Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Electricidad y Computación

Prototipo Sistema de Detección de Intrusiones Militar: Diseño y Validación con Modelo Híbrido

INGE-2865

Proyecto Integrador

Previo la obtención del Título de:

Ingeniero en Telecomunicaciones

Presentado por:
Josué Javier Tigrero Panchana
Jacinto Javier Ortiz Fernández

Guayaquil - Ecuador
Año: 2025

Dedicatoria

Este trabajo está dedicado con todo mi cariño:

A mis padres Galo Tigrero y Karina Panchana, por su apoyo constante y por la invaluable oportunidad de estudiar en la ESPOL. Su confianza en mis capacidades fue la motivación que necesité en cada paso para llegar hasta aquí.

A mi abuela Isabel, cuyo amor y ejemplo me han ayudado a convertirme en la persona que soy ahora.

A Erick, Kevin, Andrés, Steven y Boris, por el buen humor que disipaba el estrés y por hacer de la universidad una experiencia memorable. Su amistad fue clave para mantener el equilibrio.

Javier Josué Tigrero Panchana

Dedicatoria

Dedico primeramente este proyecto a Dios, quien me ha dado la fortaleza, sabiduría y bendiciones necesarias para culminar esta etapa de mi formación académica.

A mis queridos padres, Rosa y José, quienes con su amor incondicional, me brindaron todo su apoyo para alcanzar esta meta.

A mis hermanos José Andrés y Carlos, por su compañía y por celebrar conmigo cada pequeño logro durante estos años de estudio.

A mi querido abuelito Hilario, que aunque ya no está físicamente presente, sigue siendo una luz y guía en mi vida.

Y a mis compañeros de carrera, por compartir conmigo este camino de aprendizaje.

Jacinto Javier Ortiz Fernández

Agradecimientos

Mi más sincero agradecimiento a mi tutor, Germán, por su invaluable guía y paciencia, las cuales fueron fundamentales para la consecución de este proyecto.

Agradezco también a mi profesora, Patricia, por sus acertados consejos durante el desarrollo de la materia integradora.

Extiendo mi gratitud al Ing. Marcos y a la DINDES, por la oportunidad y la confianza depositadas en este trabajo.

Un reconocimiento especial a mi compañero de proyecto Jacinto, por su dedicación y trabajo en equipo.

Finalmente, un profundo agradecimiento a mi familia por su incondicional apoyo y motivación, los cuales me permitieron alcanzar esta meta.

Josue Javier Tigrero Panchana

Agradecimientos

Mi más sincero agradecimiento a mis docentes, por su dedicación y contribución a mi formación académica.

A mi tutor Germán, por su guía experta y orientación durante este proyecto.

A Leonardo, quien fue mi compañero de trabajo, por enseñarme todo lo necesario para crecer profesionalmente y compartir generosamente su experiencia.

A mi compañero Josue, por su apoyo constante para culminar este proyecto.

A mi querida familia, por la confianza puesta en mí y por creer siempre en mis capacidades.

Mi reconocimiento a todos quienes hicieron posible este logro.

Jacinto Javier Ortiz Fernández

Declaración Expresa

Nosotros Jacinto Javier Ortiz Fernández, Josué Javier Tigrero Panchana acordamos y reconocemos que:

La titularidad de los derechos patrimoniales de autor (derechos de autor) del proyecto de graduación corresponderá al autor o autores, sin perjuicio de lo cual la ESPOL recibe en este acto una licencia gratuita de plazo indefinido para el uso no comercial y comercial de la obra con facultad de sublicenciar, incluyendo la autorización para su divulgación, así como para la creación y uso de obras derivadas. En el caso de usos comerciales se respetará el porcentaje de participación en beneficios que corresponda a favor del autor o autores.

La titularidad total y exclusiva sobre los derechos patrimoniales de patente de invención, modelo de utilidad, diseño industrial, secreto industrial, software o información no divulgada que corresponda o pueda corresponder respecto de cualquier investigación, desarrollo tecnológico o invención realizada por nosotros durante el desarrollo del proyecto de graduación, pertenecerán de forma total, exclusiva e indivisible a la ESPOL, sin perjuicio del porcentaje que nos corresponda de los beneficios económicos que la ESPOL reciba por la explotación de nuestra innovación, de ser el caso.

En los casos donde la Oficina de Transferencia de Resultados de Investigación (OTRI) de la ESPOL comunique los autores que existe una innovación potencialmente patentable sobre los resultados del proyecto de graduación, no se realizará publicación o divulgación alguna, sin la autorización expresa y previa de la ESPOL.

Guayaquil, 31 de Mayo del 2025.

Firmado electrónicamente por:
JACINTO JAVIER ORTIZ FERNANDEZ
Validar únicamente con FirmaEC

Jacinto Javier Ortiz Fernández

Firmado electrónicamente por:
JOSUE JAVIER TIGRERO PANCHANA
Validar únicamente con FirmaEC

Josué Javier Tigrero Panchana

Evaluadores

Patricia Ximena Chávez Burbano,

Ph.D.

Profesor de Materia

Germán Ricardo Vargas López,

Ph.D.

Tutor de proyecto

Resumen

El presente proyecto desarrolla y valida un modelo de inteligencia artificial para un Sistema de Detección de Intrusiones (IDS), con el objetivo de identificar amenazas cibernéticas en tiempo real dentro de redes locales militares. La justificación se fundamenta en la necesidad crítica de superar las limitaciones de los sistemas tradicionales ante ataques sofisticados, proponiendo un enfoque autónomo y adaptable para infraestructuras de alta seguridad.

Para el desarrollo, se realizó una evaluación experimental comparativa entre cuatro arquitecturas de Deep Learning, utilizando el dataset CICIDS2017 y herramientas como TensorFlow y Keras. Como resultado de esta competición, se seleccionó el modelo híbrido 1D-CNN + LSTM por su rendimiento superior, el cual alcanzó un F1-Score de 0.9894 y la menor cantidad de falsos negativos, el error más crítico en este contexto. Posteriormente, un prototipo funcional integró este modelo y validó su eficacia al detectar exitosamente un ataque de escaneo de puertos en vivo.

En conclusión, el trabajo demuestra la viabilidad de aplicar arquitecturas híbridas de IA para crear sistemas de seguridad precisos, estableciendo una base sólida para la modernización de la ciberdefensa.

Palabras Clave: Ciberseguridad, Inteligencia Artificial, Redes Neuronales, Detección de Anomalías.

Abstract

This project develops and validates an artificial intelligence model for an Intrusion Detection System (IDS) with the goal of identifying cyber threats in real time within local military networks. The justification is based on the critical need to overcome the limitations of traditional systems in the face of sophisticated attacks, proposing an autonomous and adaptable approach for high-security infrastructures.

For the development, a comparative experimental evaluation was conducted between four Deep Learning architectures, using the CICIDS2017 dataset and tools such as TensorFlow and Keras. As a result of this competition, the hybrid 1D-CNN + LSTM model was selected for its superior performance, achieving an F1 score of 0.9894 and the lowest number of false negatives, the most critical error in this context. Subsequently, a functional prototype integrated this model and validated its effectiveness by successfully detecting a live port scanning attack.

In conclusion, the work demonstrates the feasibility of applying hybrid AI architectures to create precise security systems, establishing a solid foundation for modernizing cyber defense.

Keywords: *Cybersecurity, Artificial Intelligence, Neural Networks, Anomaly Detection.*

Índice general

Abreviaturas

ANN Artificial Neural Network (Red Neuronal Artificial)

APT Advanced Persistent Threat (Amenaza Persistente Avanzada)

AUC Area Under the Curve (Área Bajo la Curva)

CIC Canadian Institute for Cybersecurity (Instituto Canadiense de Ciberseguridad)

CNN Convolutional Neural Network (Red Neuronal Convolucional)

DDoS Distributed Denial-of-Service (Ataque de Denegación de Servicio Distribuido)

FN False Negative (Falso Negativo)

FP False Positive (Falso Positivo)

GRU Gated Recurrent Unit (Unidad Recurrente Cerrada)

GUI Graphical User Interface (Interfaz Gráfica de Usuario)

IA Inteligencia Artificial

IDS Intrusion Detection System (Sistema de Detección de Intrusiones)

IP Internet Protocol (Protocolo de Internet)

LSTM Long Short-Term Memory (Memoria a Corto-Largo Plazo)

ML Machine Learning (Aprendizaje Automático)

NIDS Network-based Intrusion Detection System (Sistema de Detección de Intrusiones Basado en Red)

ROC Receiver Operating Characteristic (Característica Operativa del Receptor)

TCP Transmission Control Protocol (Protocolo de Control de Transmisión)

UDP User Datagram Protocol (Protocolo de Datagramas de Usuario)

Índice de figuras

Índice de tablas

Capítulo 1

1 Introducción

En la era digital contemporánea, la seguridad cibernética constituye uno de los pilares fundamentales para la protección de infraestructuras críticas a nivel global. Las organizaciones gubernamentales, empresas privadas e instituciones militares enfrentan un panorama de amenazas que evoluciona constantemente en sofisticación y frecuencia, donde los ciberataques representan riesgos estratégicos de magnitud nacional. El informe Global Cybersecurity Outlook 2025 del Foro Económico Mundial (World Economic Forum, 2025) revela que el 72% de las organizaciones reporta un incremento en los riesgos cibernéticos durante el último año, evidenciando un panorama de amenazas cada vez más complejo. Este aumento refleja la creciente sofisticación de las actividades cibercriminales y la expansión de las superficies de ataque en un entorno digital interconectado.

El sector militar representa uno de los ámbitos más vulnerables y estratégicos en términos de seguridad cibernética, debido a que administra una amplia gama de recursos de alto valor que constituyen objetivos prioritarios para diversos actores maliciosos, tanto estatales como no estatales. Entre estos recursos se incluyen información clasificada de naturaleza crítica, sistemas de comunicaciones militares, infraestructura estratégica y tecnologías especializadas. Según el Plan Estratégico Institucional de Defensa 2021-2025, las Fuerzas Armadas del Ecuador cuentan con estructuras tecnológicas para los enlaces de telecomunicaciones e informática, las cuales requieren constante modernización, estandarización y fortalecimiento para disponer de herramientas y soluciones de Tecnologías de Información y Comunicación (TIC) específicas que permitan gestionar sus procesos y operaciones de forma integral. Estas estructuras incluyen un sistema de mando y control que integra varios subsistemas, proporcionando al mando la información necesaria de sus unidades y medios de detección para la toma de decisiones acertadas (Ministerio de Defensa Nacional, 2023). La naturaleza crítica de estas operaciones determina que cualquier compromiso de

seguridad puede generar consecuencias devastadoras no sólo para la institución militar específica, sino para la seguridad nacional integral.

Los sistemas de redes locales en entornos militares presentan características distintivas que los diferencian de las redes corporativas convencionales. Estas redes operan bajo protocolos de seguridad extremadamente estrictos, mantienen la confidencialidad absoluta de la información, garantizan la integridad de los datos y aseguran la disponibilidad continua de servicios críticos (National Institute of Standards and Technology, 2018). Adicionalmente, frecuentemente funcionan en condiciones adversas, con recursos limitados y en ubicaciones remotas donde el acceso a soporte técnico especializado puede ser restringido o inexistente.

Las amenazas cibernéticas han experimentado una evolución significativa, transformándose desde simples virus diseñados por entusiastas, hasta sofisticadas herramientas de espionaje y sabotaje utilizadas por actores estatales y criminales organizados para ejecutar campañas de amenazas persistentes avanzadas (ESET, 2024). Los vectores de ataque contemporáneos incluyen técnicas avanzadas de ingeniería social, explotación de vulnerabilidades de día cero, ataques de denegación de servicio distribuido (DDoS), infiltración lateral, y el uso emergente de inteligencia artificial para automatizar y optimizar los procesos de ataque (CrowdStrike, 2025).

Los actores de amenazas han diversificado significativamente sus motivaciones y capacidades, pasando de los ataques cibernéticos perpetrados históricamente por individuos con motivaciones personales o financieras, a un panorama actual caracterizado por un incremento exponencial en ataques patrocinados por estados, grupos terroristas organizados y organizaciones criminales que poseen recursos y capacidades técnicas avanzadas. Estos actores invierten recursos considerables en el desarrollo de herramientas de ataque sofisticadas y en la realización de reconocimiento extensivo de sus objetivos estratégicos (Microsoft, 2024).

En el contexto militar específico, las amenazas incluyen espionaje cibernético dirigido a obtener información de inteligencia, sabotaje de sistemas críticos, manipulación de comunicaciones, compromiso de sistemas de armas, y ataques diseñados para interrumpir operaciones militares. La proliferación de dispositivos conectados (IoT) en entornos militares ha expandido exponencialmente la superficie de ataque, creando nuevos vectores de vulnerabilidad que requieren enfoques de seguridad innovadores y adaptativos.

