clase real es "ataque" y el modelo también predijo "ataque". Un alto valor indica que el modelo es eficaz en su función principal (identificar intrusiones).
- **Verdaderos Negativos "TN· (2178):** Casos donde la clase real es "normal" y el modelo predijo correctamente "normal". Un valor alto indica que el modelo no está sobrecargando el sistema con falsos positivos innecesarios.
- **Falsos Positivos "FP" (521):** Casos donde la clase real es "normal" pero el modelo predijo "ataque". Se trata de alertas falsa, es sistemas de producción, un exceso de FP puede reducir la eficiencia y confianza en las alertas.

- **Falsos negativos "FN" (596):** Casos donde la clase real es "ataque" pero el modelo predijo "normal". Este es el tipo de error más crítico, ya que significa ataques que pasan desapercibidos. A estos FN reducirlos es prioritario.

Implementación y pruebas con RoBERTa

Para garantizar la comparabilidad, se replicó el pipeline (Walther, 2023) utilizado con bert sustituyendo únicamente el tokenizador, el modelo preentrenado por RoBERTa, mejoras en PyTorch y entrenamiento. Es decir, se mantendrá los primeros procesos idénticos, obtención de datos, procesamiento, etc.

RoBERTa, al emplear un preentrenamiento más intensivo y Dynamic masking (Richman, 2024), suele ofrecer mejoras de desempeño en tarea de clasificación, por esta razón se ejecutará con registros de 5000. Considerando el mismo Pipeline de BERT, nos enfocaremos en los cambios en los siguientes componentes:

Tokenizador y modelo preentrenado

Se sustituyeron simplemente el tokenizador y el modelo preentrenado por *Roberta-base*, manteniendo intacto el pipeline (ver Figura 4- 20). RoBERTa utiliza *byte-pair encoding y dynamic masking,* lo que trae esto es una mejora en F1 frente a BERT con el mismo esquema de entrenamiento.

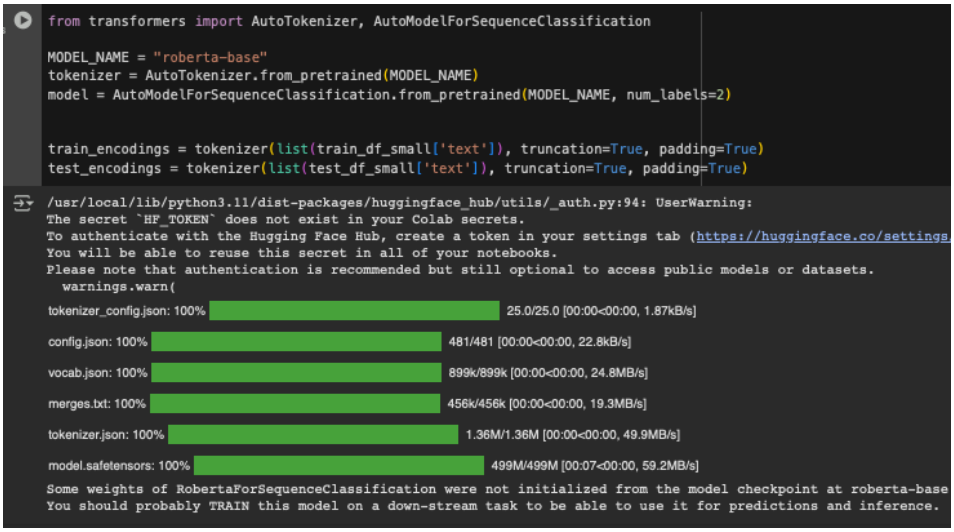


Figura 4- 20 Código del tokenizador y modelo preentrenado de RoBERTa

Pytorch carga de datos

Tras la tokenización, las secuencias se organizaron en una estructura de datos compatible con PyTorch que integra, para cada observación, la representación del texto y su etiqueta de clase. Con el código de la Figura 4- 21 se logra organizar el Dataset y un DataLoader con los mismos parámetros utilizados en BERT.

```python
from torch.utils.data import Dataset, DataLoader
import torch

class TextClsDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __len__(self):
        return len(self.labels)
    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx], dtype=torch.long) for k, v in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx], dtype=torch.long)
        return item

train_dataset_small = TextClsDataset(train_encodings, list(train_df_small['label']))
test_dataset_small  = TextClsDataset(test_encodings,  list(test_df_small['label']))

g = torch.Generator(); g.manual_seed(SEED)  # opcional, para shuffle reproducible
train_loader = DataLoader(train_dataset_small, batch_size=16, shuffle=True, generator=g)
test_loader  = DataLoader(test_dataset_small,  batch_size=16)
```

Figura 4- 21 Código de carga de datos (PyTorch) en modelo RoBERTa

Para garantizar comparabilidad entre modelos, se aplicó una técnica llamada “semilla aleatoria” (SEED) al inicio del código (ver anexos). Esta técnica garantiza que el muestreo, el barajado y la inicialización se mantenga constantes; por tanto, las diferencias observadas entre modelos se atribuyen al modelo mismo y no de cambios en el entrenamiento.

Entrenamiento corto RoBERTa (prueba de 5000 registros)

Se mantuvo el mismo procedimiento utilizado con el modelo anterior, empleando el optimizador AdamW con los mismos parámetros y una época inicial de verificación para validar el pipeline (ver Figura 4- 22). Adicionalmente, se aplicó *gradient clipping* (DeepAI, s.f.) para evitar explosión de gradientes en lotes de alta varianza.

Entrenamiento corto RoBERTa

```python
from torch.utils.data import DataLoader
from torch.optim import AdamW
import torch
import torch.nn.utils as nn_utils

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

train_loader = DataLoader(train_dataset_small, batch_size=16, shuffle=True)  # o con generator=g
optimizer = AdamW(model.parameters(), lr=5e-5)
MAX_GRAD_NORM = 1.0  # clipping típico para Transformers

model.train()
for epoch in range(1):  # 1 época “smoke test”; luego 3–5 para resultados estables
    for batch in train_loader:
        optimizer.zero_grad()
        outputs = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device),
            labels=batch['labels'].to(device)
        )
        loss = outputs.loss
        loss.backward()

        # ✅ Gradient clipping (nuevo)
        nn_utils.clip_grad_norm_(model.parameters(), max_norm=MAX_GRAD_NORM)

        optimizer.step()
    print(f"✅ Época {epoch+1} completada — Loss: {loss.item():.4f}")
```

```
✅ Época 1 completada — Loss: 0.3929
```

Figura 4- 22 Código de entrenamiento LLM RoBERTa

El entrenamiento se ejecutó en GPU (Intel Corporation, s.f.) y tuvo una duración aproximada de 18 minutos, coincidiendo con el tiempo registrado en la ejecución de BERT bajo las mismas condiciones. Y con una pérdida final de 0.3929, relativamente bajo para una primera época en un subconjunto de 5000 registros.

Evaluación del modelo y métricas avanzadas

La Figura 4- 23 muestra que RoBERTa aplica el mismo procedimiento que la fase de verificación de BERT, utilizando un Dataloader sin barajado y un tamaño de lote de 16. Este modelo alcanzó una precisión de 76.78%, valor que constituye un rendimiento inicial relevante considerando que se trata de un entrenamiento de una sola época sobre una muestra de 5000 registros. No obstante, este valor podría mejorarse aumentando épocas, ajustar la tasa de aprendizaje, balancear clases o ajuste de hiperparámetros. Todo esto se observará en la fase final de comparación entre modelos.

```
model.eval()

# mismo batch_size que en BERT; sin shuffle
test_loader = DataLoader(test_dataset_small, batch_size=16)

correct, total = 0, 0
with torch.no_grad():
    for batch in test_loader:
        outputs = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device)
        )
        preds = torch.argmax(outputs.logits, dim=1)

        # comparamos en CPU contra las etiquetas tal como hiciste en BERT
        correct += (preds.cpu() == batch['labels']).sum().item()
        total   += batch['labels'].size(0)

accuracy = correct / total
print(f"📊 Precisión del modelo RoBERTa en UNSW-NB15: {accuracy:.2%}")
```

Figura 4- 23 Código de evaluación de LLM RoBERTa

El siguiente bloque de código (ver Figura 4- 24) ejecuta el proceso de evaluación avanzada del modelo RoBERTa sobre el conjunto de prueba, obteniendo métricas de desempeño por clase y globales, así como la matriz de confusión correspondiente.

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import torch
import numpy as np

model.eval()  # RoBERTa ya cargado
all_preds, all_labels = [], []

with torch.no_grad():
    for batch in test_loader:
        outputs = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device)
        )
        preds = torch.argmax(outputs.logits, dim=-1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(batch['labels'].cpu().numpy())

print(classification_report(all_labels, all_preds,
                            target_names=['Normal','Ataque'],
                            digits=2, zero_division=0))