Los sistemas tradicionales de detección de intrusiones presentan limitaciones críticas para abordar el panorama actual de amenazas. Los sistemas basados en firmas, que constituyen la base de numerosas soluciones comerciales actuales, dependen de bases de datos de patrones conocidos de ataques. Esta aproximación presenta deficiencias significativas: requiere actualizaciones constantes de las bases de datos de firmas, resulta inefectiva contra ataques de día cero o variantes desconocidas, genera volúmenes elevados de falsos positivos en entornos dinámicos, y presenta dificultades para detectar ataques que utilizan técnicas de evasión sofisticadas.

Los sistemas de detección basados en anomalías, aunque teóricamente superiores para detectar ataques desconocidos, enfrentan desafíos significativos en su implementación práctica. Estos sistemas requieren perfiles de comportamiento normal extremadamente precisos, son sensibles a cambios en los patrones de tráfico legítimo, y frecuentemente generan tasas inaceptables de falsos positivos que sobrecargan a los equipos de seguridad (Kumari et al., 2024).

La integración de técnicas de inteligencia artificial y aprendizaje automático en sistemas de ciberseguridad representa una evolución necesaria para superar las limitaciones de los enfoques tradicionales. Los algoritmos de ML pueden procesar y analizar volúmenes masivos de datos de red en tiempo real, identificar patrones complejos que resultan imposibles de detectar manualmente, adaptarse dinámicamente a nuevos tipos de amenazas, y mejorar

continuamente su precisión a través del aprendizaje incremental. Sin embargo, estos sistemas también presentan desafíos propios, incluyendo la posibilidad de generar falsos positivos debido a cambios en los patrones de tráfico normal, o más crítico aún, falsos negativos que permitan que ataques sofisticados pasen desapercibidos (Dini et al., 2023).

Los modelos de ML contemporáneos, incluyendo algoritmos de ensemble como XGBoost, redes neuronales profundas, y máquinas de vectores de soporte, han demostrado capacidades superiores para la clasificación de tráfico de red y la detección de anomalías. Estos modelos pueden aprender representaciones complejas de comportamientos normales y maliciosos, generalizar a partir de datos de entrenamiento para detectar variantes de ataques, y operar con tasas de falsos positivos significativamente menores que los sistemas tradicionales.

El desarrollo de plataformas de computación especializadas para aplicaciones de IA en el borde, como la NVIDIA Jetson Orin Nano, ha creado oportunidades sin precedentes para el despliegue de sistemas inteligentes en entornos con restricciones de recursos. A diferencia de los enfoques basados en servidores centralizados, que requieren conectividad externa constante e introducen latencias de comunicación inaceptables para la detección de amenazas en tiempo real, las plataformas Edge computing permiten el procesamiento local de datos de red sin dependencias externas. Estas plataformas combinan capacidades de procesamiento de IA aceleradas por hardware con factores de forma compactos y eficiencia energética optimizada, características esenciales para aplicaciones militares donde el espacio, el peso y el consumo de energía constituyen factores críticos (Washington Technology, 2025).

1.1 Descripción del Problema

El problema central que aborda este proyecto consiste en la necesidad crítica de desarrollar un sistema de detección de intrusiones inteligente y autónomo que proporcione capacidades de seguridad avanzadas para redes locales en infraestructuras militares. Las

instituciones de defensa nacional enfrentan desafíos complejos para proteger sus redes contra amenazas cibernéticas sofisticadas, utilizando tecnologías que frecuentemente resultan inadecuadas para el panorama actual de riesgos.

La organización cliente para este proyecto corresponde a una institución militar de defensa nacional reconocida, especializada en operaciones de seguridad y defensa del territorio nacional. Esta institución opera múltiples instalaciones distribuidas geográficamente, maneja información clasificada de alto valor estratégico, y requiere sistemas de seguridad cibernética que cumplan con estándares militares rigurosos. La institución ha identificado vulnerabilidades significativas en sus sistemas actuales de detección de intrusiones y busca implementar soluciones tecnológicas avanzadas que mejoren su postura de seguridad cibernética.

Los requerimientos específicos del problema incluyen el desarrollo de un sistema que debe detectar amenazas cibernéticas en tiempo real con alta precisión, operar de manera completamente autónoma sin dependencia de conectividad externa, funcionar continuamente en entornos con recursos computacionales limitados, adaptarse dinámicamente a nuevos tipos de amenazas, y generar alertas accionables con latencias mínimas. El sistema debe procesar volúmenes significativos de tráfico de red, analizar patrones complejos de comportamiento, y distinguir eficazmente entre actividad legítima y maliciosa.

Las restricciones operacionales del proyecto incluyen limitaciones estrictas de seguridad operacional (OPSEC) que prohíben el uso de servicios externos o conectividad a internet, restricciones de recursos de hardware que limitan la capacidad de procesamiento y memoria disponible, requisitos de resistencia ambiental para operación en condiciones adversas, y necesidades de mantenimiento mínimo debido a la ubicación remota de muchas instalaciones.

Las variables de interés primarias incluyen la precisión de detección de amenazas medida a través de métricas como precisión, recall y F1-score, la tasa de falsos positivos que

debe mantenerse por debajo de umbrales aceptables para evitar fatiga de alertas, el tiempo de respuesta del sistema desde la detección hasta la generación de alertas, el consumo de recursos computacionales incluyendo CPU, GPU y memoria, y la capacidad de adaptación del sistema a nuevos patrones de ataque (Khraisat et al., 2019).

La importancia del problema radica en varios factores críticos que convergen para crear una necesidad imperativa de soluciones avanzadas. Primero, las amenazas cibernéticas contra infraestructuras militares han incrementado exponencialmente tanto en frecuencia como en sofisticación, creando riesgos estratégicos de seguridad nacional. Segundo, los sistemas tradicionales de detección de intrusiones han demostrado ser inadecuados para detectar amenazas contemporáneas, particularmente ataques de día cero y campañas APT sofisticadas. Tercero, la naturaleza crítica de las operaciones militares requiere sistemas de seguridad que operen con niveles de confiabilidad y precisión superiores a los estándares comerciales convencionales.

La actualidad del problema se evidencia en el incremento documentado de incidentes cibernéticos contra infraestructuras militares a nivel global, la evolución constante de técnicas de ataque que superan las capacidades de los sistemas existentes, y la disponibilidad reciente de tecnologías de IA y hardware especializado que hacen viable el desarrollo de soluciones avanzadas. Los reportes de seguridad nacional indican que los ciberataques contra infraestructuras críticas han experimentado un incremento significativo durante los últimos años, mientras que los sistemas tradicionales de detección enfrentan crecientes desafíos para mantener su efectividad ante amenazas cada vez más sofisticadas (Industrial Cyber, 2024).

El problema es susceptible de observación y medición a través de múltiples metodologías. La efectividad del sistema puede medirse utilizando datasets estándar de la industria como CICIDS2017 y NSL-KDD, que proporcionan benchmarks reconocidos para la evaluación de sistemas de detección de intrusiones; en particular, el dataset CICIDS2017,

desarrollado por el Canadian Institute for Cybersecurity, incluye tráfico benigno y ataques comunes actualizados, lo que lo hace representativo de escenarios del mundo real (Sharafaldin et al., 2018). El rendimiento puede observarse mediante métricas cuantitativas de precisión, recall, F1-score, y análisis de curvas ROC (Receiver Operating Characteristic), que representan gráficamente la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos, permitiendo evaluar el rendimiento del clasificador en diferentes umbrales de decisión. La eficiencia operacional puede medirse a través del monitoreo de recursos computacionales, tiempos de respuesta, y análisis de throughput de red.

Las capacidades de análisis incluyen la evaluación comparativa con sistemas existentes, análisis de sensibilidad para diferentes tipos de ataques, estudios de ablación para determinar la contribución de diferentes componentes del modelo, y análisis de robustez bajo diversas condiciones operacionales, ya que la evaluación de datasets juega un papel vital en la validación de cualquier enfoque de IDS, permitiendo evaluar la capacidad del método propuesto para detectar comportamiento intrusivo (Khraisat et al., 2019). El proyecto permitirá análisis longitudinales del comportamiento del sistema, estudios de adaptabilidad a amenazas emergentes, y evaluación de la escalabilidad de la solución.

1.2 Justificación del Problema

La justificación para resolver este problema se fundamenta en múltiples dimensiones críticas que convergen para crear una necesidad imperativa de desarrollo de soluciones avanzadas de seguridad cibernética para entornos militares.

Desde la perspectiva de seguridad nacional, la protección de infraestructuras militares contra amenazas cibernéticas constituye una prioridad estratégica fundamental. Los sistemas de defensa nacional almacenan y procesan información clasificada cuyo compromiso podría resultar en consecuencias devastadoras para la seguridad del país. La naturaleza evolutiva de

las amenazas cibernéticas requiere soluciones tecnológicas que puedan adaptarse dinámicamente a nuevos vectores de ataque, superando las limitaciones de los enfoques tradicionales basados en firmas estáticas.

La dimensión tecnológica de la justificación radica en la convergencia de varios factores habilitadores. El desarrollo de algoritmos de inteligencia artificial avanzados ha alcanzado un nivel de madurez que permite su aplicación efectiva en dominios de seguridad crítica. La disponibilidad de plataformas de computación en el borde especializadas, como la NVIDIA Jetson Orin Nano, proporciona las capacidades de procesamiento necesarias para ejecutar modelos de IA complejos en entornos con restricciones de recursos. La proliferación de datasets de alta calidad para entrenamiento de modelos de detección de intrusiones permite el desarrollo de sistemas con capacidades de generalización superiores.

La justificación económica se basa en el costo potencial de incidentes cibernéticos exitosos contra infraestructuras militares. Según el reporte "The Hidden Costs of Cybercrime" del Center for Strategic and International Studies en asociación con McAfee, las pérdidas globales por cibercrimen se acercan a 1 billón de dólares, representando más del 1% del PIB mundial, lo que subraya el impacto económico significativo de los incidentes cibernéticos en infraestructuras críticas (McAfee & CSIS, 2020). Los costos asociados incluyen no solo los daños directos a sistemas y datos, sino también los costos de remediación, pérdida de capacidades operacionales, comprometimiento de operaciones en curso, y potencial exposición de información clasificada. El desarrollo de un sistema de detección avanzado representa una inversión preventiva que puede evitar costos exponencialmente mayores asociados con incidentes de seguridad exitosos.

Desde una perspectiva operacional, los sistemas militares requieren capacidades de seguridad que operen de manera autónoma en entornos desafiantes. Las ubicaciones remotas de muchas instalaciones militares, combinadas con restricciones de personal especializado,

crean la necesidad de sistemas que puedan funcionar efectivamente con mínima intervención humana. La capacidad de detectar y responder a amenazas en tiempo real, sin depender de conectividad externa o soporte remoto, constituye un requisito operacional crítico que justifica el desarrollo de soluciones especializadas.

La justificación científica y académica del proyecto radica en su contribución al avance del conocimiento en la intersección de la inteligencia artificial y la ciberseguridad. El proyecto aborda desafíos técnicos significativos relacionados con la optimización de modelos de IA para hardware con restricciones de recursos, el desarrollo de técnicas de detección adaptativas, y la validación de sistemas de seguridad en entornos operacionales realistas. Los resultados del proyecto contribuirán al cuerpo de conocimiento académico y proporcionarán metodologías y mejores prácticas aplicables en otros contextos críticos.

La urgencia temporal de la justificación se fundamenta en la escalada continua de amenazas cibernéticas y la brecha creciente entre las capacidades de ataque y defensa. Reportes públicos de agencias como el FBI Internet Crime Complaint Center documentan que en 2023 se recibieron 880,418 quejas con pérdidas potenciales que excedieron los $12.5 mil millones, representando un aumento de 22% en pérdidas comparado con 2022 (FBI IC3, 2023). Análisis de tendencias del MITRE ATT&CK framework indican que los actores de amenazas están adoptando técnicas automatizadas y herramientas potenciadas por inteligencia artificial para mejorar sus capacidades de reconocimiento, evasión de detección y explotación de vulnerabilidades (MITRE, 2024), creando una carrera armamentística cibernética donde las organizaciones defensoras deben implementar contramedidas tecnológicas avanzadas para mantener la paridad.

La viabilidad técnica del proyecto se justifica a través de la convergencia de múltiples factores tecnológicos maduros. Los algoritmos de aprendizaje automático han demostrado efectividad superior en tareas de clasificación y detección de anomalías en dominios similares.