# Matriz de confusión (cuentas absolutas)
cm = confusion_matrix(all_labels, all_preds, labels=[0,1])
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal','Ataque'],
            yticklabels=['Normal','Ataque'])
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión - RoBERTa')
plt.show()
```

Figura 4- 24 Còdigo métricas avanzadas LLM RoBERTa

El procedimiento recolecta las predicciones del modelo y las compara con las etiquetas reales, generando valores que permiten cuantificar la capacidad del clasificador para identificar correctamente instancias de tráfico normal y de intrusión.

Análisis RoBERTa

Clase	Precision	Recall	F1-score	Support
Normal	0.72	0.78	0.75	2226
Ataque	0.81	0.76	0.78	2774
Accuracy			0.77	5000
Macro avg	0.77	0.77	0.77	5000
Weighted avg	0.77	0.77	0.77	5000

Tabla 2 Evaluación de modelo RoBERTa con una muestra de 5k registros

En la Tabla 2, el modelo RoBERTa alcanza una precisión global del 77% (accuracy), con promedios macro y ponderados (Macro avg y Weighted avg) tambien de 0.77, lo que indica un desempeño equilibrado entre ambas clases.

En la clase “Normal”, se observa una precisión de 0.72 y un recall de 0.78, reflejando que el modelo identifica correctamente la mayoría de las instancias normales, aunque aún presenta falsos positivos.

En la clase “ataque”, la precisión asciende a 0.81 y el recall a 0.76, evidenciando una capacidad ligeramente mayor para discriminar intentos de intrusión. El F1-score en ambas clases es homogéneo (0.75 y 0.78 respectivamente), lo que indica un balance adecuado entre precisión y exhaustividad en cada categoría.

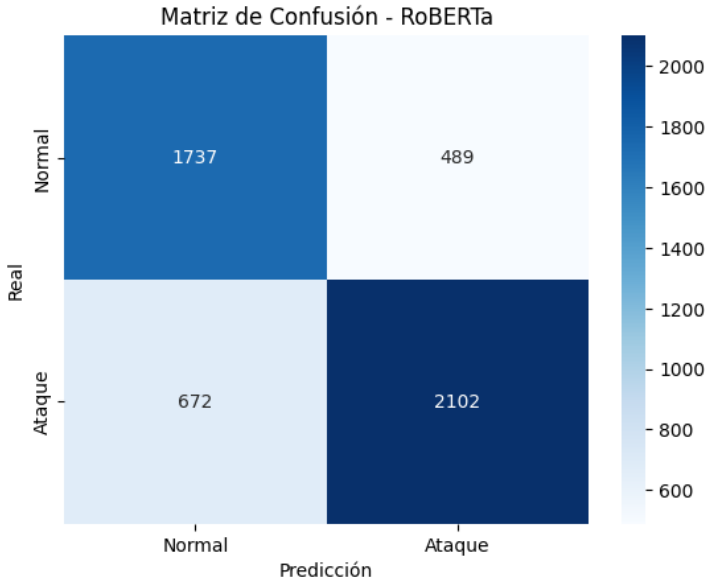


Figura 4- 25 Matriz de confusión LLM RoBERTa

La matriz de confusión (ver Figura 4- 25) permite visualizar la distribución de aciertos y errores del modelo, a continuación, la interpretación:

- **Verdaderos Negativos “TN” (1737):** Casos donde la clase real es “normal” y el modelo también predijo “normal”. Un valor alto indica que el sistema no está generando excesivas alertas falsas.
- **Falsos positivos “FP” (489):** Casos donde la clase real es “normal” pero el modelo predijo “ataque”. Corresponde a alarma falsas; en operación, un exceso de FP puede degradar la eficiencia y la confianza del equipo de seguridad.
- **Falsos negativos “FN” (672):** Casos donde la clase real es “ataque” pero el modelo predijo “normal”. Este es el error más crítico, porque implica intrusiones que pasan inadvertidas; reducir FN es prioritario.
- **Verdaderos positivos “TP” (2102):** Casos donde la clase real es “ataque” y el modelo también predijo “ataque”. Un valor elevado sugiere buena capacidad de detección de intrusiones.

Implementación y pruebas con GPT

La implementación de GPT en este estudio se inspira o toma de referencia la propuesta de NIDS-GPT desarrollada por (Huang, 2024), en la cual se explora la oficia de modelos autorregresivos en Sistemas de Detección de Intrusiones en Redes (NIDS). A diferencia de arquitecturas bidireccionales como BERT y RoBERTa, GPT se basa en un enfoque unidireccional autorregresivo, lo que permite modelar dependencias contextuales secuenciales de manera distinta. Este contraste arquitectónico representa un aporte relevante para la comparabilidad, ya que permite evaluar cómo varía el rendimiento de un modelo generativo en un escenario de clasificación supervisada.

Para este LLM, se mantuvieron los mismos hiperparámetros utilizados en los modelos anteriores, su única variación corresponderá a los bloques de códigos en el tokenizador e inicialización del modelo.

Se puntualizará las ejecuciones que serán similares y se detallará donde existe modificaciones, a continuación, los bloques similares:

- Librerías
- Carga de datos
- Procesamiento de los datos (formato de texto)
- Configuración de semillas

Bloques con modificaciones:

Tokenización LLM GPT

```
# === TOKENIZADOR GPT-2
from transformers import GPT2TokenizerFast, GPT2ForSequenceClassification

MAX_LEN = 128
gpt_tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")
gpt_tokenizer.pad_token = gpt_tokenizer.eos_token
gpt_tokenizer.padding_side = "right"

train_encodings_gpt = gpt_tokenizer(
    train_df_small["text"].tolist(),
    truncation=True, padding="max_length", max_length=MAX_LEN
)
test_encodings_gpt = gpt_tokenizer(
    test_df_small["text"].tolist(),
    truncation=True, padding="max_length", max_length=MAX_LEN
)
# carga del modelo
model = GPT2ForSequenceClassification.from_pretrained("gpt2", num_labels=2)
model.config.pad_token_id = gpt_tokenizer.pad_token_id
```

Figura 4- 26 Código de tokenización LLM GPT

La Figura 4- 26 muestra el bloque de código para la tokenización y carga de datos necesaria para el modelo GPT, la diferencia con los demás modelos es que GPT emplea el tokenizador propio del modelo *GPT2TokenizerFast* (Navarro, 2025) lo que ajusta la segmentación de secuencias al formato GPT. Y con respecto al modelo, se emplea la arquitectura GPT-2, adaptada para clasificación de secuencias.

PyTorch carga de datos

La carga de datos tokenizados tiene una estructura similar, salvo la variable "train_encodings:gpt y test_encoding_gpt" que se creó en el bloque de tokenización GPT. Podemos observar en la Figura 4- 27 su código de ejecución.

```python
from torch.utils.data import Dataset, DataLoader
import torch

class TextClsDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __len__(self):
        return len(self.labels)
    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx], dtype=torch.long) for k, v in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx], dtype=torch.long)
        return item

train_dataset_small = TextClsDataset(train_encodings_gpt, list(train_df_small['label']))
test_dataset_small  = TextClsDataset(test_encodings_gpt,  list(test_df_small['label']))

g = torch.Generator(); g.manual_seed(SEED)  # opcional, para shuffle reproducible
train_loader = DataLoader(train_dataset_small, batch_size=16, shuffle=True, generator=g)
test_loader  = DataLoader(test_dataset_small,  batch_size=16)
```

Figura 4- 27 Código Pytorch LLM GPT

Entrenamiento corto GPT (prueba de 5000 registros)

La Figura 4- 28 muestra el código de entrenamiento de GPT, en él se observa que se mantiene la estructura y configuración empleada en los modelos anteriores, manteniendo optimizador, épocas, gradientes, etc. Lo que asegura la comparación directa de los resultados sin introducir variaciones en los parámetros.

```python
from torch.optim import AdamW
import torch
import torch.nn.utils as nn_utils

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

LR = 5e-5
MAX_GRAD_NORM = 1.0

optimizer = AdamW(model.parameters(), lr=LR)

model.train()
for epoch in range(1):
    for batch in train_loader:  # ya creado con generator=g y batch_size=16
        optimizer.zero_grad()
        outputs = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device),
            labels=batch['labels'].to(device)
        )
        loss = outputs.loss
        loss.backward()
        nn_utils.clip_grad_norm_(model.parameters(), max_norm=MAX_GRAD_NORM)
        optimizer.step()
    print(f"✅ Época {epoch+1} completada — Loss: {loss.item():.4f}")
```

```
✅ Época 1 completada — Loss: 0.3199
```

Figura 4- 28 Código de entrenamiento LLM GPT

El entrenamiento se ejecutó por 1 época (epoch), con un valor de pérdida (loss) de 0.3199, lo cual indica una rápida convergencia inicial y un ajuste adecuado del modelo sobre el conjunto de entrenamiento.

Dado el alto costo computacional de entrenar modelos de lenguaje en Google Colab con recursos gratuitos, se contrató un plan con mayores recursos, reduciendo los tiempos de una hora a cerca de un minuto por época, un aspecto clave para sistemas de detección de intrusos en 5G en tiempo real.

Evaluación del modelo y métricas avanzadas

```python
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np, seaborn as sns, matplotlib.pyplot as plt
import torch

model.eval()

# Accuracy (precisión global)
correct, total = 0, 0
with torch.no_grad():
    for batch in test_loader:  # sin shuffle
        out = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device)
        )
        preds = torch.argmax(out.logits, dim=1)
        correct += (preds.cpu() == batch['labels']).sum().item()
        total   += batch['labels'].size(0)
acc = correct / total
print(f"📊 Precisión (accuracy) del modelo GPT en UNSW-NB15: {acc:.2%}")

# Reporte detallado y matriz de confusión
all_preds, all_labels = [], []
with torch.no_grad():
    for batch in test_loader:
        out = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device)
        )
        preds = torch.argmax(out.logits, dim=-1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(batch['labels'].cpu().numpy())

print(classification_report(
    all_labels, all_preds,
    target_names=['Normal','Ataque'],
    digits=2, zero_division=0
))

cm = confusion_matrix(all_labels, all_preds, labels=[0,1])
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal','Ataque'], yticklabels=['Normal','Ataque'])
plt.xlabel('Predicción'); plt.ylabel('Real'); plt.title('Matriz de Confusión - GPT')
plt.show()
```

```
📊 Precisión (accuracy) del modelo GPT en UNSW-NB15: 77.40%
              precision    recall  f1-score   support

      Normal       0.74      0.77      0.75      2226
      Ataque       0.81      0.78      0.79      2774

    accuracy                           0.77      5000
   macro avg       0.77      0.77      0.77      5000
weighted avg       0.78      0.77      0.77      5000
```

Figura 4- 29 Código para métricas avanzadas LLM GPT

La Figura 4- 29 muestra el bloque de métricas de GPT y se observa la misma estructura utilizada en los modelos anteriores, lo que asegura la comparabilidad de resultados. Existen ligeras diferencias, pero debido a las arquitecturas de cada modelo y no a cambios metodológicos en la evaluación.

La evaluación del modelo GPT sobre el conjunto de prueba mostró los siguientes resultados:

Análisis GPT

Clase	Precision	Recall	F1-score	Support
Normal	0.74	0.77	0.75	2226
Ataque	0.81	0.78	0.79	2774
Accuracy			0.77	5000
Macro avg	0.77	0.77	0.77	5000
Weighted avg	0.78	0.77	0.77	5000

Tabla 3 Evaluación de modelo GPT con una muestra de 5k registros

GPT, incluso con un entrenamiento corto de una época, alcanzó un rendimiento comparable con BERT y RoBERTa en el mismo escenario experimental.

Si bien los FP (Falsos positivos) (ver Figura 4- 30) representan un reto al generar alertas innecesarias, lo más crítico son los FN (Falsos Negativos), ya que implican ataques no detectados. Si bien GPT alcanzó un rendimiento competitivo bajo las condiciones de entrenamiento establecidas, el modelo podría mejorar significativamente su precisión y reducir tanto los falsos positivos como los falsos negativos en la fase final de experimentación.

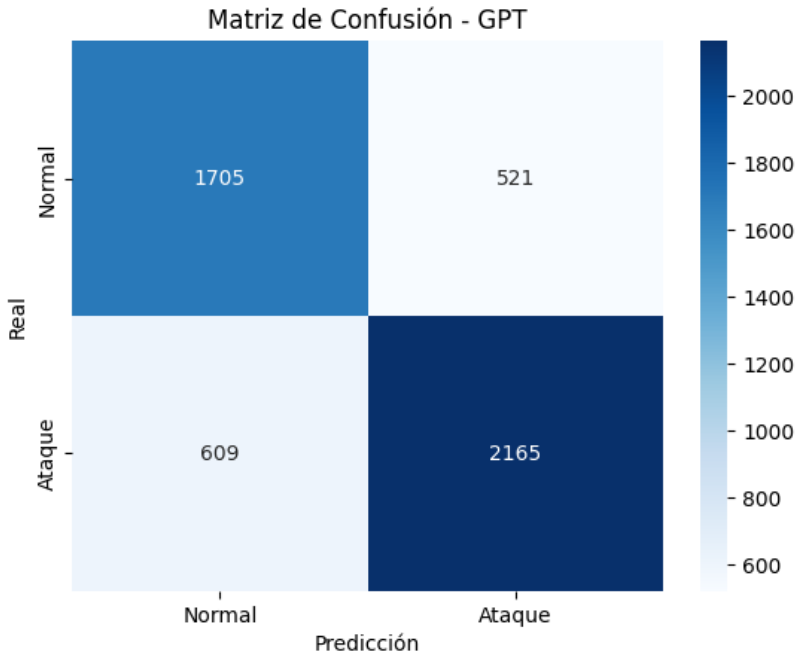


Figura 4- 30 Matriz de confusión LLM GPT

Estos hallazgos coinciden parcialmente con el proyecto NIDS-GPT de (Huang, 2024), donde se destaca el potencial de modelos basados en GPT para tareas de detección de intrusiones, aunque no se ha usado configuraciones de hiperparámetros que destaca en el trabajo de NIDS-GPT, pero en lo posterior se lo hará para maximizar su rendimiento.

4.2.3 Herramientas tecnológicas utilizadas

- **Lenguaje de programación**: Python 3.x
- **Frameworks de deep learning**: PyTorch, TensorFlow
- **Bibliotecas de NLP y LLM**: Hugging Face Transformers, Tokenizers
- **Manejo de datos y métricas**: Pandas, Scikit-learn
- **Visualización de resultados**: Matplotlib, Seaborn
- **Entornos de desarrollo**: Google Colab y/o Jupyter Notebook
- **Dataset principal**: UNSW-NB15 (conjuntos experimentales reales de tráfico de red con ataques e intrusiones)

4.3 Recursos requeridos

- **Recursos técnicos**
  - Hardware

  1 laptop MacBook Pro:

  Procesador: 2,4 GHz Intel Core i5 de 4 núcleos

  Gráficos: Intel Iris Plus Graphics 655 1536 MB

  Memoria: 8 GB 2133 MHz LPDDR3

  Pantalla: 13,3”

  1 monitor (adicional)

  Modelo: Xiaomi P27FBB-RGGL

  Tamaño: 27”

  Resolución: 1920 x 1080

  Frecuencia: 165 Hz

  Tiempo de respuesta: 1 ms (typ)

  - Software

  Sistema operativo: macOS Sequoia (version 15.4.1)

  Entorno de desarrollo: Jupyter Notebook, Google colab, Visual Studio Code.

Asistencia bibligráfica: Google, Google academic, Biblioteca CRAI Universidad Europea de Madrid.

Otros: Herramientas IA (ChatGPT), para consultas especificas, asesoramiento para mejora de sintáxis en texto.

4.4 Presupuesto

Tipo de coste	Valor	Comentarios
Horas de trabajo en el proyecto	350horas * 3,05 $/hora = $1067,50	Tiempo dedicado al desarrollo, análisis de datos y reuniones con el tutor.
Equipo técnico utilizado	$ 319,00	Valor aproximado de una MacBook pro-2019, similar al mercado
Software utilizado	$ 0,00	Herramientas de código abierto: PyThon, TensorFlow, PyTorch. Etc.
Estudios e informes	$ 0,00	No se ha adquirido ningún informe adicional o suscripciones.
Materiales empleados	$ 0,00	No se requirieron materiales físicos o de laboratorio.

4.5 Viabilidad

4.5.1 Viabilidad Técnica

El proyecto ha sido desarrollado utilizando herramientas de código abierto y ampliamente adaptadas en el ámbito del aprendizaje profundo y el procesamiento de lenguaje natural, como Python, Pytorch, Tensorflow y bibliotecas de Hugging Face. Además, el uso de plataformas como Goggle Colab ha permitido disponer de entornos con recursos computacionales (CPU/GPU) sin coste, lo cual ha facilitado la ejecución de modelos de gran tamaño sin necesidad de infraestructura física dedicada. El conjunto de datos empleado, UNSW-NB15, es público y ha sido adaptado mediante procesos de preprocesamiento compatibles con los modelos LLM seleccionados.

4.5.2 Viabilidad económica

El coste económico del proyecto ha sido mínimo, ya que no ha sido necesario adquirir software, licencias ni hardware especifico. El principal recurso invertido ha sido el tiempo dedicado al análisis, desarrollo, pruebas y documentación, estimado en aproximadamente 350 horas. Esta inversión se justifica plenamente por el valor académico del trabajo y la capacitación técnica

obtenida. La estructura modular del entorno desarrollado también permite reutilizar componentes para proyectos futuros sin incurrir en nuevos costes significativos.

4.5.3 Sostenibilidad del proyecto

El diseño del entorno experimental permite la incorporación de nuevos modelos, datasets o métricas sin necesidad de rediseñar por completo la arquitectura base. Esta característica favorece la sostenibilidad y escalabilidad del proyecto, ya sea en investigaciones académicas posteriores o en aplicaciones prácticas en entornos reales de ciberseguridad. Asimismo, el enfoque reproducible y basado en tecnologías ampliamente utilizadas facilita su transferencia y adopción en otras iniciativas relacionadas con la seguridad en redes móviles 5G.

4.6 Resultados del proyecto

Los experimentos de los modelos seleccionados se ejecutan bajo un pipeline homogéneo:

- Mismo Dataset: UNSW-NB15
- Misma construcción de secuencias textuales (proto, service, state)
- Fijación de semillas
- Mismo esquema de entrenamiento (optimizador AdamW, épocas y batch size)
- Y protocolo de evaluación: accuraccy, precisión, recall y F1-macro con matriz de confusión.

Tal como se estableció en la metodología, solo varían los bloques estrictamente necesarios:

- Tokenizador
- CheckPoint por modelo

Esta decisión es coherente con el enfoque reproducible adoptado en este estudio, y para interpretar estos resultados, es indispensable apoyarse en métricas de evaluación que permitan medir no solo el desempeño global del sistema, sino también su efectividad real en la detección de intrusiones, minimizando falsos positivos y garantizando que los ataques no pasen desapercibidos.

Las métricas de validación que se han utilizado son las siguientes:

- **Accuracy (Exactitud):** Es la métrica más sencilla y pfrece una visión general del rendimiento, calculada como la proporción de predicciones correctas sobre el total. En el contexto de intrusiones, la accuracy refleja el índice global de aciertos al clasificar tanto el tráfico normal como malicioso.

$$accuracy = \frac{\text{№ } predicciones\ correctas}{\text{№ } predicciones\ totales} = \frac{TP + TN}{TP + TN + FP + FN}$$

  Una alta exactitud indica que la mayoría de los flujos de red se clasificaron correctamente, aunque puede resultar engañosa si existe un fuerte desbalance entre clases (muchos más registros normales que ataque).