Los resultados experimentales muestran que métodos como Random Forest y Autoencoder son altamente efectivos en la detección de anomalías, con Random Forest alcanzando precisiones de hasta 99.9% en tareas de clasificación supervisada y Autoencoder logrando 99.2697% de precisión en enfoques no supervisados (Krzysztoń et al., 2024). Las plataformas de hardware especializadas proporcionan las capacidades computacionales necesarias. Los datasets de entrenamiento de alta calidad, como CICIDS2017 del Canadian Institute for Cybersecurity, UNSW-NB15 de la Universidad de New South Wales y NSL-KDD, se encuentran disponibles en repositorios académicos y proporcionan datos etiquetados con ataques modernos, características extraídas automáticamente y configuraciones de entrenamiento y prueba validadas, lo que los convierte en recursos fundamentales para el desarrollo de sistemas de detección de intrusiones (IDS). Las herramientas de desarrollo y frameworks de software han alcanzado un nivel de madurez que facilita la implementación eficiente de soluciones complejas.

La justificación se refuerza por el potencial de impacto transformador de la solución. Un sistema de detección de intrusiones basado en IA exitoso no solo proporcionará capacidades de seguridad mejoradas para la organización cliente, sino que también establecerá un precedente para la adopción de tecnologías avanzadas en el sector de seguridad nacional. La solución puede servir como modelo para implementaciones similares en otras instituciones militares y de seguridad, multiplicando el impacto del proyecto.

En conclusión, la justificación para resolver este problema se basa en la convergencia de necesidades operacionales críticas, disponibilidad de tecnologías habilitadoras maduras, potencial de impacto significativo en la seguridad nacional, y oportunidades de contribución al avance del conocimiento científico y tecnológico. La urgencia de las amenazas cibernéticas contemporáneas y las limitaciones de los enfoques tradicionales crean una ventana de

oportunidad para el desarrollo de soluciones innovadoras que pueden transformar fundamentalmente las capacidades de seguridad cibernética en entornos militares.

1.3 Objetivos

Los objetivos del presente proyecto es estructurar de manera jerárquica para abordar de forma sistemática el desarrollo del sistema de detección de intrusiones propuesto. El objetivo general establece el propósito principal del proyecto, mientras que los objetivos específicos detallan las metas particulares que deben alcanzarse para cumplir con objetivo principal.

1.3.1 Objetivo general

Desarrollar y validar un modelo de inteligencia artificial integrado en un prototipo de Sistema de Detección de Intrusiones (IDS) para identificar amenazas cibernéticas en redes locales militares mediante análisis de tráfico de red en tiempo real.

1.3.2 Objetivos específicos

1. Establecer el marco teórico y metodológico mediante revisión bibliográfica exhaustiva sobre sistemas de detección de intrusiones, técnicas de inteligencia artificial aplicadas a ciberseguridad y datasets de tráfico de red para fundamentar el diseño del modelo propuesto.
2. Desarrollar y evaluar modelos de inteligencia artificial a través del preprocesamiento de datasets especializados, aplicación de técnicas de machine learning y deep learning, y evaluación del rendimiento mediante métricas de precisión, recall y F1-score para seleccionar el modelo óptimo de clasificación de amenazas.
3. Implementar un prototipo funcional de software IDS que integre el modelo de IA seleccionado con capacidades de captura de tráfico en tiempo real, preprocesamiento automático y generación de alertas para demostrar la viabilidad del sistema.

4. Validar el rendimiento del modelo y prototipo mediante pruebas en escenarios controlados que simulen tráfico normal y ataques cibernéticos comunes para evaluar la efectividad, precisión y establecer las bases para la implementación futura en hardware embebido.

1.4 Marco Teórico

Este marco teórico aborda los conceptos fundamentales de los Sistemas de Detección de Intrusiones (IDS) y su evolución mediante la inteligencia artificial, así como los datasets y plataformas de hardware especializado que son cruciales para su desarrollo o implementación en infraestructuras críticas.

1.4.1 Sistemas de detección de intrusiones (IDS)

Los Sistemas de Detección de Intrusiones constituyen una tecnología fundamental en la ciberseguridad moderna, diseñada para identificar y alertar sobre actividades maliciosas o no autorizadas en redes de computadoras. Según Scarfone y Mell (2007), los IDS se clasifican en dos categorías principales: sistemas basados en red (NIDS) que monitorean el tráfico de red, y sistemas basados en host (HIDS) que supervisan actividades en sistemas individuales.

La evolución de estos sistemas ha sido documentada extensamente por investigadores como Liao et al. (2013), quienes identifican tres generaciones de IDS: la primera basada en firmas, la segunda en anomalías estadísticas, y la tercera incorporando técnicas de inteligencia artificial. Esta progresión refleja la necesidad de adaptarse a amenazas cada vez más sofisticadas y evasivas.

1.4.2 *Aplicación de inteligencia artificial en ciberseguridad*

La integración de técnicas de inteligencia artificial en ciberseguridad ha experimentado un crecimiento significativo en la última década. Buczak y Guven (2016) realizaron una

revisión exhaustiva de métodos de machine learning aplicados a detección de intrusiones, identificando algoritmos como Support Vector Machines (SVM), Random Forest, y redes neuronales como los más efectivos para la clasificación de amenazas.

Investigaciones recientes de Aldweesh et al. (2020) demuestran que los modelos de deep learning, particularmente las redes neuronales profundas y los autoencoders, superan consistentemente a los métodos tradicionales en la detección de ataques zero-day y variantes de malware. Estos estudios establecen que los modelos de IA pueden identificar patrones complejos en grandes volúmenes de datos de tráfico de red que serían imposibles de detectar mediante métodos convencionales.

1.4.3 *Redes Neuronales Convolucionales 1D (1D-CNN)*

Las Redes Neuronales Convolucionales unidimensionales representan una adaptación de las CNN tradicionales para el procesamiento de datos secuenciales como el tráfico de red. A diferencia de las CNN bidimensionales utilizadas en procesamiento de imágenes, las 1D-CNN operan sobre secuencias temporales aplicando filtros convolucionales que pueden detectar patrones locales en los datos.

En el contexto de ciberseguridad, las 1D-CNN han demostrado efectividad superior para la extracción automática de características relevantes del tráfico de red. Kim et al. (2018) demostraron que las 1D-CNN pueden identificar automáticamente firmas de ataques sin requerir ingeniería manual de características, superando en precisión a métodos tradicionales de machine learning.

1.4.4 *Redes LSTM (Long Short-Term Memory)*

Las redes LSTM constituyen una arquitectura especializada de redes neuronales recurrentes diseñada para superar el problema del desvanecimiento del gradiente en secuencias largas. Introducidas por Hochreiter y Schmidhuber (1997), las LSTM incorporan mecanismos

de compuertas que regulan el flujo de información, permitiendo el mantenimiento selectivo de información relevante a largo plazo.

La arquitectura LSTM incluye tres tipos de compuertas: la compuerta de olvido que determina qué información descartar del estado celular, la compuerta de entrada que decide qué nueva información almacenar, y la compuerta de salida que controla qué partes del estado celular utilizar para generar la salida. Esta estructura permite el procesamiento efectivo de secuencias largas manteniendo información relevante para la clasificación de amenazas.

1.4.5 Modelos Híbridos 1D-CNN + LSTM

Los modelos híbridos que combinan 1D-CNN y LSTM representan un enfoque avanzado que aprovecha las fortalezas complementarias de ambas arquitecturas. Las capas convolucionales extraen características locales relevantes del tráfico de red, mientras que las capas LSTM capturan dependencias temporales a largo plazo entre estas características.

Investigaciones de Tang et al. (2020) demostraron que los modelos híbridos 1D-CNN+LSTM superan significativamente el rendimiento de modelos individuales en tareas de detección de intrusiones. La combinación permite la extracción automática de características espaciales y temporales sin requerir preprocesamiento manual extensivo.

1.4.6 *Datasets y benchmarks para evaluación de IDS*

La evaluación rigurosa de sistemas IDS requiere datasets estandarizados que representen condiciones realistas de tráfico de red. El dataset CICIDS2017, desarrollado por Sharafaldin et al. (2018), ha emergido como un estándar de facto en la comunidad científica, conteniendo tráfico de red capturado durante cinco días con ataques contemporáneos como DDoS, ataques de fuerza bruta, infiltración, y botnet.

Anteriormente, el dataset NSL-KDD, una versión mejorada del clásico KDD Cup 99, fue ampliamente utilizado por investigadores como Tavallaee et al. (2009) para evaluar

algoritmos de detección. Sin embargo, estudios posteriores de Ring et al. (2019) han señalado limitaciones en datasets más antiguos debido a su falta de representatividad con respecto a ataques modernos y patrones de tráfico actuales.

1.4.7 *Desafíos en infraestructuras militares*

La ciberseguridad en contextos militares presenta requisitos únicos documentados por organizaciones como NIST y el Departamento de Defensa de Estados Unidos. Según Cybersecurity Framework del NIST (2018), las infraestructuras críticas requieren sistemas de detección con capacidades de operación continua, alta disponibilidad y resistencia a ataques sofisticados.

Estudios de caso realizados por Singer y Friedman (2014) en "Cybersecurity and Cyberwar" documentan la evolución de amenazas en entornos militares, desde ataques convencionales hasta amenazas persistentes avanzadas (APT). Estos análisis subrayan la necesidad de sistemas IDS adaptativos capaces de evolucionar con el panorama de amenazas, justificando el enfoque basado en IA del presente proyecto.

La literatura revisada establece que la convergencia de técnicas de inteligencia artificial, hardware embebido especializado y datasets contemporáneos representa una oportunidad significativa para desarrollar sistemas IDS de próxima generación, particularmente para aplicaciones en infraestructuras críticas donde la detección temprana y precisa de amenazas es fundamental para la seguridad nacional.

Capítulo 2

2 Metodología

Este capítulo detalla la estrategia metodológica seguida para el diseño y la validación del prototipo de Sistema de Detección de Intrusiones (IDS). Se describe la formulación y evaluación de las alternativas de solución, el diseño conceptual y detallado del sistema, la selección de recursos tecnológicos, y la rigurosa metodología de evaluación que permitió identificar la arquitectura de Inteligencia Artificial más eficaz para el contexto de la ciberseguridad militar.

2.1 Formulación de Alternativas de Solución

El desafío principal fue desarrollar un sistema capaz de detectar amenazas avanzadas con alta precisión y en tiempo real, superando las limitaciones de los enfoques tradicionales que son ineficaces ante ataques novedosos o de día cero. La naturaleza crítica de las redes militares demandó un sistema capaz de detectar amenazas sofisticadas con alta precisión y mínimas tasas de falsos positivos. En lugar de evaluar alternativas teóricas, la metodología se centró en una competición experimental entre diferentes arquitecturas de Deep Learning para determinar empíricamente cuál era la más idónea.

Se formularon cuatro alternativas, cada una representada por un modelo de red neuronal con capacidades distintas, para ser entrenados y evaluados bajo las mismas condiciones.

Alternativa 1: Modelo Híbrido 1D-CNN + LSTM. Esta fue la alternativa principal, donde la hipótesis planteaba que la combinación sinérgica de la extracción de características espaciales locales proporcionada por la CNN con el modelado de secuencias temporales de la LSTM permitiría detectar ataques complejos y coordinados que se desarrollan en múltiples pasos, algo que los modelos no secuenciales no podrían capturar eficazmente.

Alternativa 2: Red Neuronal Artificial (ANN). Se implementó un modelo de red neuronal denso tradicional como línea base. Su propósito era establecer un umbral de

rendimiento fundamental para cuantificar de manera objetiva la ganancia en eficacia obtenida por las arquitecturas recurrentes y la híbrida, que son computacionalmente más costosas.

Alternativa 3: Modelo Bi-LSTM. Se evaluó un modelo basado en LSTM Bidireccional, considerando que al procesar las secuencias de datos en ambas direcciones (hacia adelante y hacia atrás), el modelo podría capturar un contexto más completo de los patrones de ataque, potencialmente mejorando la precisión en la detección de amenazas cuyas pistas clave no se encuentran al inicio de la secuencia de datos.

Alternativa 4: Modelo Bi-GRU. Similar a la anterior, esta alternativa utilizó unidades GRU (Gated Recurrent Unit) bidireccionales, una variante más ligera que las LSTM, cuyo objetivo era investigar si se podía alcanzar un rendimiento comparable al de la Bi-LSTM pero con una menor complejidad computacional y, por ende, tiempos de entrenamiento más rápidos, un factor relevante para futuros reentrenamientos del sistema.

2.2 Selección de la Mejor Alternativa

La selección de la mejor alternativa se realizó tras una fase de evaluación exhaustiva, utilizando un conjunto de métricas de rendimiento críticas para la seguridad. La decisión se basó en los resultados empíricos obtenidos de la competición de modelos, donde el modelo híbrido 1D-CNN + LSTM fue seleccionado como la solución superior.