- **F1-score:** Combina en una sola medida la precisión y la sensibilidad (recall):

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

En detección de intrusiones, el F1-score es crucial porque equilibra la capacidad del modelo para detectar intrusiones reales (recall) y para no generar demasiadas falsas alarmas (precisión). Un valor cercano a 1 indica un sistema robusto en ambos aspectos.

- **Precision (Precisión):** Evalúa cuantas de las predicciones positivas hechas por el modelo corresponden afectivamente a intrusiones.

$$Precision = \frac{TP}{TP + FP}$$

En este contexto, una alta precisión significa que cuando el sistema alerta una intrusión, es muy probable que sea realmente un ataque y no un tráfico normal confundido.

- **Recall (Sensibilidad o Exhaustividad):** Mide la proporción de intrusiones correctamente detectadaspor el modelo frente al total de intrusiones presentes en el dataset.

$$Recall = \frac{TP}{TP + FN}$$

En detección de intrusiones, un alto recall asegura que el sistema no deje pasar ataques sin ser detectados. Sin embargo, puede estar acompañado de más falsos positivos si no se equilibra con la precisión.

Finalmente, no solo se debe medir el rendimiento cuantitativo de los modelos, sino también comprender qué atributos del tráfico de red influyen en sus predicciones, se incorporan técnicas de Inteligencia Artificial Explicable (XAI). Estas permitirán analizar la atención de los modelos y visualizar qué características de los paquetes resultan determinantes para clasificar un registro como normal o como intrusión.

4.6.1 Resultados de LLM BERT

En la experimentación se configuró el modelo BERT con un máximo de 5 épocas, con el fin de observar el comportamiento de loss (pérdida) a lo largo de varias interacciones. Los resultados mostraron que el modelo alcanzó su mejor desempeño en la primera época.

- ✓ Época 1 – Loss: 0.2676

Esto demuestra un buen índice de desempeño del modelo entrenado.

Evaluación del modelo

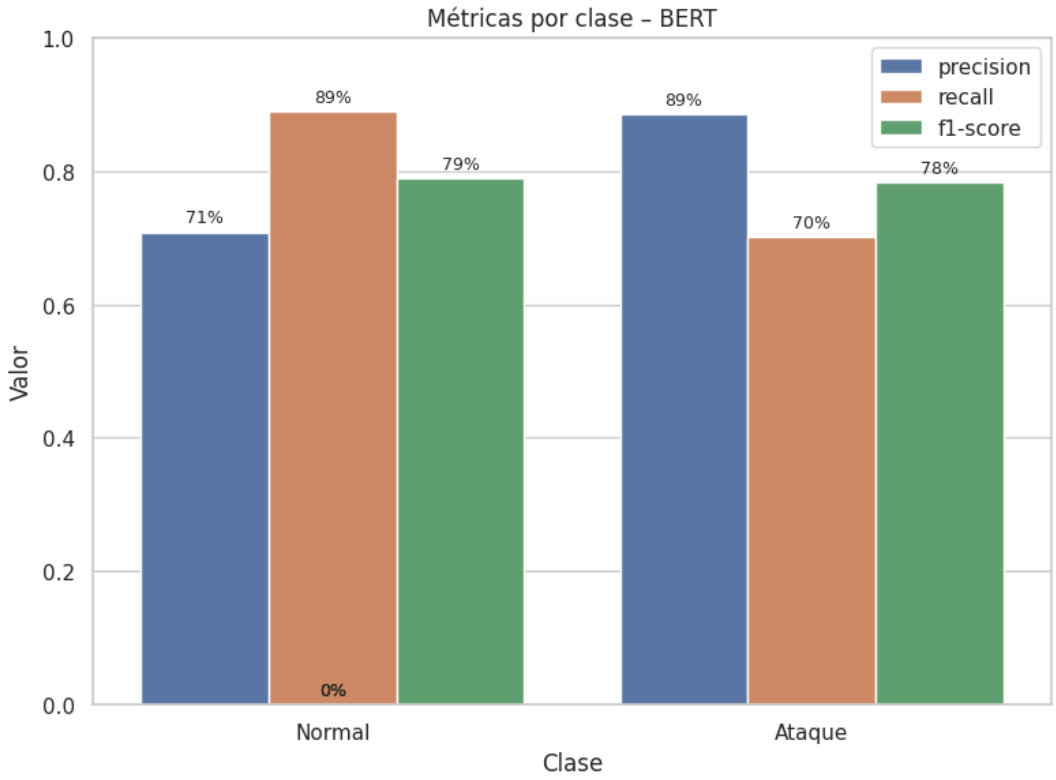


Figura 4- 31 Métricas de evaluación por clase LLM BERT

El modelo BERT logra un buen equilibrio entre detectar tráfico normal y tráfico malicioso en un escenario de red 5G.

- Fortalezas: alta precisión al identificar ataques 0.89 (Barra azul Figura 4- 31 de la clase ataque)
- Aspectos para mejorar: aumentar la sensibilidad a ataques para reducir los que se escapan. Un recall de 0.70 (Barra naranja Figura 4- 31 de la clase ataque)

En un entorno real de red, este desempeño ya es útil como IDS (Sistema de Detección de Intrusiones), pueden existir mejoras implementando otras configuraciones o reglas.

El modelo BERT alcanzó un desempeño consistente en todas las métricas evaluadas:

	precision	recall	f1-score	support
Normal	0.71	0.89	0.79	37000
Ataque	0.89	0.70	0.78	45332
accuracy			0.79	82332
macro avg	0.80	0.80	0.79	82332
weighted avg	0.81	0.79	0.79	82332

- **Accuracy (79%):** el modelo clasifica correctamente casi 8 de 10 registros, lo que refleja un buen rendimiento general.
- **F1 macro (79%):** el promedio del F1 entre las clases "normal y ataque" muestra que el modelo mantiene un equilibrio en su capacidad para ambas, sin favorecer de forma marcada a una sobre la otra.
- **F1 weighted (79%):** al ponderar por la proporción de ejemplos de cada clase, se confirma que el modelo conserva un rendimiento estable incluso considerando el desbalance natural de los datos.

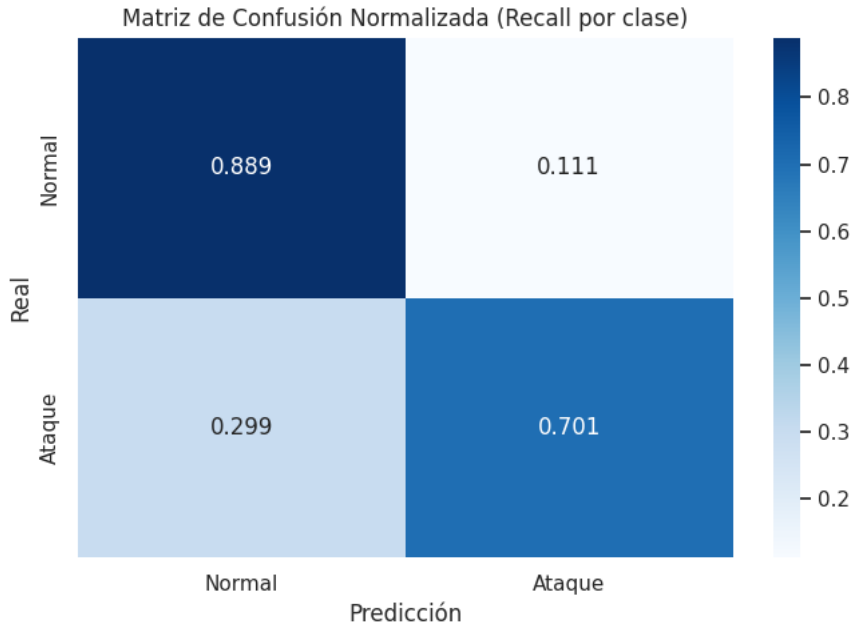


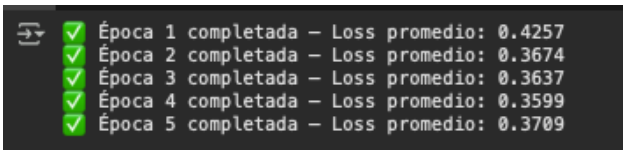
Figura 4- 32 Matriz de confusión BERT datasetfull

La Figura 4- 32 muestra la matriz de confusión e indica que para el tráfico "normal", el modelo alcanza una tasa de acierto de 88,9%, lo que indica una alta capacidad de identificar correctamente patrones legítimos de comunicación. No obstante, un 11,1% de estas instancias fueron clasificadas erróneamente como ataques, esto generan alertas innecesarias.

En el tráfico de "ataque", el modelo identifica correctamente un 70,1% de las instancias, mientras que el 29,9% restantes corresponden a falsos negativos, es decir, ataques no detectados, factor importante que sugiere una mejora en entornos de ciberseguridad, donde no detectar un ataque puede ser significativo.

4.6.2 Resultados de LLM GPT

En la experimentación se configuró el modelo GPT-base con un máximo de 5 épocas, con el fin de observar el comportamiento de loss (pérdida) a lo largo de varias interacciones. Los resultados mostraron que el modelo alcanzó su mejor desempeño en la cuarta época.



- ✓ Época 4 – Loss: 0.3599

Esto demuestra un buen índice de desempeño del modelo entrenado.

Evaluación del modelo

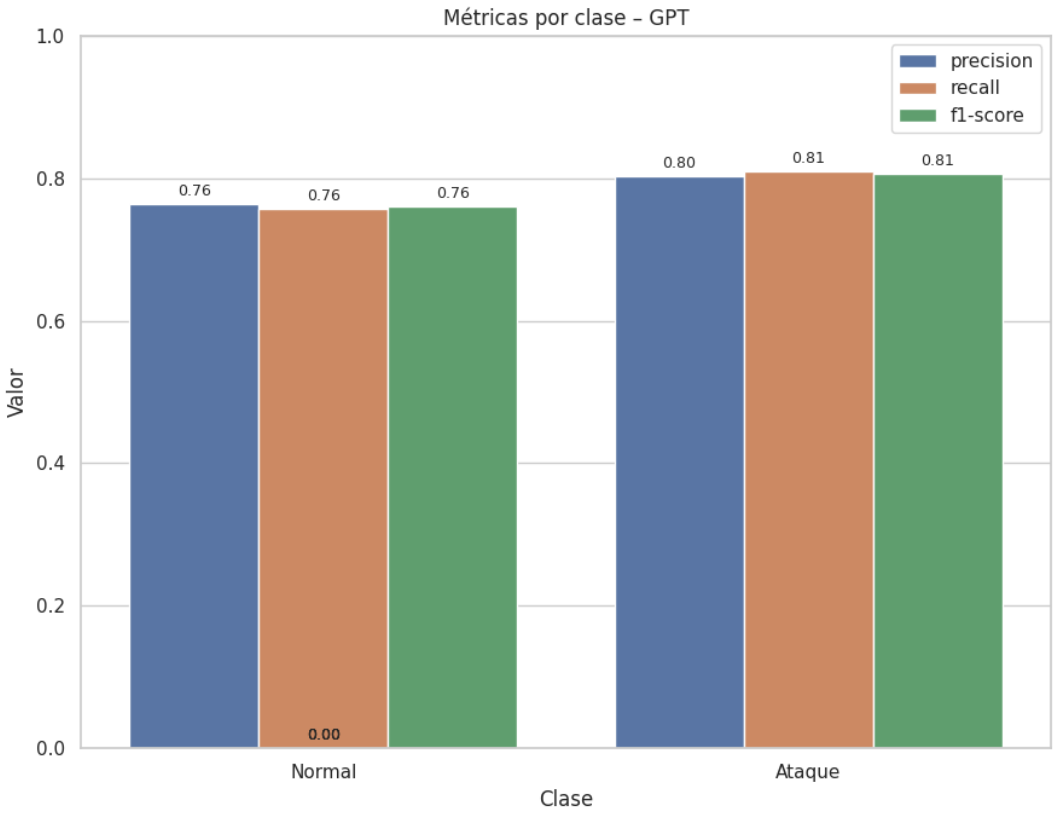


Figura 4- 33 Métricas de evaluación por clase LLM GPT

El modelo GPT logra un desempeño robusto y equilibrado en la clasificación de tráfico normal y malicioso en un entorno de red 5G.

- Fortalezas: alta sensibilidad para identificar ataques (recall=0.81, ver Figura 4- 33 barra naranja de la clase ataque), lo que reduce de manera importante el riesgo de falsos negativos y aumenta la capacidad de respuestas frente a intrusiones.

- Aspectos para mejorar: aunque la precisión en ataques (0.80, barra azul) es adecuada, se podría optimizar aún más para disminuir la proporción de tráfico normal clasificado erróneamente como ataque.

En un entorno real de red, este comportamiento es particularmente valioso como IDS, ya que prioriza la detección temprana de amenazas. En términos de ciberseguridad, es preferible contar con algunas alertas adicionales (falsos positivos) que permitan proteger la red, antes que dejar escapar intrusiones que comprometan la seguridad del servicio.

El modelo GPT alcanzó un desempeño consistente en todas las métricas evaluadas:

Precisión (accuracy) del modelo GPT en UNSW-NB15: 77.53%

	precision	recall	f1-score	support
Normal	0.76	0.76	0.76	37000
Ataque	0.80	0.81	0.81	45332
accuracy			0.79	82332
macro avg	0.78	0.78	0.78	82332
weighted avg	0.79	0.79	0.79	82332

- **Accuracy (79%):** Al igual que BERT, el modelo GPT clasifica correctamente casi 8 de 10 registros, lo que refleja un buen rendimiento general.
- **F1 macro (78%):** el promedio del F1 entre las clases “normal y ataque” muestra que el modelo mantiene un equilibrio en su capacidad para ambas, con una ligera ventaja en la clase Ataque, donde logra una mayor sensibilidad.
- **F1 weighted (79%):** al ponderar por la proporción de ejemplos de cada clase, se confirma que el modelo conserva un rendimiento estable incluso considerando el desbalance natural de los datos.

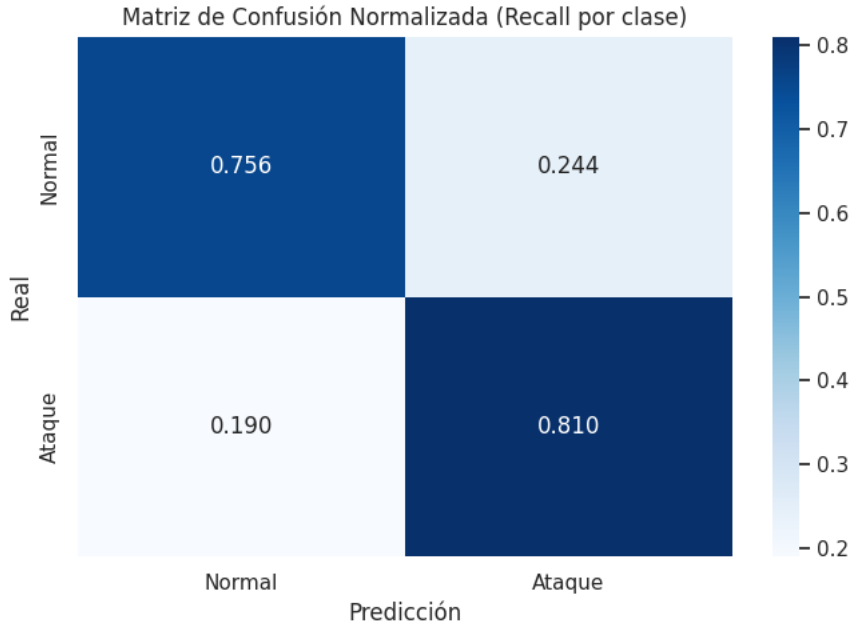


Figura 4- 34 Matriz de confusión GPT datasetfull

En la Figura 4- 34 se muestra la matriz de confusión normalizada para el modelo GPT. En el tráfico "normal", el modelo alcanza una tasa de acierto del 75,6%, lo que indica una buena capacidad de identificar correctamente tráfico normal. Sin embargo, un 24,4% de las instancias fueron clasificadas erróneamente como ataques, lo que podría generar alertas innecesarias en un entorno real de red.