Esta selección se justificó por su destacada capacidad para obtener el mejor equilibrio entre todas las métricas, especialmente el F1-Score, y su habilidad para minimizar los Falsos Negativos (ataques no detectados), el tipo de error más crítico en un entorno militar. En la figura 2.1, extraído de los resultados experimentales, ilustra visualmente la superioridad del modelo CNN-LSTM en las métricas clave en comparación con las otras arquitecturas evaluadas.

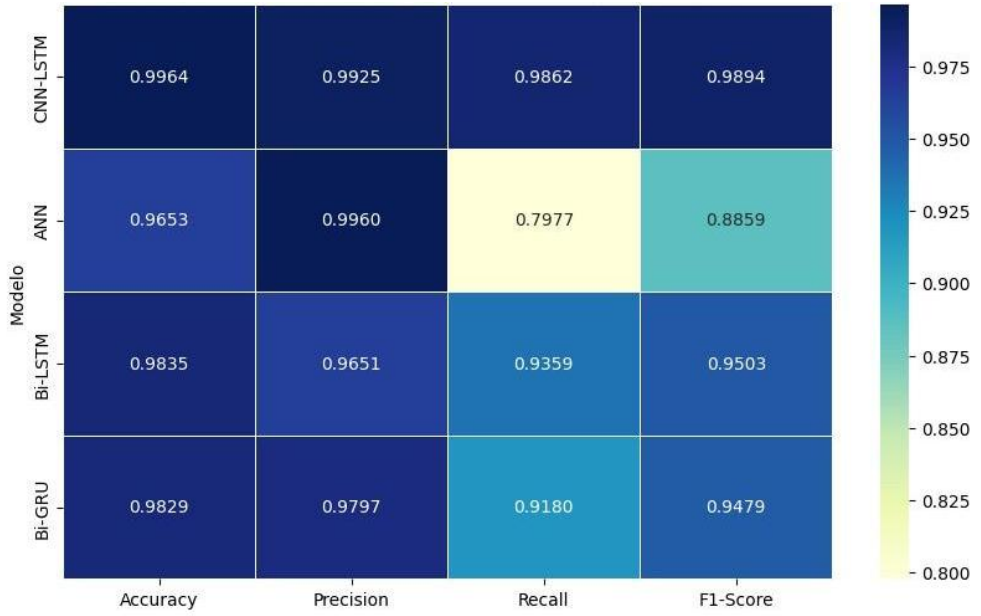

Figura 2.1 Evaluación comparativa de métricas de rendimiento en los diferentes modelos

2.3 Diseño Conceptual del Sistema

El diseño conceptual del sistema se estructuró en cuatro módulos principales interconectados para facilitar el mantenimiento, la escalabilidad y la integración. El flujo de datos fue diseñado siguiendo un pipeline de procesamiento en tiempo real, donde los paquetes de red son capturados, procesados y clasificados por el modelo de IA, tal como se ilustra en la figura 2.2.

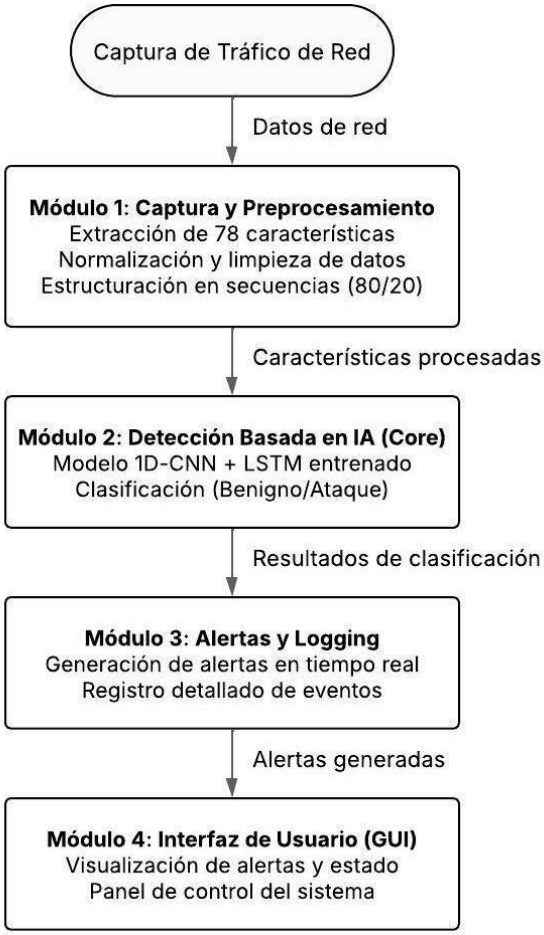

Figura 2.2 Arquitectura Conceptual del Prototipo IDS

Este diseño conceptual guio el desarrollo de un prototipo funcional capaz de analizar tráfico y generar alertas basadas en las predicciones del modelo de IA seleccionado.

2.4 Metodología de Desarrollo Experimental

Se adoptó una metodología mixta que combinó una rigurosa investigación experimental con un desarrollo de software iterativo. La estrategia de Machine Learning se basó en el framework CRISP-DM, adaptado al dominio de la ciberseguridad.

a) **Comprensión del Negocio y los Datos:** Se analizaron los requerimientos para un IDS militar y se condujo un análisis exploratorio de varios datasets, incluyendo UNSW-NB15 y NSL-KDD. Se determinó que CICIDS2017 era el más adecuado para el

entrenamiento debido a que, como se muestra en la siguiente gráfica en la figura 2.3, posee un volumen y una diversidad de ataques significativamente mayores, lo que permite entrenar un modelo más robusto y generalizable.

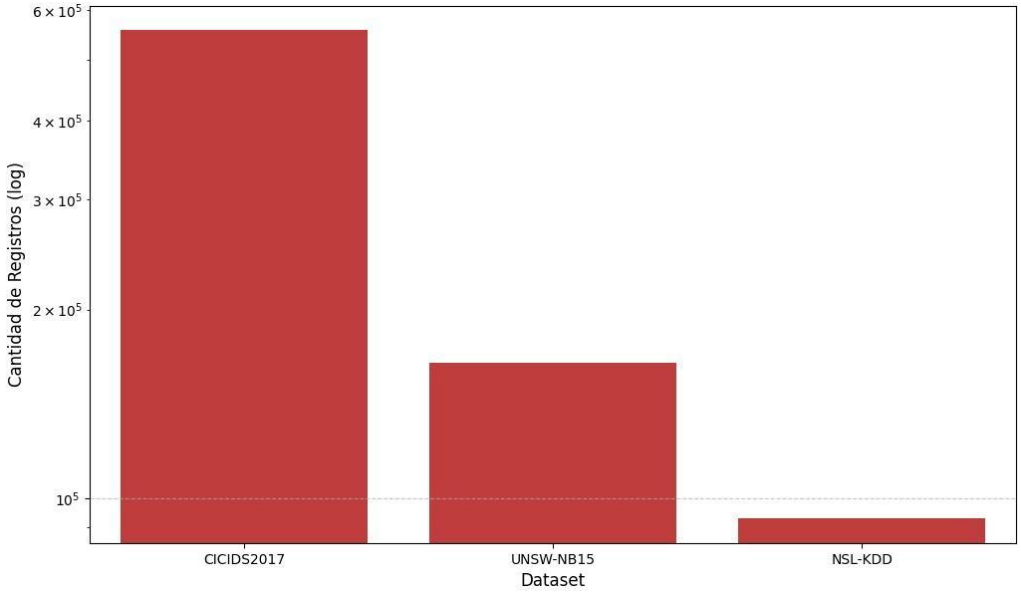

Figura 2.3 Comparación de la cantidad de registros de ataque por Dataset

b) **Preparación de los Datos:** Se desarrolló un pipeline de preprocesamiento que incluyó la consolidación y limpieza de los datos de CICIDS2017, la codificación de etiquetas a formato binario (0 para benigno, 1 para ataque), y la normalización de todas las características numéricas.

c) **Modelado y Experimentación:** Se construyeron las cuatro arquitecturas de Deep Learning (CNN-LSTM, ANN, Bi-LSTM, Bi-GRU) y se entrenaron usando los datos de CICIDS2017.

d) **Evaluación:** Se realizó la evaluación comparativa que llevó a la selección del modelo CNN-LSTM, como se detalló en la sección 2.2.

e) **Despliegue (Prototipo):** El modelo ganador se integró en un script de Python que simulaba un prototipo funcional, validando la solución completa.

2.5 Diseño Detallado del Modelo Seleccionado (1D-CNN + LSTM)

La arquitectura del modelo híbrido fue diseñada para procesar secuencialmente los datos de tráfico de red, combinando la extracción de características con el análisis de dependencias temporales. El flujo de datos a través de las capas del modelo se detalla en el siguiente diagrama mostrado en la figura 2.4.

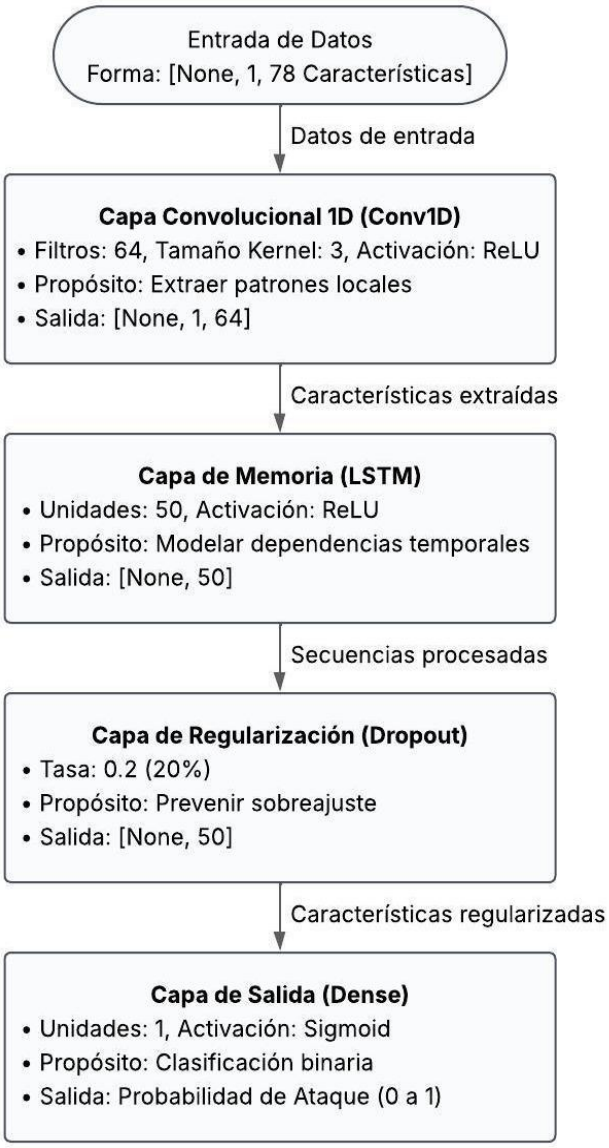

Figura 2.4 Arquitectura Detallada del Modelo 1D-CNN + LSTM

La sinergia de esta arquitectura fue clave: la capa Conv1D actuó como un extractor de características de bajo nivel, identificando patrones locales en las 78 características de entrada. La salida de esta capa, un conjunto refinado de 64 características fue luego alimentada a la capa

LSTM, que está especializada en encontrar relaciones a lo largo del tiempo en estas secuencias de características, permitiendo así la detección de ataques coordinados y persistentes.

2.6 Selección de Recursos y Tecnologías

Dataset Principal: Se seleccionó el dataset CICIDS2017 como la fuente principal de datos para el entrenamiento y la evaluación del modelo. Esta decisión se basó en su representatividad de ataques contemporáneos, su gran volumen de datos y su diversidad de vectores de ataque, características que son esenciales para desarrollar un modelo robusto capaz de generalizar a diferentes tipos de amenazas.

Plataforma y Herramientas: Se implementó un conjunto de herramientas de código abierto, estándar en la industria, para maximizar la reproducibilidad y eficiencia del proyecto.

- **Google Colab:** Se utilizó como la plataforma de desarrollo principal, proporcionando acceso gratuito a recursos de GPU que fue fundamental para acelerar significativamente el proceso de entrenamiento de los modelos complejos, reduciendo los tiempos de experimentación de días a horas.
- **Frameworks:** Se empleó la dupla TensorFlow y Keras, donde Keras, con su API de alto nivel, facilitó la implementación rápida de prototipos y la experimentación ágil con las diferentes configuraciones arquitectónicas evaluadas.
- **Librerías:** Se integró el ecosistema estándar de ciencia de datos en Python, incluyendo Pandas y NumPy para la manipulación y procesamiento de datos, y Scikit-learn para tareas de preprocesamiento, división de datos y cálculo de métricas de rendimiento.

2.7 Especificaciones Técnicas y de Evaluación

Las especificaciones técnicas fueron definidas de manera integral para cubrir todos los aspectos del desarrollo, evaluación y despliegue del sistema de detección de intrusiones basado en deep learning.