En el tráfico de "ataque", el modelo identifica correctamente el 81,0% de las instancias, mientras que el 19,0% restante corresponde a falsos negativos, es decir, ataques que no fueron detectados. Aunque esta cifra aún representa un riesgo en ciberseguridad, el desempeño supera al de BERT en este aspecto.

4.6.3 Resultados de LLM RoBERTa

En la experimentación se configuró RoBERTa con un máximo de 7 épocas, al principio se configuró con 5 pero dada las observaciones, se apreció una convergencia estable del valor de pérdida (loss) desde 0.4258 hasta 0.3607, lo que indica un aprendizaje efectivo sin señales de sobreajuste o así lo muestra el gráfico de evolución del Loss en el entrenamiento de RoBERTa. (ver Figura 4- 35)

Figura 4- 35 Evolución de loss entrenamiento LLM RoBERTa

Evaluación del modelo

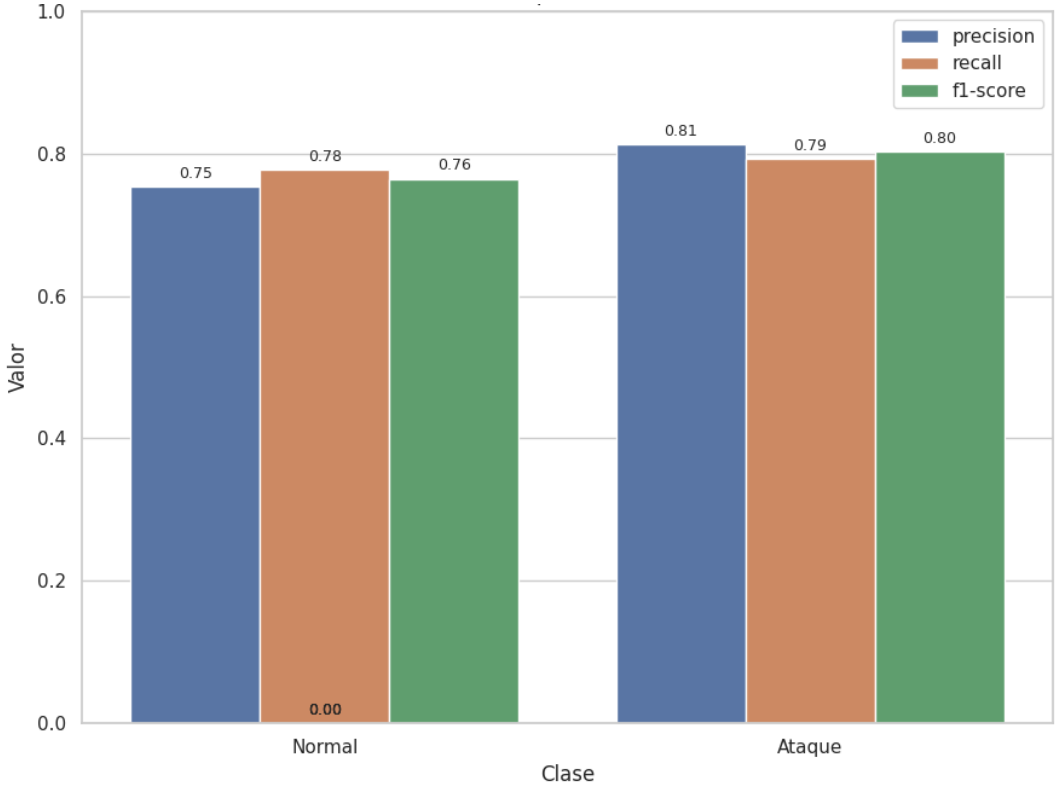


Figura 4- 36 Métricas de evaluación por clase LLM RoBERTa

A fecha de este estudio, no se han encontrado publicaciones académicas que apliquen RoBERTa directamente a datos de tráfico de red en entornos 5G para tareas de detección de intrusiones, lo que convierte este estudio como una propuesta innovadora en la convergencia entre técnicas avanzadas de PLN y los desafíos emergentes de la ciberseguridad en redes de nueva generación.

- Fortalezas: Al igual que GPT, el modelo RoBERTa demostró una alta sensibilidad para identificar tráfico malicioso, alcanzando un recall= 0.79 y un G1-score= 0.80 en la clase "ataque", lo cual es crucial para minimizar el riesgo de falsos negativos en IDS. Además, la precisión de 0.81 en la detección de ataques (barra azul de la clase ataque, ver Figura 4- 36) refuerza la confianza en la validez de las alertas.
- Aspectos que mejorar: la clase "normal" mostró una precisión del 0.75 y un F1-score del 0.76 (barra azul y naranja de la clase normal), si bien se puede considerar aceptable, pero sugiere una ligera tendencia a clasificar erróneamente algunas conexiones legítimas de ataques. Esto genera falsos positivos y podría generar carga operativa adicional en los sistemas de detección, claro que se puede mejorar escogiendo campos adicionales al contexto semántico en los tokens para que aprenda a distinguir tráfico "normal".

El modelo RoBERTa alcanzo un desempeño notable y a la altura de los modelos que fueron desarrollados especialmente para este tipo de datos, esto lo vemos en las métricas evaluadas:

Precisión (accuracy) de RoBERTa en UNSW-NB15: 78.52%

Reporte de métricas:

	precision	recall	f1-score	support
Normal	0.75	0.78	0.76	37000
Ataque	0.81	0.79	0.80	45332
accuracy			0.79	82332
macro avg	0.78	0.78	0.78	82332
weighted avg	0.79	0.79	0.79	82332

- **Accuracy (79%):** Al igual que BERT y GPT, RoBERTa logra clasificar correctamente casi 8 de 10 registros, lo que refleja un buen rendimiento general y competitivo.
- **F1 macro (78%):** el promedio del F1 entre las clases “normal y ataque” muestra que el modelo mantiene un equilibrio en su capacidad para ambas, con una ligera ventaja en la clase Ataque, lo cual es deseable desde la perspectiva de ciberseguridad.
- **F1 weighted (79%):** al considerar el desbalance natural entre clases, RoBERTa demuestra un rendimiento consistente y bien generalizado, lo que permite su uso práctico en entornos reales de red sin requerir ajustes extremos de balanceo o filtrado de alertas.

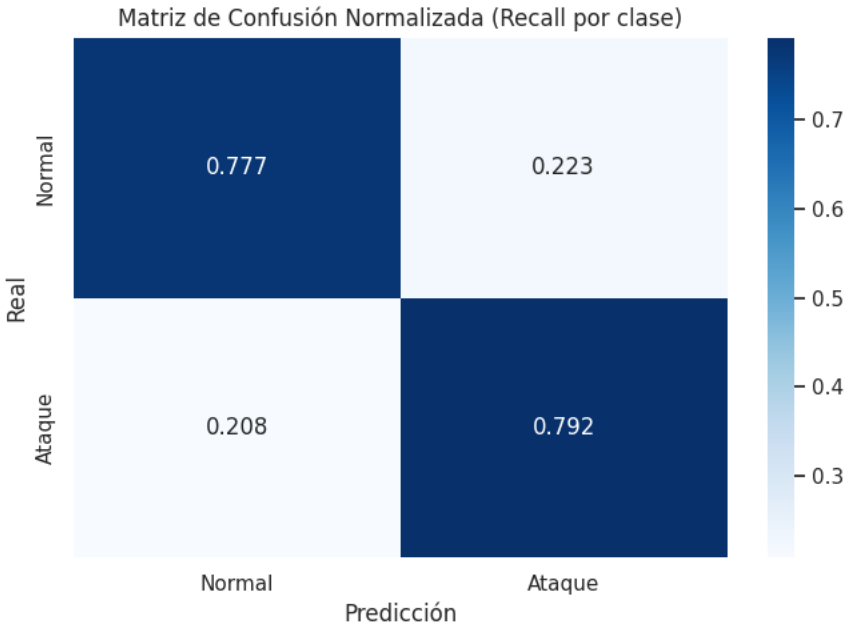


Figura 4- 37 Matriz de confusión RoBERTa datasetfull

En el tráfico “Normal”, el modelo logra una tasa de acierto del 77,7%, lo que representa una mejora respecto a GPT. Sin embargo, un 22,3% de las instancias normales fueron clasificadas erróneamente como ataques (falsos positivos), lo cual podría generar algunas alertas innecesarias en producción. No obstante, este margen de error se encuentra dentro de rangos

aceptables para sistemas IDS, especialmente si se prioriza la seguridad frente a la omisión de amenazas.

En cuanto al tráfico de "Ataque", RoBERTa identifica correctamente el 79,2% de las instancias, mientras que el 20,8% restante corresponde a falsos negativos. Aunque esta cifra es levemente inferior al recall de GPT (81%), se compensa con una mayor precisión en la detección de ataques (0.81), lo que reduce el número de falsos positivos y aumenta la confiabilidad de las alertas emitidas.

Cabe destacar que, a diferencia de los modelos anteriores, RoBERTa no ha sido tradicionalmente aplicado al análisis de tráfico de red, y mucho menos en entornos 5G, lo que hace de este experimento una aplicación innovadora y de alto impacto dentro del cruce entre PLN y ciberseguridad.

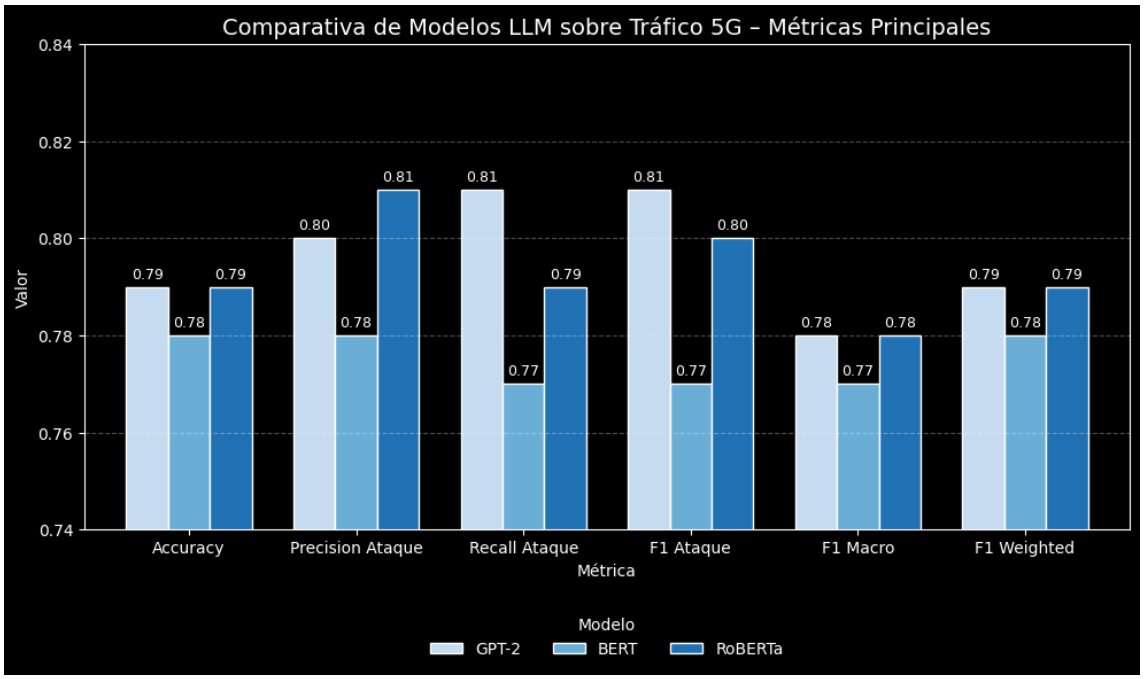


Figura 4- 38 Comparativa de modelos evaluados - métricas principales

El análisis del rendimiento de los tres modelos LLM que se evaluó (GPT-2, BERT y RoBERTa) en relación con el tráfico del dataset UNSW-NB15 revela diferencias significativas en términos de precisión, sensibilidad y capacidad de generalización. Como se puede ver en la Figura 4- 38, RoBERTa se destaca como el modelo más equilibrado y efectivo, logrando los mejores resultados en:

- Precisión en ataques (0.81): esto significa que la mayoría de las alertas generadas por el modelo realmente corresponden a eventos maliciosos, lo que ayuda a reducir los falsos positivos.
- F1-score en ataques (0.80): al combinar precisión y recall de manera efectiva, muestra un rendimiento óptimo en situaciones críticas, como la detección temprana de amenazas en redes 5G.

- Consistencia global: RoBERTa alcanza un F1 macro de 0.78 y un F1 weighted de 0.79, igualando a los otros modelos en todas las métricas clave, incluso teniendo en cuenta el desbalance de clases del dataset. Aunque GPT-2 también presenta una excelente sensibilidad (recall de 0.81) en la clase de ataque, su precisión es un poco más baja, lo que sugiere una mayor tendencia a falsos positivos. Por otro lado, BERT se queda atrás en todas las métricas, mostrando un rendimiento más conservador.

En resumen, los resultados indican que RoBERTa es el modelo más sólido para la detección de intrusiones en el tráfico de red 5G, destacándose no solo por sus métricas superiores, sino también por su estabilidad durante el entrenamiento y su capacidad para adaptarse a representaciones sintéticas del tráfico (como las secuencias proto-service-state utilizadas como entrada).

4.6.4 Interpretabilidad del modelo RoBERTa con XAI

Además de evaluar cuantitativamente el rendimiento del modelo RoBERTa en la detección de intrusiones en redes 5G, se incorporan técnicas de Inteligencia Artificial Explicable (XAI) con el objetivo de comprender cómo y por qué el modelo toma sus decisiones.

Se consideraron tres enfoques complementarios:

- SHAP (Shapley Additive Explanations) (DataScientest, 2023): Esta técnica proporciona explicaciones que permitan entender qué facotres impulsan o reducen la probabilidad de ataque.

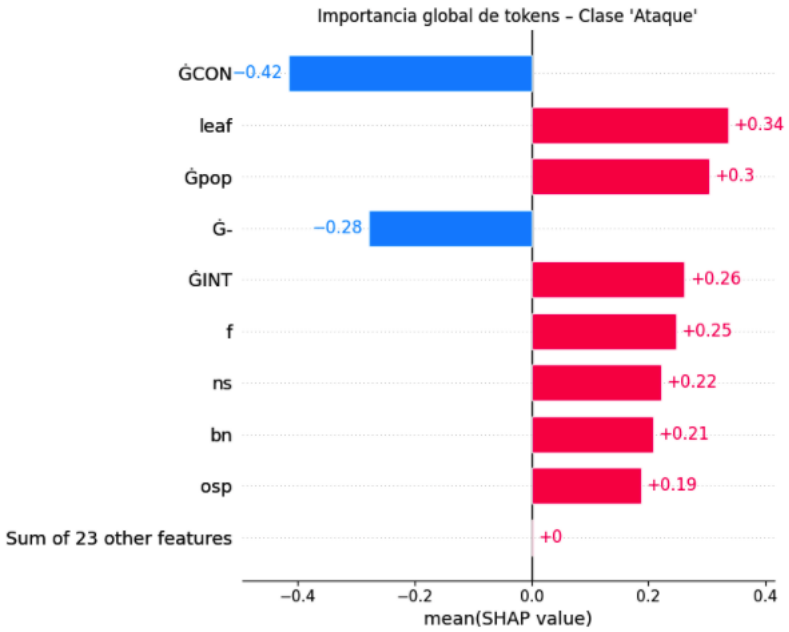


Figura 4- 39 XAI-SHAP / Contribución global de tokens a la clase "ataque"

Este análisis con SHAP confirma que RoBERTa aprendió patrones coherentes con el dominio. Servicios poco frecuentes en tráfico normal, como “leaf, pop y osp”, aparecen como los tokens más influyentes hacia la clase ataque.

Los LLM como RoBERTa, auqnue potentes, suelen comportarse como cajas negras, es decir, el modelo puede predecir que un flujo es ataque, pero no sabemos si fue por algún servicio como TCP,http, etc. Con las XAI e incluso con SHAP que se acaba de aplicar se acaba de descubrir que "leaf y osp" son los que aumentaron la probabilidad de ataque, mientras que "GCON" la redujo (ver Figura 4- 39).

- LIME (Local Interpretable Model-agnostic Explanations) (Ribeiro, Singh, & Guestrin, 2016): Es un algoritmo que puede explicar las predicciones de cualquier clasificador o regresor de manera transparente, aproximándolo localmente con un modelo interpretable.

Figura 4- 40 XAI-LIME / Contribución de tokens mediante LIME

Con LIME se analizó una secuencia distinta (tcp http SF) estos protocolos o servicios fueron clasificados como ataque (56% vs 44%). Es decir, el modelo clasifica este flujo como "ataque", aunque la diferencia no sea muy amplia.

A diferencia de SHAP, que ofrece una visión global de la importancia de los tokens, LIME permite comprender por qué el modelo toma una decisión específica. En laFigura 4- 40 se muestra la explicación para la secuencia *tcp http SF.* Los tokens TCP y HTTP se asociaron a la clase "normal" por ser protocolos comunes, mientras que el estados SF, aunque con menor peso, contribuyó a la predicción de "ataque". Este análisis local permite abrir la 'caja negra' del modelo en un caso concreto, mostrando cómo interactúan los tokens en la decisión final. LIME, de esta forma, complementa el análisis de SHAP al aportar interpretabilidad en instancias individuales.

Capítulo 5. DISCUSIÓN

Este estudio ha permitido hacer una comparación directa entre tres modelos de Lenguaje de Gran Escala (LLM) como GPT, BERT y RoBERTa aplicados a la detección de intrusiones en redes 5G, utilizando el dataset UNSW-NB15 como referencia. Durante el proceso, se identificaron varios factores técnicos y operacionales que impactaron directamente en el rendimiento de los modelos.

5.1.1 Análisis de resultados

Los resultados experimentales indican que RoBERTa logró el mejor rendimiento general, especialmente en las métricas relacionadas con el tráfico malicioso. Alcanzó una precisión de 0.81 y un F1-score de 0.80, lo que demuestra su capacidad para identificar patrones anómalos de manera robusta, incluso en situaciones con ruido y desbalanceo de clases. Este comportamiento coincide con estudios anteriores que han mostrado la alta eficacia de RoBERTa en tareas de clasificación en entornos complejos.

Por otro lado, GPT, logró un desempeño competitivo, con una precisión de 0.81 en la clase de ataque y un F1-score de 0.79, lo que confirma su adaptabilidad a tareas más allá del procesamiento de lenguaje natural. Sin embargo, presentó una tasa relativamente alta de falsos negativos (19%), lo que representa un riesgo crítico en escenarios de ciberseguridad.

BERT, aunque sólido en sus predicciones, mostró limitaciones al detectar ataques, con un recall de 0.70. Esto sugiere que tiene una menor sensibilidad para identificar eventos maliciosos, lo que podría resultar en una mayor cantidad de intrusiones no detectadas.

5.1.2 Limitaciones del estudio

Una de las principales limitaciones fue el uso de recursos computacionales moderados, lo que limitó la posibilidad de incluir modelos más recientes y exigentes, como LLaMA, GPT-3 o variantes de arquitecturas basadas en decodificadores de gran tamaño.

5.1.3 Reflexión sobre la metodología

La tokenización de los datos de red en formato texto y su adaptación a modelos originalmente diseñados para lenguaje natural, funcionó sin comprometer la validez de los resultados. Además, se logró mantener la reproducibilidad al emplear estructuras homogéneas de entrenamiento para todos los modelos.

5.1.4 Impacto de los resultados

Los resultados obtenidos tienen un impacto significativo en el ámbito de la detección de intrusiones en redes 5G. La evidencia sugiere que modelos LLM pueden ser empleados con éxito en ciberseguridad, incluso sino fueron diseñados originalmente para este propósito.

Capítulo 6. CONCLUSIONES

6.1 Conclusiones del trabajo

Este estudio logró alcanzar con éxito el objetivo de evaluar y comparar modelos de lenguaje de gran escala (LLM) aplicados a la detección de intrusiones en redes 5G, utilizando como referencia el dataset UNSW-NB15 y aplicando una metodología de análisis basada en métricas de clasificación estándar.

A través del estado del arte se cumplió con la selección fundamentada de tres modelos representativos (GPT, BERT Y ROBERTA), cada uno con características arquitectónicas distintas. Se preparó adecuadamente el conjunto de datos, estructurando los registros en secuencias compatibles con modelos de lenguaje, sin perder la semántica del tráfico de red. Ademas, se llevó a cabo una evaluación detallada, obteniendo resultados cuantitativos y visuales que permitieron establecer comparaciones objetivas entre modelos.

La evaluación y resultados empleados demostraron que RoBERTa es el modelo con mejor rendimiento global, con un F1score de 80% en la clase ataques, precisión del 81% y una notable consistencia en todas las métricas a pesar de no haber estudios relacionados de este modelo con tráfico de red 5G, lo que convierte esta implementación en una propuesta pionera dentro del cruce entre PLN y ciberseguridad.

Finalmente, se destaca el valor añadido de técnicas de Inteligencia Artificial Explicable (XAI), cuya incorporación permitió comprender las decisiones del modelo RoBERTa y validar su comportamiento interno, aumentando así la transparencia del sistema y su aplicabilidad práctica en entornos reales de red.

6.2 Conclusiones personales

El desarrollo de esta investigación fue una experiencia innovadora, tecnológica y muy educativa que permitió aplicar en profundidad las técnicas aprendidas durante el máster, dando ideas de aplicar estas técnicas tanto en a nivel profesional como personal.

La experiencia de trabajar con estos modelos y enfrentar los desafíos del procesamiento de estos lenguajes amplió la comprensión, habilidades y ganas de aprender más de la aplicabilidad de estos modelos.

Con estas técnicas y lecciones aprendidas proporcionan una base sólida de conocimientos para que sean empleados y puesto a prueba en los proyectos más ambiciosos tanto en nuevos campos laborales o de autoaprendizaje en el análisis de datos.

Capítulo 7. FUTURAS LÍNEAS DE TRABAJO

7.1 Trabajos futuros

Este estudio se centró en arquitecturas accesibles como BERT, GPT y RoBERTa. Futuras investigaciones pueden incluir modelos de última generación como LLaMA, Mistral, Gemma, etc. Y con infraestructuras computacionales que compensen y puedan aprovechar el máximo el potencial de estos modelos, un factor clave son los tiempos de respuestas, ya que en un entorno de tráfico de red en tiempo real se necesita respuestas inmediatas. Con recursos más potentes permitiría reducir los tiempos de entrenamiento, trabajar con datasets más grandes y explorar modelos de última generación sin limitaciones operativas.