2.7.1 Métricas de evaluación y su significado operacional

Se estableció un conjunto completo de métricas para obtener una visión multidimensional del rendimiento del modelo, cada una con un significado operacional claro en el contexto de la ciberseguridad militar.

Accuracy (Exactitud): Representa la proporción total de predicciones correctas, aunque puede ser engañosa en datasets desbalanceados al no reflejar adecuadamente el rendimiento en clases minoritarias.

Precision (Precisión): Mide cuántas de las alertas de "ataque" generadas fueron realmente ataques, siendo crítica para mantener la confianza del operador y minimizar el tiempo perdido investigando falsas alarmas.

Recall (Sensibilidad): Mide cuántos de los ataques reales que ocurrieron fueron detectados por el sistema, siendo esencial para no pasar por alto amenazas reales que podrían tener consecuencias graves.

F1-Score: Representa la media armónica de Precisión y Recall, constituyendo la métrica más importante para un IDS al buscar el equilibrio óptimo entre no abrumar a los operadores con falsas alarmas y no fallar en detectar ataques reales.

AUC-ROC: Mide la capacidad del modelo para discriminar entre tráfico benigno y malicioso a través de todos los posibles umbrales de decisión, proporcionando una evaluación integral del rendimiento clasificatorio.

2.7.2 Protocolo de validación

Se implementó una estrategia de validación robusta para asegurar la imparcialidad de los resultados, dividiendo los datos del dataset CICIDS2017 en 80% para el conjunto de entrenamiento y 20% para el conjunto de prueba, manteniendo este último completamente aislado durante todo el proceso de entrenamiento y ajuste para utilizarlo una única vez al final. Este protocolo garantiza que el rendimiento reportado refleje fielmente la capacidad del modelo para generalizar a datos completamente nuevos y no vistos previamente.

2.7.3 Requerimientos del sistema

Las especificaciones fueron definidas para la fase de desarrollo y el futuro despliegue.

1. **Hardware:** Procesador Intel Core i5 o AMD equivalente, mínimo 8GB de RAM (16GB recomendados) y 50GB de almacenamiento.
2. **Software:** El entorno de desarrollo se basó en las siguientes versiones, compatibles con Ubuntu 20.04 LTS o Windows 10/11:
   - Python: 3.11.13
   - TensorFlow: 2.18.0
   - NumPy: 2.1.3
   - Pandas: Utilizado extensivamente en el entorno de desarrollo de Google Colab, pero no es una dependencia estricta para el script de despliegue final del prototipo.

2.8 Consideraciones Éticas y Legales

El desarrollo del sistema consideró principios éticos fundamentales relacionados con la privacidad de datos y el uso responsable de tecnologías de inteligencia artificial. Se implementaron medidas para asegurar que el procesamiento de datos de red se realizara

exclusivamente con fines de detección de amenazas, sin comprometer la privacidad de usuarios legítimos, utilizando para ello datasets públicos y anonimizados.

El diseño del sistema consideró los marcos regulatorios aplicables y se establecieron principios para el uso responsable del sistema, incluyendo la necesidad de supervisión humana en la toma de decisiones críticas de seguridad, la transparencia en el funcionamiento del sistema y procedimientos para la evaluación continua del rendimiento en entornos operacionales.

Capítulo 3

3 Introducción a la Fase Experimental

Este capítulo detalla los resultados de la fase experimental, diseñada para evaluar y comparar de manera cuantitativa el rendimiento de cuatro arquitecturas de redes neuronales. El propósito principal consiste en seleccionar el modelo más robusto y fiable para la implementación de un prototipo de Sistema de Detección de Intrusiones (IDS) para redes militares, cuyo análisis se centra no solo en la precisión final, sino también en la estabilidad del rendimiento y, de manera crítica, en la capacidad de minimizar los errores que representan mayor riesgo para la seguridad operacional.

3.1 Rendimiento Comparativo de los Modelos

Los cuatro modelos propuestos (CNN + LSTM, ANN, Bi-LSTM y Bi-GRU) fueron entrenados y evaluados bajo condiciones idénticas con el dataset CICIDS2017, asegurando que las diferencias de rendimiento se deban exclusivamente a las capacidades de cada arquitectura.

3.1.1 Análisis de métricas de clasificación

La efectividad de un IDS no se mide solo por la exactitud (Accuracy), dado que los datasets de tráfico de red suelen estar desbalanceados. Por ello, se priorizó un análisis multidimensional que incluye Precisión, Sensibilidad (Recall) y, de forma destacada, el F1-Score, que equilibra ambas métricas. La Tabla 3.1 resume el rendimiento de cada modelo en el conjunto de prueba:

Tabla 3.1

Comparación de Métricas de Rendimiento por Modelo

Modelo	Accuracy	Precision	Recall	F1-Score
CNN-LSTM	0.9964	0.9925	0.9862	0.9894

ANN	0.9653	0.996	0.7977	0.8859
Bi-LSTM	0.9835	0.9651	0.9359	0.9503
Bi-GRU	0.9829	0.9797	0.918	0.9479

De los resultados se desprende que el modelo híbrido CNN + LSTM supera a las demás arquitecturas en las métricas más críticas. Aunque el modelo ANN muestra una precisión muy alta, su bajo Recall (0.7977) indica que falla en detectar más del 20% de los ataques reales, un riesgo inaceptable en un entorno de alta seguridad. Los modelos Bi-LSTM y Bi-GRU ofrecen un rendimiento sólido y balanceado, pero no alcanzan el equilibrio superior del modelo híbrido. Para complementar el análisis, se evaluó la estabilidad de cada modelo durante la fase de validación. La Tabla 3.2 presenta el promedio y la desviación estándar de las métricas clave durante las últimas 10 épocas de entrenamiento, donde una desviación estándar baja indica que el rendimiento del modelo es consistente y no sufre de fluctuaciones abruptas, lo que se traduce en mayor fiabilidad.

Tabla 3.2

Estabilidad del Rendimiento de los Modelos (Promedio ± Desviación Estándar en las Últimas 10 Épocas de Validación)

Modelo	**Validation Accuracy**	**Validation Precision**	**Validation Recall**
CNN-LSTM	0.9944 ± 0.0032	0.9838 ± 0.0182	0.9835 ± 0.0060
ANN	0.9628 ± 0.0021	0.9966 ± 0.0024	0.7825 ± 0.0136
Bi-LSTM	0.9832 ± 0.0012	0.9634 ± 0.0039	0.9362 ± 0.0105
Bi-GRU	0.9784 ± 0.0047	0.9578 ± 0.0234	0.9131 ± 0.0144

El análisis de estabilidad confirma la superioridad del modelo CNN + LSTM, el cual no solo alcanzó las mejores métricas promedio en Accuracy y Recall, sino que también mostró

una desviación estándar muy baja, especialmente en Recall (±0.0060), lo que demuestra un aprendizaje consistente y fiable.

3.1.2 Análisis de errores de clasificación: falsos positivos vs. falsos negativos

En la ciberseguridad militar, el error más crítico es el Falso Negativo (FN), que representa un ataque real no detectado e implica vulnerabilidades catastróficas. Los Falsos Positivos (FP), aunque menos críticos que los FN, generan alertas innecesarias que consumen recursos operativos y pueden ser explotados como vector de ataque para ahogar los recursos del objetivo mediante la saturación del sistema de respuesta.

Tabla 3.3

Comparación de Errores de Clasificación por Modelo

Modelo	Falsos Positivos (FP)	Falsos Negativos (FN)
CNN + LSTM	633	1174
ANN	275	17227
Bi-LSTM	2878	5460
Bi-GRU	1616	6981

La Tabla 3.3 revela que el modelo CNN + LSTM obtuvo la menor cantidad de Falsos Negativos (1174), validando su robustez para detectar amenazas reales. Aunque la ANN generó menos FPs, su alarmante cifra de 17,227 ataques no detectados lo hace completamente inviable, mientras que los modelos Bi-LSTM y Bi-GRU registraron entre 4 y 6 veces más FNs que el modelo híbrido. La capacidad de minimizar el error más perjudicial fue un factor decisivo en la selección del modelo CNN + LSTM.

3.2 Análisis Detallado del Modelo Seleccionado: 1D-CNN + LSTM

Tras confirmar su superioridad en las métricas comparativas, se realizó un análisis más profundo del modelo híbrido para comprender su comportamiento y validar su idoneidad.

3.2.1 Estabilidad y convergencia del entrenamiento

El análisis de las curvas de entrenamiento y validación a lo largo de las épocas ofrece una visión clara de la estabilidad del aprendizaje del modelo. Tal como se puede observar en la figura 3.1, las curvas del modelo CNN + LSTM muestran una convergencia suave y estable.

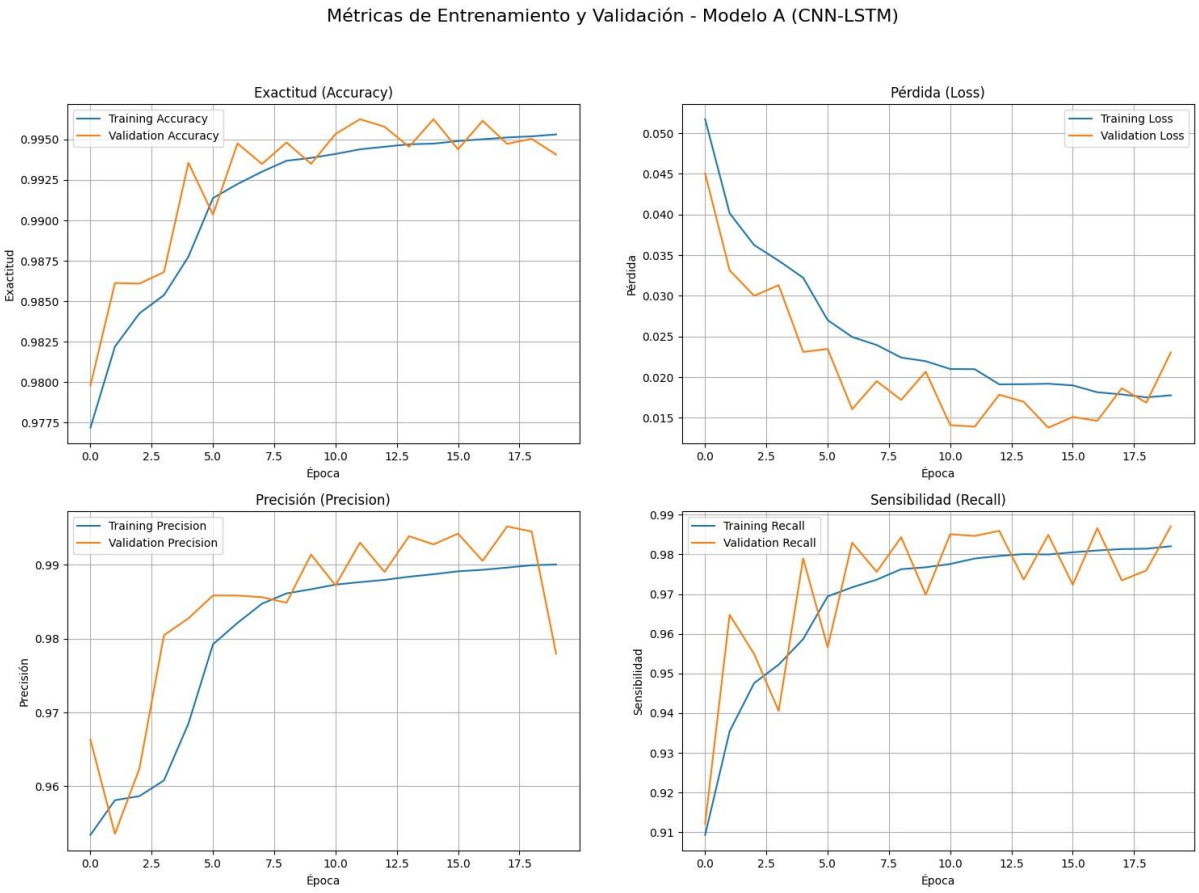

Figura 3.1 Métricas de entrenamiento y validación del Modelo CNN + LSTM

La curva de pérdida de validación (Validation Loss) desciende consistentemente junto a la pérdida de entrenamiento, y ambas se estabilizan en un valor bajo, indicando que el modelo generaliza bien a datos no vistos sin mostrar signos significativos de sobreajuste. Este

comportamiento estable, reflejado en la baja desviación estándar de sus métricas (Tabla 3.2), contrasta con la mayor volatilidad observada en los otros modelos, especialmente en la ANN.

3.2.2 Capacidad de discriminación: curvas ROC y PR

La capacidad del modelo seleccionado para distinguir entre clases se evaluó mediante las curvas ROC (Receiver Operating Characteristic) y Precision-Recall (PR), ilustradas en la figura 3.2.