En este estudio se evaluó una clasificación binaria (normal vs ataque). Futuras investigaciones pueden abordar un enfoque multi-clase que permita identificar tipos específicos de ataques (DDoS, Exploits, Fuzzers, etc.), lo cual sería clave para una detección más específica en sistemas de detección.

IDS basado en LLM como servicio, se propone el desarrollo de un software interactivo que integre modelos de lenguaje entrenados para detección de intrusiones. Esta herramienta permitiría a los usuarios:

- Cargar datasets personalizados
- Seleccionar características relevantes (proto, state, srcio, etc.).
- Elegir el modelo a aplicar (RoBERTa, GPT, BERT, Llama, etc)
- Visualizar en tiempo real las métricas y mostrar dashboard del o los modelos para una comparación de resultados.
- Aplicar explicabilidad para entender decisiones del modelo.

Capítulo 8. REFERENCIAS

Adjewa, F., Esseghir, M., & Merghem-Boulahia, L. (2024, septiembre 28). *Efficient Federated Intrusion Detection in 5G ecosystem using optimized BERT-based model*. arXiv preprint arXiv:2409.19390

Adjewa, F. (2024). *Federated-security* [Repositorio GitHub]. Recuperado de https://github.com/freddyAdjh/Federated-security

AENOR. (2010). *AEN/CTN 157 - PROYECTOS*. Recuperado el 25 de abril de 2013 de http://www.aenor.es/aenor/normas/ctn/fichactn.asp?codigonorm=AEN/CTN%20157

AWS Amazon. (2024). *¿Qué es un modelo de lenguaje grande (LLM)?* Recuperado de https://aws.amazon.com/es/what-is/large-language-model/

Canales Luna, J. (2024, septiembre 11). *¿Qué es BERT? Una introducción a los modelos BERT*. DataCamp. Recuperado de https://www.datacamp.com/es/blog/what-is-bert-an-intro-to-bert-models

Cisco Systems. (s.f.). *¿Qué es 5G?* Recuperado de https://www.cisco.com/c/en/us/solutions/what-is-5g.html

Cloudflare. (s.f.). *¿Qué es un protocolo de red?* Recuperado de https://www.cloudflare.com/es-es/learning/network-layer/what-is-a-protocol/

DeepAI. (s.f.). *Gradient clipping - Glosario de aprendizaje automático*. Recuperado de https://deepai.org/machine-learning-glossary-and-terms/gradient-clipping

Evidently AI. (2025, enero 9). *Accuracy vs. precision vs. recall in machine learning: What's the difference?* Recuperado de https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall

FAIRSEQ. (s.f.). *RoBERTa - PyTorch Hub*. Recuperado de https://pytorch.org/hub/pytorch_fairseq_roberta

Fortinet Inc. (s.f.). *¿Qué es 5G?* Recuperado de https://www.fortinet.com/lat/resources/cyberglossary/what-is-5g

Google. (2025). *Google Colaboratory*. Recuperado de https://colab.google

Google Cloud. (s.f.). *Modelos de lenguaje grandes (LLM)*. Recuperado de https://cloud.google.com/ai/llms?hl=es-419

Gugger, S., & Howard, J. (2018, julio 2). *AdamW optimizer and weight decay*. Blog de Fast.ai. Recuperado de https://www.fast.ai/posts/2018-07-02-adam-weight-decay.html

Huang, J. (2024, diciembre 9). *Take Package as Language: Anomaly Detection Using Transformer*. arXiv [2412.0447].

Hugging Face. (s.f.). *Transformers Documentation*. Recuperado de https://huggingface.co/docs/transformers/index

IBM. (2024, marzo 12). *¿Qué es DNS?* Recuperado de https://www.ibm.com/es-es/think/topics/dns-protocol

Intel Corporation. (s.f.). *CPU vs. GPU*. Recuperado de https://www.intel.la/content/www/xl/es/products/docs/processors/cpu-vs-gpu.html

Jóvenes en la Ciencia. (2022). *Artículo académico sobre IA y PLN*. Recuperado de https://www.jovenesenlaciencia.ugto.mx/index.php/jovenesenlaciencia/article/view/4598/4070

Kan, K. B., Mun, H., Cao, G., & Lee, Y. (2024). *Mobile-LLaMA: Instruction fine-tuning open-source LLM for network analysis in 5G networks*. IEEE Network, 38(5), 76-83.

Leung, K. (2023, enero 4). *Micro, macro & weighted averages of F1 score, clearly explained*. KDnuggets. Recuperado de https://www.kdnuggets.com/2023/01/micro-macro-weighted-averages-f1-score-clearly-explained.html

Lenovo. (2024, abril 3). *Epoch: ¿Qué es y para qué sirve?* Recuperado de https://www.lenovo.com/co/es/glosario/epoch

Lenovo. (s.f.). *¿Qué es FTTP?* Recuperado de https://www.lenovo.com/es/es/glossary/fttp/

Meta. (2023, febrero 24). *LLAMA: Large Language Model by Meta AI*. Recuperado de https://ai.meta.com/blog/large-language-model-llama-meta-ai/

Microsoft. (2024, mayo 17). *Introducción a HTTP*. Microsoft Learn. Recuperado de https://learn.microsoft.com/es-es/xandr/industry-reference/intro-to-http

Miró Julià, J. (2010). *Recursos para aprender a escribir*. Recuperado de http://bioinfo.uib.es/~joemiro/RecEscr/manual.pdf

NVIDIA Corporation. (s.f.). *¿Qué es PyTorch?* Recuperado de https://www.nvidia.com/en-us/glossary/pytorch/

Omar Elharrouss, Y. M. (2025, abril 5). *Loss functions in deep learning: A comprehensive review*. *Revista Latinoamericana de Tecnología*.

OpenAI. (2019, noviembre 5). *GPT-2 Release*. Recuperado de https://openai.com/index/gpt-2-1-5b-release/

Python Software Foundation. (2024). *Documentación oficial de Python 3*. Recuperado de https://docs.python.org/3/

Richman, A. (2024, junio 10). *Dynamic data masking: Why you need it*. K2View. Recuperado de https://www.k2view.com/blog/dynamic-data-masking/

Stryker, C., & Holdsworth, J. (2024, agosto 11). *¿Qué es el PLN (procesamiento del lenguaje natural)?* IBM Think. Recuperado de https://www.ibm.com/es-es/think/topics/natural-language-processing

UNE 157001. (2002). *Criterios generales para la elaboración de proyectos*. Recuperado de http://www.coiib.es/coiib/documentos/DocumentosContenidos/Guía%20de%20elaboración%20de%20proyectos/2-Electricidad/5_PNE_157701_Criterios.pdf

University of New South Wales. (2015). *UNSW-NB15 Dataset*. Recuperado de https://research.unsw.edu.au/projects/unsw-nb15-dataset

Walther, J. (2023, septiembre 4). *¿Qué es Pipeline? Una guía completa para principiantes*. Tutoriales Dongee. Recuperado de https://www.dongee.com/tutoriales/que-es-pipeline-una-guia-completa-para-principiantes/

Capítulo 9. ANEXOS

Con respecto a nuestro entorno, se utilizó la plataforma de Google Colab y una suscripción a Colab-Pro ¿por qué?, dado a los limitados recursos de la suscripción gratis este demoraba alrededor de 45min a 2 horas un entrenamiento dependiendo del modelo. Por esa razón se optó por la suscripción Pro (pago de 10.00 USD mensuales), para tener acceso a aceleradores de hardware como un GPU A100, que optimizó tiempos de entrenamiento por época considerando el dataset con el que se trabajó que contiene miles de datos.

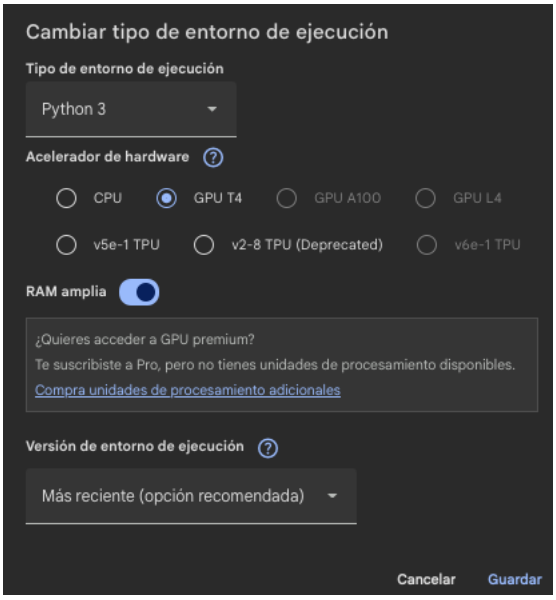


Figura 4- 41 Entorno de ejecución permitidos por Google Colab Pro

9.1 ANEXO A: Carga de datasets en el entorno

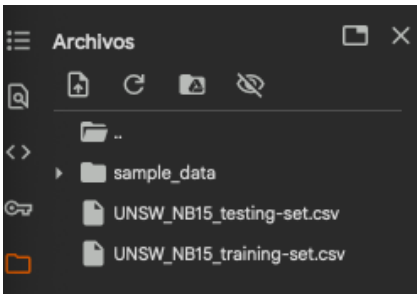


Figura 4- 42 Datasets UNSW_NB15 cargado a Google Colab

Es necesario cargar los datasets en el entorno de trabajo. En este caso, se utilizó los archivos tanto de entrenamiento como del conjunto de prueba.

9.2 ANEXO B: Bloques de código comunes entre LLM aplicados al estudio

- Instalación de librerías requeridas