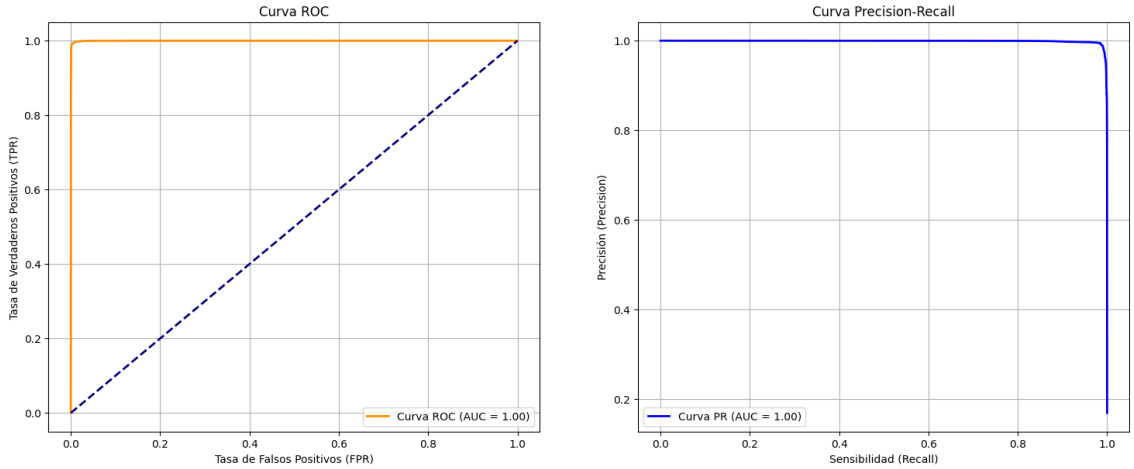

Figura 3.2 Curva ROC y Precision-Recall del modelo evaluado

Curva ROC: El modelo alcanzó un valor de AUC (Area Under the Curve) de 1.00 (redondeado de 0.9998), como se ve en la curva ROC. Un valor tan cercano a 1.0 indica una capacidad de discriminación casi perfecta entre el tráfico benigno y el malicioso a través de todos los posibles umbrales de decisión.

Curva Precision-Recall: Para datasets desbalanceados, la curva PR es a menudo más informativa, y el modelo obtuvo un AUC-PR de 1.00 (redondeado de 0.9990). Esto confirma que el modelo puede mantener una alta precisión y sensibilidad (recall) simultáneamente, lo cual es el escenario ideal para un IDS.

3.2.3 Optimización del umbral de decisión

Por defecto, el umbral de clasificación es de 0.5, pero en aplicaciones de seguridad puede ser beneficioso ajustarlo para priorizar la detección de ataques (Recall) a costa de un ligero aumento en falsas alarmas (menor Precisión). Se realizó una prueba evaluando el rendimiento del modelo CNN + LSTM con un umbral de decisión de 0.2, y la tabla 3.4 compara el rendimiento antes y después de este ajuste.

Tabla 3.4

Efecto del Ajuste del Umbral en el Modelo CNN + LSTM

Métrica	**Umbral 0.5 (Defecto)**	**Umbral 0.2 (Optimizado)**	**Cambio/Impacto**
Falsos Negativos (FN)	1174	524	-55.40% de reducción de riesgo
Falsos Positivos (FP)	633	2277	+259.7% de aumento de alertas
Recall (Sensibilidad)	0.986	0.99	Aumento de la capacidad de detección
F1-Score	0.99	0.98	Rendimiento general se mantiene excelente

Como muestra la Tabla 3.4, este simple ajuste estratégico permitió reducir los ataques no detectados en más de un 55%. Si bien esto incrementó el número de falsas alarmas, el F1-Score se mantuvo en un extraordinario 0.98, demostrando que el modelo puede ser calibrado para un perfil de seguridad más agresivo sin sacrificar su rendimiento general.

3.2.4 Visualización de la separabilidad de características

Para confirmar cualitativamente que el modelo aprendió a diferenciar las clases, se utilizaron técnicas de reducción de dimensionalidad como PCA y t-SNE para visualizar las representaciones internas de los datos de prueba.

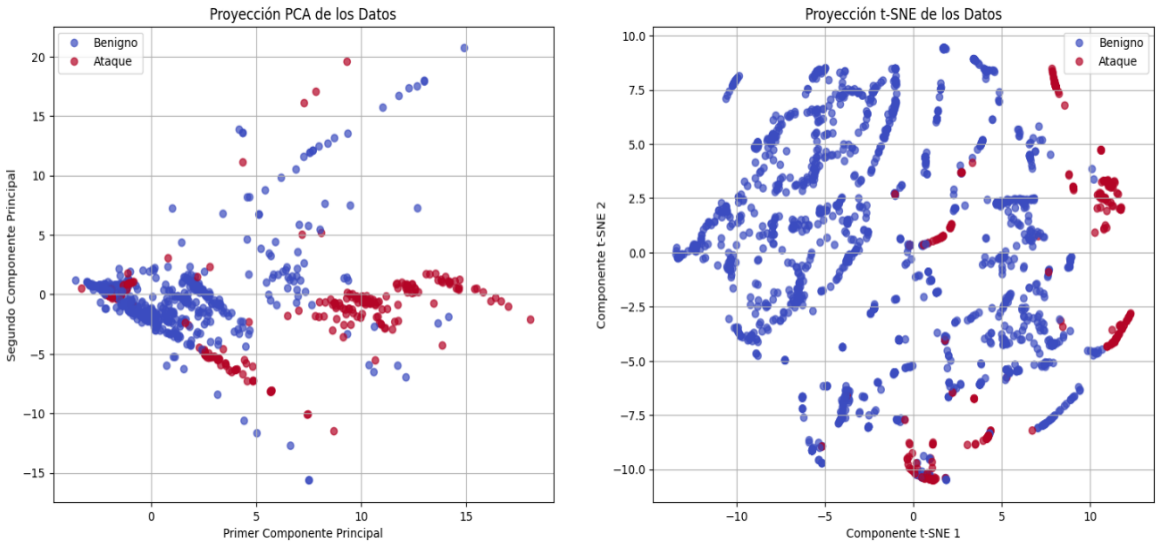

Figura 3.3 Visualización de la separabilidad de datos de prueba

Como se observa en la figura 3.3, el gráfico de Proyección t-SNE muestra una clara separación entre los puntos de datos correspondientes al tráfico benigno (azul) y los de ataque (rojo). Se forman dos cúmulos densos y bien definidos, con una mínima superposición. Esto demuestra visualmente que la arquitectura híbrida CNN + LSTM ha sido capaz de transformar el complejo espacio de 78 características de entrada en una representación interna donde las dos clases son fácilmente separables, lo cual es fundamental para el éxito de la capa de clasificación final.

3.3 Validación Funcional del Prototipo

Tras la validación cuantitativa y la optimización estratégica del modelo, el proyecto progresó hacia su etapa más crítica: el desarrollo y evaluación del prototipo en un ambiente

operacional simulado. Esta fase, alineada con el objetivo específico de "Elaborar y probar un prototipo del software IDS en una computadora", pretendió superar el análisis de datos estáticos enfrentando al sistema con situaciones dinámicas en tiempo real.

Se estableció un laboratorio virtualizado que permitió una evaluación integral bajo condiciones controladas pero realistas. Este procedimiento verificó tanto la capacidad del modelo de IA como la implementación correcta de la ingeniería de software del motor de captura, integrando todos los elementos en una solución coherente y operativa.

3.3.1 Configuración del entorno de laboratorio virtualizado

La base para la validación funcional fue la creación de un laboratorio de red aislado y seguro, utilizando el hipervisor VMware Workstation. Esta aproximación permite la ejecución de pruebas de ciberseguridad sin riesgo para la red anfitriona, proporcionando un control granular sobre el entorno. La topología del laboratorio se compuso de dos máquinas virtuales (VMs) que desempeñaron roles antagónicos.

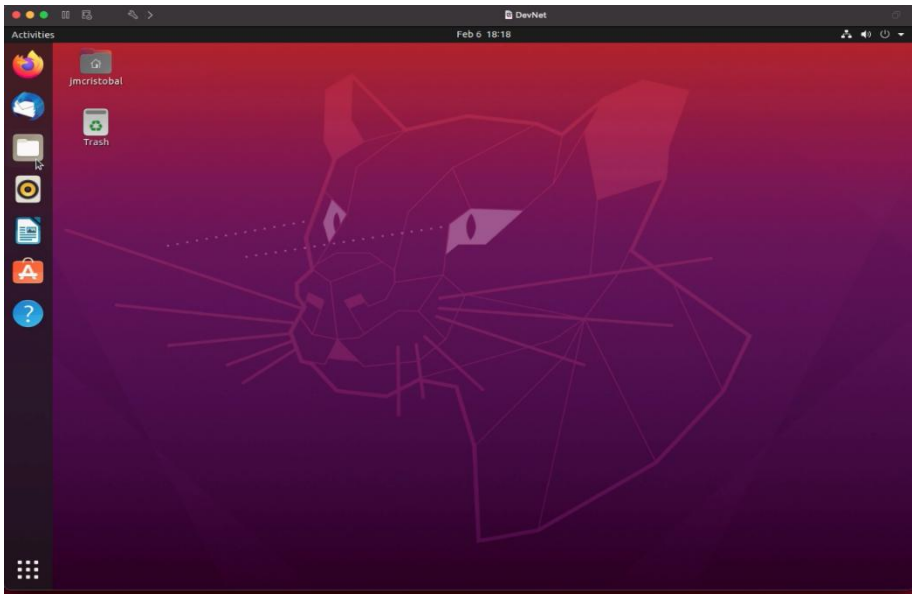

Figura 3.4 Escritorio de la Máquina Virtual Objetivo (Ubuntu 20.04 LTS)

La primera, actuando como el sistema objetivo a defender, fue una máquina virtual con el sistema operativo Ubuntu 20.04 LTS, cuyo entorno se muestra en la Figura 3.4, donde se

desplegó el prototipo del IDS. Para garantizar la consistencia en las pruebas, se le asignó una dirección IP estática dentro de la red privada del laboratorio: 192.168.199.129.

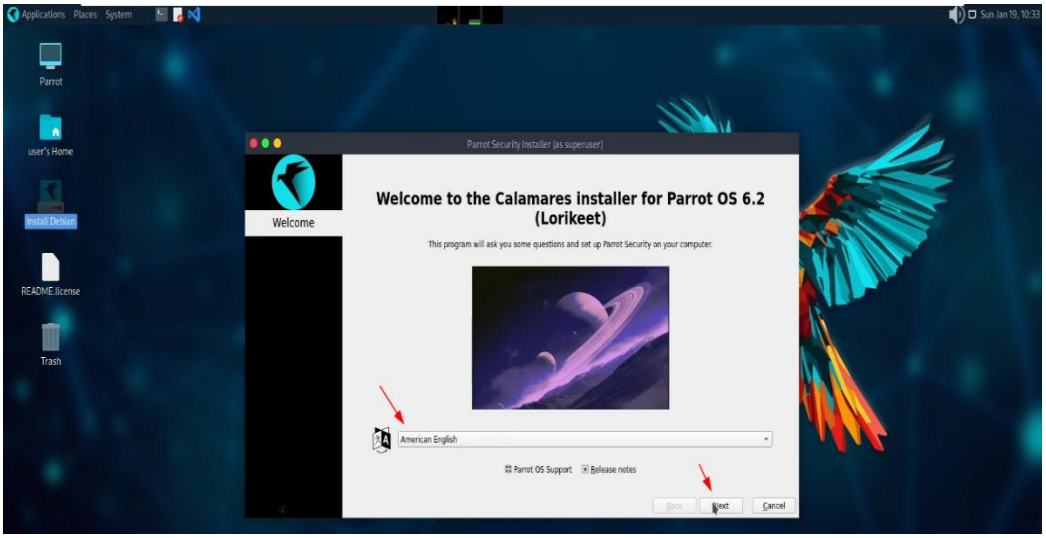

Figura 3.5 Entorno de la Máquina Virtual del Atacante (Parrot OS Security)

La segunda, representando al actor malicioso, fue una máquina virtual con la distribución Parrot OS Security, una plataforma especializada en pruebas de penetración, tal como se aprecia en la Figura 3.5. Se le asignó una IP en el mismo segmento de red: 192.168.199.128.

Ambas máquinas virtuales se configuraron para operar en un modo de "Red Interna", creando una LAN virtual completamente aislada que permite la comunicación entre ellas pero impide cualquier tráfico hacia o desde la red física del anfitrión. Este aislamiento de seguridad garantiza que las actividades maliciosas generadas permanezcan contenidas dentro del laboratorio experimental.