```
# Librerías necesarias
!pip install transformers datasets torch pandas matplotlib seaborn
```

Figura 4- 43 Código de instalación de librerías requeridas

Este fragmento o bloque de código se utilizó para todas las experimentaciones de los tres modelos LLM utilizados para el estudio.

- Carga de datasets

```
import pandas as pd

train_nb15= pd.read_csv('/content/UNSW_NB15_training-set.csv')
test_nb15 = pd.read_csv('/content/UNSW_NB15_testing-set.csv')

# Verificar Registros
train_nb15['label'].value_counts()
train_nb15['attack_cat'].value_counts()
```

Figura 4- 44 Bloque de código de la carga de los datasets

- Configuración de semillas para reproducibilidad

```
# ==== Semillas para reproducibilidad ====
import os, random, numpy as np, torch
from transformers import set_seed

SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(SEED)

# para que cuDNN no cambie heurísticas entre ejecuciones
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# utilidad de transformers (afecta generadores internos)
set_seed(SEED)
# ====================================

print("✅ Reproducibilidad configurada (SEED = 42)")
print(f"📊 Registros de entrenamiento: {train_nb15.shape[0]}")
print(f"📊 Registros de prueba: {test_nb15.shape[0]}")
train_nb15.head()

✅ Reproducibilidad configurada (SEED = 42)
📊 Registros de entrenamiento: 175341
📊 Registros de prueba: 82332
```

Figura 4- 45 Bloque de código "semillas" para asegurar comparabilidad

Este bloque de código garantiza la reproducibilidad de los experimentos.

- Se fijan semillas para random, numpy y torch, evitando que los resultados varíen entre ejecuciones.
- Se configuran parámetros de cuDNN (deterministic=True, benchmark=False) para asegurar consistencia en la ejecución en GPU.
- Se emplea la función set_seed de Hugging Face Transformers, que afecta a los generadores internos de la librería.

- Preprocesamiento de datos para la construcción de secuencias de texto

```
# === dataset completo (dataset ya preprocesado y limpio) ===
train_df_full = train_nb15.copy()
test_df_full  = test_nb15.copy()

# Construcción de la secuencia textual (proto + service + state)
train_df_full['text'] = train_df_full['proto'] + " " + train_df_full['service'] + " " + train_df_full['state']
test_df_full['text']  = test_df_full['proto']  + " " + test_df_full['service']  + " " + test_df_full['state']

# Visualizar ejemplos generados
print("📝 Ejemplos de secuencias de texto para RoBERTa:")
display(train_df_full[['proto', 'service', 'state', 'text']].sample(3))
```

```
📝 Ejemplos de secuencias de texto para RoBERTa:
```

	proto	service	state	text
15482	tcp	-	FIN	tcp - FIN
133349	udp	dns	INT	udp dns INT
80485	tcp	-	FIN	tcp - FIN

Figura 4- 46 Bloque de código para preprocesamiento de la secuencia de texto

Este bloque corresponde al preprocesamiento previo al entrenamiento de los modelos LLM.

- Se crean copias del conjunto de entrenamiento y de prueba.
- Se construye una nueva columna llamada text, que concatena los campos proto (protocolo), service (servicio) y state (estado de conexión).
- Esta concatenación transforma cada registro en una secuencia textual interpretable por el tokenizerador.

9.3 ANEXO C: Bloques de código tokenización y carga de modelos

- Tokenización y carga de LLM BERT

```
from transformers import BertTokenizer, BertForSequenceClassification

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(list(train_df_full['text']), truncation=True, padding='max_length', max_length=16)
test_encodings = tokenizer(list(test_df_full['text']), truncation=True, padding='max_length', max_length=16)
```

Figura 4- 47 Bloque de código tokenizador y carga de modelo BERT

- Tokenización y carga de LLM RoBERTa

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

MODEL_NAME = "roberta-large"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=2)

MAX_LEN= 16
train_encodings_roberta = tokenizer(
    list(train_df_full['text']),
    truncation=True,
    padding='max_length',
    max_length=MAX_LEN,
    return_tensors="pt"
)

test_encodings_roberta = tokenizer(
    list(test_df_full['text']),
    truncation=True,
    padding='max_length',
    max_length=MAX_LEN,
    return_tensors="pt"
)

print("✅ Tokenización finalizada. Ready to fine-tune RoBERTa.")
```

Figura 4- 48 Bloque de código tokenizador y carga de modelo RoBERTa

- Tokenización y carga de LLM GPT

```python
# === TOKENIZADOR GPT-2
from transformers import GPT2TokenizerFast, GPT2ForSequenceClassification

MAX_LEN = 16
gpt_tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")
gpt_tokenizer.pad_token = gpt_tokenizer.eos_token
gpt_tokenizer.padding_side = "right"

train_encodings_gpt = gpt_tokenizer(
    train_df_full["text"].tolist(),
    truncation=True, padding="max_length", max_length=MAX_LEN
)
test_encodings_gpt = gpt_tokenizer(
    test_df_full["text"].tolist(),
    truncation=True, padding="max_length", max_length=MAX_LEN
)
# carga del modelo
model = GPT2ForSequenceClassification.from_pretrained("gpt2", num_labels=2)
model.config.pad_token_id = gpt_tokenizer.pad_token_id
```

En este bloque se definen las diferencias entre modelos:

- RoBERTa: utiliza AutoTokenizer y AutoModelForSequenceClassification, con secuencias de longitud reducida (MAX_LEN=16).
- BERT: emplea BertTokenizer y BertForSequenceClassification, con longitud de secuencia mayor (max_length=16).
- GPT-2: requiere configuraciones adicionales, ya que por defecto no tiene un *padding token*. Se añadió el token de fin de secuencia (eos_token) como pad_token para permitir el entrenamiento.

Este anexo muestra claramente las particularidades de cada modelo antes de proceder al entrenamiento y evaluación.

9.4 ANEXO D: Creación de DataLoaders (estructura general)

```python
from torch.utils.data import Dataset, DataLoader
import torch

class TextClsDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __len__(self):
        return len(self.labels)
    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx], dtype=torch.long) for k, v in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx], dtype=torch.long)
        return item

train_dataset_full = TextClsDataset(train_encodings, list(train_df_full['label']))
test_dataset_full  = TextClsDataset(test_encodings,  list(test_df_full['label']))

g = torch.Generator(); g.manual_seed(SEED)  # opcional, para shuffle reproducible
train_loader = DataLoader(train_dataset_full, batch_size=16, shuffle=True, generator=g, num_workers=2, pin_memory=True)
test_loader  = DataLoader(test_dataset_full,  batch_size=16, shuffle=False, num_workers=2, pin_memory=True)
```

Figura 4- 49 Bloque de código para creación de DataLoaders

Este bloque se reutilizó para los tres modelos evaluados, especificando parámetros importantes como batch_size=16, definido para equilibrar memoria y estabilidad del entrenamiento.

9.5 ANEXO E: Configuración y bucle de entrenamiento

```python
from torch import nn
from torch.optim import AdamW
from transformers import get_scheduler
from tqdm import tqdm

# === Configuración de entrenamiento ===
EPOCHS = 5
LEARNING_RATE = 2e-5
WEIGHT_DECAY = 0.01
MAX_GRAD_NORM = 1.0

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# === Optimizer y Scheduler con warmup ===
optimizer = AdamW(model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY)

total_steps = len(train_loader) * EPOCHS
scheduler = get_scheduler(
    name="linear",
    optimizer=optimizer,
    num_warmup_steps=int(0.1 * total_steps),  # 10% warmup
    num_training_steps=total_steps
)

# === Función de pérdida
criterion = nn.CrossEntropyLoss()

# === Entrenamiento
for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
    loop = tqdm(train_loader, desc=f"🚀 Epoch {epoch+1}/{EPOCHS}", leave=False)

    for batch in loop:
        batch = {k: v.to(device) for k, v in batch.items()}

        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()

        # Clipping de gradientes
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=MAX_GRAD_NORM)

        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()

        total_loss += loss.item()
        loop.set_postfix(loss=loss.item())

    avg_loss = total_loss / len(train_loader)
    print(f"✅ Época {epoch+1} completada — Loss promedio: {avg_loss:.4f}")
```

Figura 4- 50 Bloque de código para entrenamiento

Hiperparámetros principales

- EPOCHS = 5: número de veces que el modelo recorre todo el dataset.
- LEARNING_RATE = 2e-5: tasa de aprendizaje baja, recomendada para fine-tuning de LLMs.
- WEIGHT_DECAY = 0.01: regularización L2 para evitar sobreajuste.
- MAX_GRAD_NORM = 1.0: evita explosión de gradientes al limitar su magnitud.

9.6 ANEXO F: Guardado del modelo y tokenizador entrenados

```
model.save_pretrained("./roberta_ids_model")
tokenizer.save_pretrained("./roberta_ids_model")
```

```
('./roberta_ids_model/tokenizer_config.json',
 './roberta_ids_model/special_tokens_map.json',
 './roberta_ids_model/vocab.json',
 './roberta_ids_model/merges.txt',
 './roberta_ids_model/added_tokens.json',
 './roberta_ids_model/tokenizer.json')
```

Figura 4- 51 Bloque de código para guardar entrenamiento, ejemplo LLM RoBERTa

Este procedimiento se aplicó a cada modelo entrenado (RoBERTa, BERT y GPT-2), guardando sus versiones fine-tuned con nombres distintos de carpeta (./roberta_ids_model, /bert_ids_model, /gpt2_ids_model).

9.7 ANEXO F: Evaluación y métricas de rendimiento

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np, seaborn as sns, matplotlib.pyplot as plt
import torch

model.eval()

# === Accuracy simple ===
correct, total = 0, 0
with torch.no_grad():
    for batch in test_loader:
        out = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device)
        )
        preds = torch.argmax(out.logits, dim=1)
        correct += (preds.cpu() == batch['labels']).sum().item()
        total   += batch['labels'].size(0)

acc = correct / total
print(f"📊 Precisión (accuracy) en UNSW-NB15: {acc:.2%}")

# === Reporte detallado + matriz de confusión ===
all_preds, all_labels = [], []

with torch.no_grad():
    for batch in test_loader:
        out = model(
            batch['input_ids'].to(device),
            attention_mask=batch['attention_mask'].to(device)
        )
        probs = torch.softmax(out.logits, dim=-1)
        preds = (probs[:,1] >= 0.5).long()  # Umbral ajustado a 0.5
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(batch['labels'].cpu().numpy())

# === Classification report ===
print("\n📈 Reporte de métricas:")
print(classification_report(
    all_labels,
    all_preds,
    target_names=['Normal', 'Ataque'],
    digits=2,
    zero_division=0
))

# === Matriz de confusión ===
cm = confusion_matrix(all_labels, all_preds, labels=[0,1])

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal','Ataque'], yticklabels=['Normal','Ataque'])
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión')
plt.show()
```

Figura 4- 52 Bloque de código para evaluación de los modelos entrenados

9.8 ANEXO F: QR pipeline de los modelos evaluados

Figura 4- 53 QR para acceso a los pipelines de los modelos evaluados

El código QR permite acceder directamente al repositorio donde se encuentra el pipeline implementado, incluyendo las fases de preprocesamiento, tokenización, entrenamiento, evaluación y explicabilidad con SHAP y LIME. Este recurso busca facilitar la reproducibilidad del trabajo y servir como guía práctica para futuras investigaciones.

[PÁGINA INTENCIONADAMENTE EN BLANCO]