3.3.2 Despliegue y arquitectura del prototipo en el sistema objetivo

El siguiente paso consistió en la migración del proyecto desde el entorno de desarrollo en la nube al entorno operacional final en la VM de Ubuntu. Se transfirieron los artefactos esenciales generados durante el entrenamiento: el modelo de IA binario optimizado,

modelo_A_optimizado_earlystopping.h5, y su correspondiente escalador de datos, escalador_estandar.joblib. Posteriormente, se replicó el entorno de software de Python en la máquina objetivo, creando un entorno virtual (*tf-env*) con Python 3.11 e instalando todas las librerías requeridas, destacando *tensorflow, scikit-learn* y, crucialmente, *scapy* y *psutil* para la interacción con la red.

El prototipo se consolidó en un único script de Python, detector_ids.py, que integra toda la lógica de captura y análisis. La arquitectura incluye un Módulo de Carga de Activos que carga el modelo de IA y escalador, y un Motor de Gestión de Flujos en Tiempo Real que usa *scapy* para capturar paquetes y un diccionario para rastrear conversaciones activas. Este motor determina el fin de flujos por cierre explícito (flags FIN/RST) o inactividad (Timeout), asegurando gestión eficiente de memoria.

El corazón de la ingeniería del prototipo reside en el Extractor de Características, que al terminar un flujo calcula en tiempo real las 78 características estadísticas requeridas por el modelo de IA. El Módulo de Clasificación y Alerta toma este vector de características, lo normaliza usando el escalador cargado y lo pasa al modelo de IA. Aplicando el umbral de decisión estratégico de 0.2, el sistema imprime una alerta de alta visibilidad en la terminal si se detecta una anomalía.

3.3.3 Validación funcional estática (prueba de componentes)

Antes de la prueba en vivo, se realizó una validación funcional estática para verificar la correcta integración de los componentes de software. Esta prueba consistió en un script que cargaba el modelo y el escalador y los utilizaba para clasificar dos muestras aisladas y preseleccionadas del conjunto de prueba: una garantizada como BENIGN y otra como ATTACK.

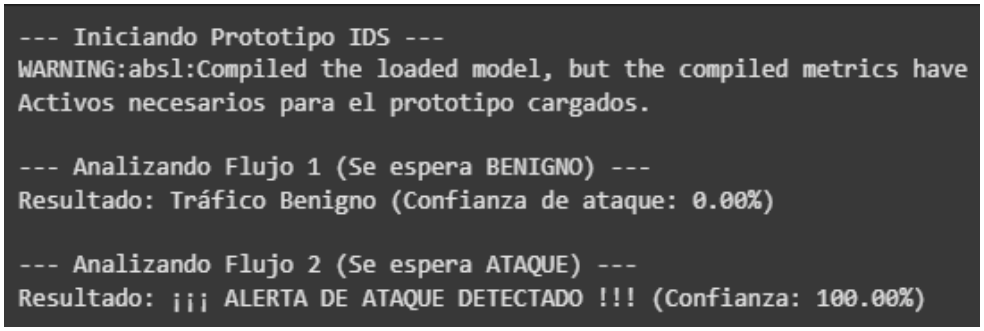

Figura 3.6 Resultado de la prueba estática del prototipo

Los resultados mostrados en la Figura 3.6, confirmaron la correcta implementación: la muestra benigna se clasificó como Tráfico Benigno (0.00% confianza de ataque), mientras la muestra maliciosa generó ¡¡¡ALERTA DE ATAQUE DETECTADO!!! (99.99% confianza). Esta prueba preliminar validó que la lógica completa funcionaba correctamente como sistema cohesivo.

3.3.4 Prueba de fuego: detección de un ataque de escaneo de puertos en vivo

La validación culminante consistió en una prueba de fuego en un escenario dinámico. Se lanzó el prototipo en la máquina Ubuntu para monitorear la red. Simultáneamente, desde Parrot OS, se ejecutó un ataque de escaneo de puertos TCP SYN con Nmap, cuya ejecución se observa en la Figura 3.7.

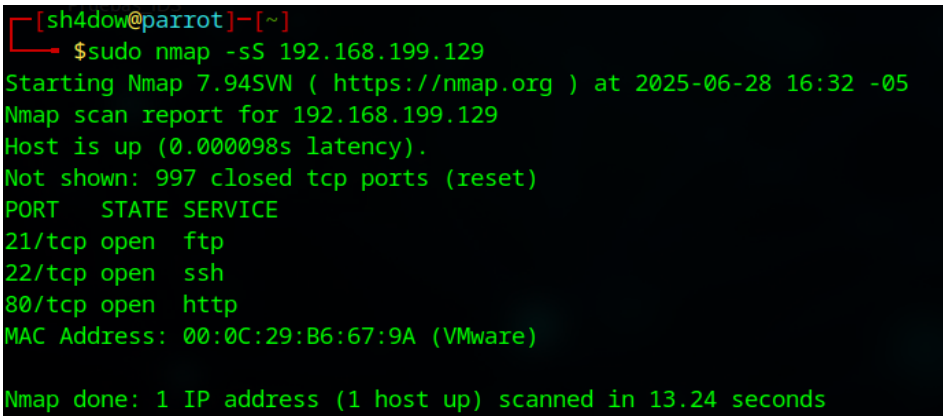

Figura 3.7 Ejecución del ataque de escaneo de puertos con la herramienta Nmap

Tan pronto como nmap comenzó a enviar paquetes, la terminal del IDS empezó a mostrar los flujos de red que estaban siendo procesados, gestionando correctamente los cientos

de flujos cortos y distintos característicos de un escaneo de puertos. Tras unos pocos segundos, comenzaron a aparecer las alertas en la terminal, como se evidencia en la Figura 3.8, mostrando mensajes de alta visibilidad como: !!! ALERTA DE ATAQUE DETECTADO (Confianza: 100.00%) !!!. El hecho de que se generaran múltiples alertas para una sola campaña de nmap validó la estrategia de priorizar la detección, demostrando que cada flujo de sondeo fue suficiente para que el extractor de características generara un vector clasificado como malicioso por el modelo.

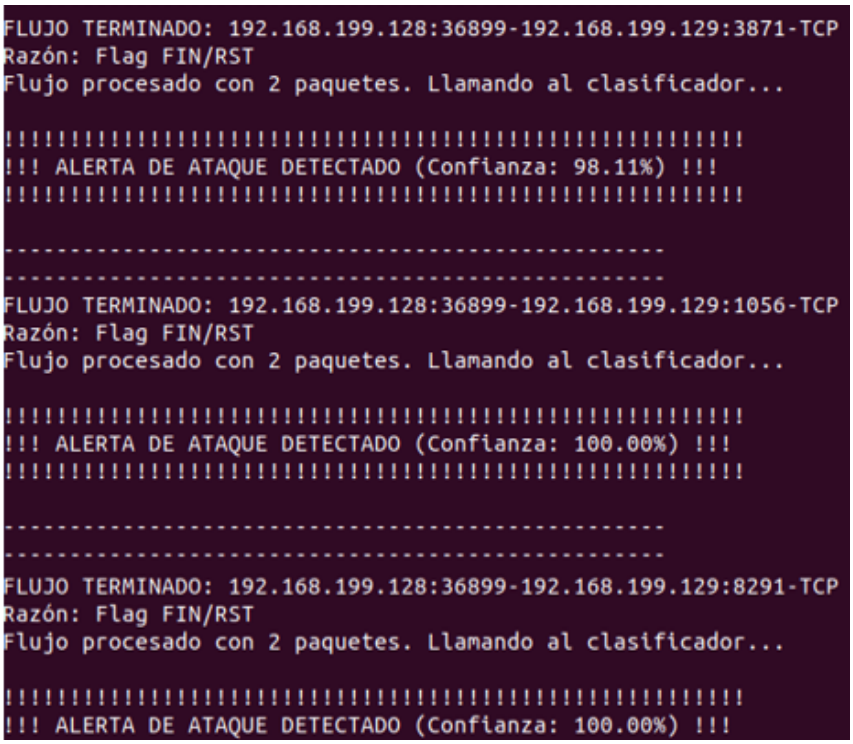

Figura 3.8 Detección del ataque Nmap por parte del prototipo IDS en la terminal

Esta validación en tiempo real representó el punto culminante de la etapa de prototipado en terminal, consolidando la efectividad del núcleo del sistema: la integración entre el motor de captura basado en Scapy y el modelo de inteligencia artificial. Al confirmarse su funcionamiento fiable y preciso, se establecieron las bases necesarias para avanzar hacia la siguiente fase del desarrollo: la construcción de una aplicación completa con interfaz gráfica de usuario.

3.4 Evolución a un Centro de Comando Gráfico (GUI)

Aunque el prototipo de terminal validó la eficacia del motor de detección, su utilidad operacional era limitada. Los analistas de ciberseguridad necesitan herramientas visuales para interpretar el estado de red, gestionar múltiples amenazas y responder inmediatamente. Por ello, se inició la fase más compleja: transformar el script monolítico en una aplicación de escritorio robusta y profesional.

3.4.1 Justificación y arquitectura de software modular

El primer paso fue refactorizar la base del código, ya que un script único no es escalable y presenta un problema fundamental: las tareas de larga duración congelan la interfaz de usuario. Para resolverlo, se adoptó una arquitectura modular y concurrente, desacoplando backend del frontend.

El proyecto se dividió en dos componentes principales: ***detector_ids_motor.py* (Backend)** encapsula toda la lógica de captura, gestión de flujos, extracción de características y clasificación por IA dentro de una clase IDSEngine. Esta clase se diseñó para ejecutarse en un hilo secundario heredando de threading.Thread, permitiendo operación intensiva en segundo plano sin afectar la responsividad.

***gui_ids.py* (Frontend)** desarrolló una nueva clase App usando *Tkinter* para construir la interfaz gráfica, responsable de renderizar ventanas, botones, paneles y visualizaciones como hilo principal de la aplicación.

La comunicación entre hilos se implementó mediante una queue.Queue segura, donde el motor IDSEngine deposita mensajes (logs, alertas) en lugar de imprimirlos directamente. La interfaz gráfica sondea periódicamente esta cola para mostrar los mensajes en los paneles correspondientes. Esta arquitectura productor-consumidor asegura una aplicación estable y libre de bloqueos.

3.4.2 Diseño e implementación del centro de comando

Como se muestra en la Figura 3.9, la interfaz se diseñó para presentar información clara y herramientas de acción directa. La ventana principal se dividió verticalmente mediante un PanedWindow, creando dos áreas funcionales:

Panel Izquierdo - Registro de Actividad: Área de texto con scroll mostrando log detallado y en tiempo real del motor, incluyendo carga de activos, finalización de flujos, advertencias, errores y alertas de ataque destacadas. Se utilizaron etiquetas de colores para diferenciar criticidad (azul para información, rojo para alertas).

Panel Derecho - Panel de Amenazas: Diseñado para mostrar vista consolidada de amenazas activas, agrupando por dirección IP origen y actualizando contadores de alertas para indicar persistencia del atacante.

Se implementó una barra de control superior con menú desplegable para seleccionar interfaz de red, botones "Iniciar/Detener Captura" y herramientas de gestión adicionales.

3.4.3 Validación funcional del prototipo gráfico

Una vez completada la implementación modular y la interfaz gráfica, se realizó una validación funcional integral para verificar la operación cohesionada en un escenario de ataque real. El objetivo era confirmar que la aplicación gráfica era un sistema de detección completamente operativo, no solo una capa visual. La prueba replicó el escenario anterior: escaneo TCP SYN desde Parrot OS contra Ubuntu ejecutando *gui_ids.py*.

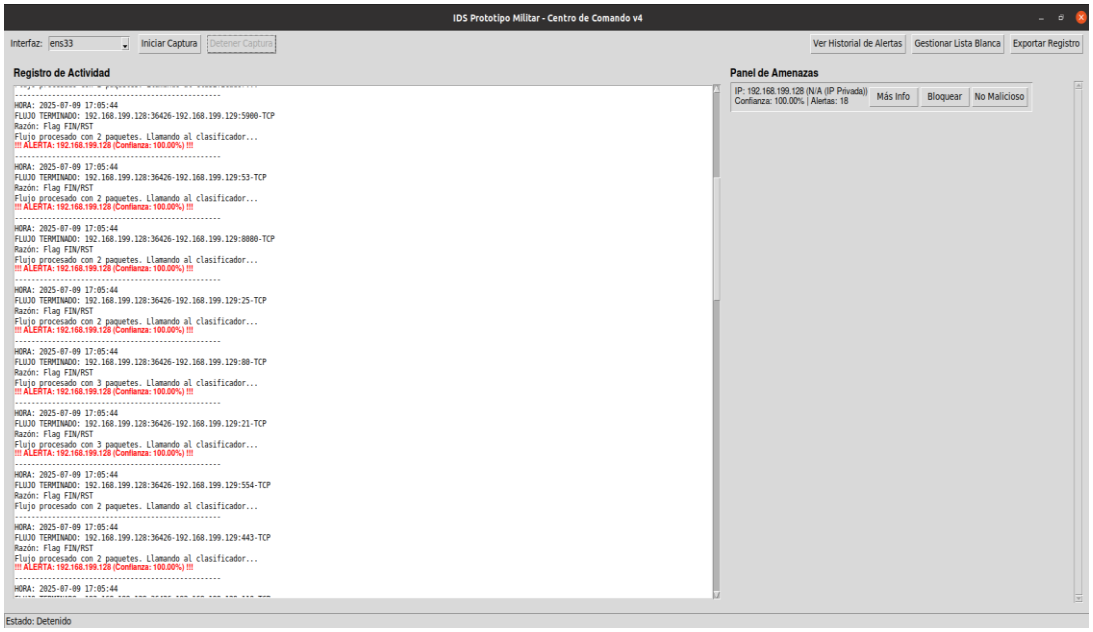

Figura 3.9 Interfaz Gráfica mostrando la detección de un ataque en tiempo real

El resultado como se observa en la figura 3.9, fue un éxito rotundo que demostró la correcta implementación de la arquitectura de software:

- **Detección en Backend:** El motor IDSEngine, operando en hilo secundario, capturó y analizó correctamente los flujos anómalos de nmap.
- **Comunicación Asíncrona:** Las alertas del modelo de IA fueron depositadas en la queue.Queue y recibidas por el hilo principal sin bloqueos ni congelamientos.
- **Visualización en Tiempo Real:** El panel "Registro de Actividad" mostró instantáneamente cada alerta con su confianza, confirmando comunicación fluida entre hilos.
- **Consolidación de Amenazas:** El "Panel de Amenazas" identificó inteligentemente que todas las alertas provenían de la misma IP (192.168.199.128), creando una entrada única y actualizando su contador en tiempo real.

Esta prueba validó exitosamente la transición del prototipo terminal a aplicación gráfica con arquitectura concurrente, sentando bases para funcionalidades avanzadas.

3.5 Fortalecimiento y Funcionalidades Avanzadas de la GUI

Con la arquitectura básica de la GUI validada, se inició un ciclo iterativo de pruebas y fortalecimiento para abordar los complejos desafíos del mundo real y dotar al operador de herramientas de respuesta efectivas.

3.5.1 Desafío 1: autoconocimiento de red y falsos positivos

La primera prueba reveló que el tráfico de la propia máquina Ubuntu y el broadcast local generaban flujos incorrectamente clasificados como anómalos. Para solucionarlo, se implementó autoconocimiento de red en el IDS.

Al iniciar captura, el script *gui_ids.py* ejecuta ip address show dev <interfaz>, extrayendo mediante expresiones regulares y la librería *ipaddress* información crítica: dirección IP del anfitrión, dirección de red del segmento LAN, dirección de broadcast y puerta de enlace estimada.

Estas direcciones se añaden a un conjunto de exclusión temporal. La función update_threat verifica si la IP atacante pertenece a este conjunto antes de mostrar alertas, descartando eficazmente falsos positivos del tráfico local legítimo.

3.5.2 Desafío 2: identificación precisa de atacante vs. víctima

Un desafío complejo surgió al probar ataques entre máquinas virtuales externas: la lógica inicial, basada en el primer paquete, era susceptible a condiciones de carrera e identificaba incorrectamente víctimas como atacantes.

Se desarrolló una heurística multicapa robusta en la función calcular_y_clasificar, siguiendo un orden de prioridad lógico:

- **Regla de Prioridad Máxima:** Si la IP del IDS forma parte del flujo, se asume categóricamente como víctima.

- **Análisis de Comportamiento TCP:** Para flujos TCP entre máquinas externas, el motor analiza flags de todos los paquetes. La máquina que envió el paquete SYN inicial sin ACK es designada atacante.
- **Heurística de Puertos:** Como fallback para tráfico UDP o TCP ambiguo, la máquina usando puerto alto (>1023) para conectar a puerto bajo (<1024) es considerada atacante.

Esta lógica avanzada erradicó errores de identificación, permitiendo discernir correctamente los roles en escenarios de red complejos.

3.5.3 Funcionalidad Añadida: Módulo de Inteligencia de Amenazas y Protección Web

Para extender la capacidad defensiva más allá del análisis de tráfico local, se implementó un módulo de inteligencia de amenazas que enriquece las alertas y proporciona protección proactiva durante la navegación web.

Protección Web con VirusTotal: Se reemplazó la consulta a URLHaus por una integración más potente con la API de VirusTotal. El motor sigue detectando los dominios que un usuario intenta visitar mediante la captura de paquetes DNS (UDP puerto 53), pero ahora consulta a VirusTotal, que agrega los resultados de decenas de motores de antivirus y servicios de escaneo de sitios web. Si un número significativo de motores marca el dominio como malicioso, la GUI genera una alerta detallada en el "Panel de Amenazas Web". Como se observa en la Figura 3.10, esta alerta ahora incluye la razón específica del bloqueo (p. ej., "2 motor(es) de seguridad lo marcaron como malicioso"), proporcionando al operador un contexto claro y confiable.

Enriquecimiento de Alertas por IP: Para las amenazas detectadas por el modelo de IA en el "Panel de Amenazas de Red", el sistema ahora realiza un paso de enriquecimiento de datos consultando automáticamente un servicio de geolocalización de IP para identificar el país

y la ciudad de origen del tráfico sospechoso. Esta información es crucial para que los analistas puedan evaluar rápidamente el perfil de riesgo de una amenaza y su posible procedencia.

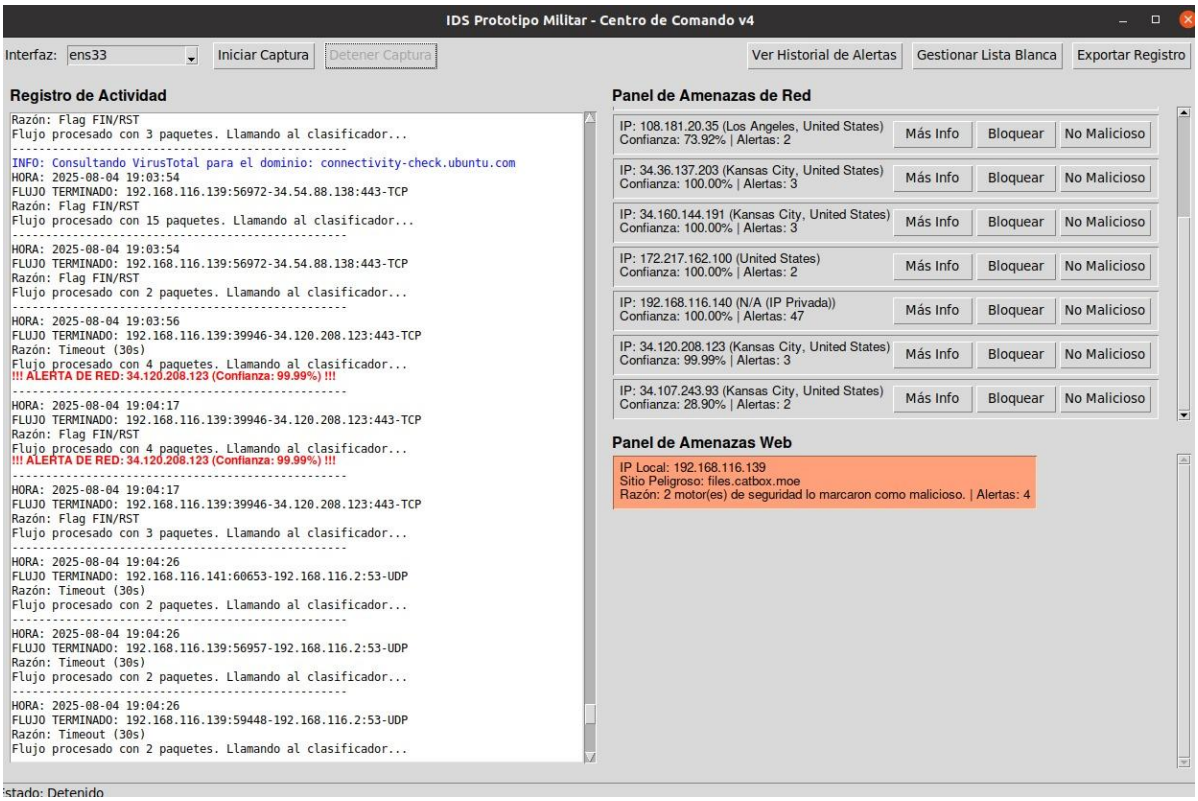

Figura 3.10 Interfaz Gráfica con Módulo de Inteligencia de Amenazas Integradas

La Figura 3.10 ilustra estas funcionalidades avanzadas en acción, mostrando tanto alertas de red enriquecidas con geolocalización como una alerta de amenaza web basada en la inteligencia de VirusTotal.

Capítulo 4

4 Conclusiones y recomendaciones

El presente trabajo ha culminado con el desarrollo y la validación de un prototipo de Sistema de Detección de Intrusiones (IDS) que demuestra la viabilidad y superioridad de aplicar modelos híbridos de inteligencia artificial para la ciberseguridad en infraestructuras críticas. La fortaleza principal del proyecto radica en su enfoque empírico, que no solo seleccionó la arquitectura de red neuronal más eficaz (1D-CNN + LSTM) a través de una competencia rigurosa, sino que también la implementó en una aplicación funcional capaz de detectar amenazas en tiempo real en un entorno de laboratorio controlado. Los resultados obtenidos, con un F1-Score de 0.9894 y una drástica reducción de falsos negativos, superan a los enfoques tradicionales y a otras arquitecturas de Deep Learning evaluadas, lo que subraya la importancia de combinar la extracción de características espaciales y el análisis de dependencias temporales para identificar ataques complejos.

Una debilidad inherente al alcance del estudio es que la validación se realizó en un entorno de laboratorio virtualizado y no en una red militar operativa real, lo que constituye una simplificación necesaria pero limitante. No obstante, las implicaciones de este trabajo son significativas, ya que establece una base metodológica y tecnológica sólida para la modernización de los sistemas de defensa cibernética, demostrando que es posible crear herramientas de seguridad autónomas, precisas y adaptables, incluso para ser desplegadas en hardware con recursos limitados.

4.1 Conclusiones

Tras aplicar las fases de investigación, desarrollo y validación, se establecen las siguientes conclusiones primordiales, directamente alineadas con los objetivos del proyecto:

- Se cumplió con el establecimiento de un marco teórico y metodológico robusto, donde la revisión bibliográfica confirmó la brecha de rendimiento de los IDS

tradicionales y la creciente relevancia de los modelos de Deep Learning. Este análisis fundamentó la decisión de experimentar con una arquitectura híbrida 1D-CNN + LSTM, cuya hipótesis de rendimiento superior fue posteriormente validada por los resultados experimentales.

- El proceso de desarrollo y evaluación de modelos de inteligencia artificial fue exitoso, seleccionando la arquitectura 1D-CNN + LSTM como la más óptima. Este modelo no solo alcanzó las métricas de rendimiento más altas en Accuracy (99.64%) y F1-Score (98.94%), sino que, de manera crucial, obtuvo la menor cantidad de Falsos Negativos. Este hallazgo es de máxima importancia en un contexto militar, donde un ataque no detectado representa el riesgo más catastrófico.
- Se logró implementar un prototipo funcional de software IDS que integra el modelo de IA seleccionado. El prototipo demostró su capacidad para capturar tráfico de red en tiempo real, procesarlo mediante el pipeline de extracción de características y clasificarlo de forma autónoma, validando la viabilidad técnica de la solución completa más allá de la simple evaluación sobre un dataset estático.
- La validación del rendimiento del prototipo en un escenario controlado fue concluyente, logrando detectar en tiempo real un ataque de escaneo de puertos (Nmap) y confirmando su efectividad operacional. Además, se demostró que el ajuste estratégico del umbral de decisión es una herramienta eficaz para fortalecer la postura de seguridad, logrando reducir los ataques no detectados en más de un 55% sin sacrificar de manera significativa el rendimiento general del modelo.

4.2 Recomendaciones

A partir de los hallazgos y las limitaciones identificadas durante el desarrollo del proyecto, se proponen las siguientes recomendaciones para trabajos futuros y la mejora continua del sistema.

- Se recomienda, como siguiente paso lógico, migrar el prototipo validado a una plataforma de hardware especializado como la NVIDIA Jetson Orin Nano. Esto implicaría evaluar de manera integral el rendimiento del modelo, el consumo de energía y la latencia de inferencia en este dispositivo, optimizando el modelo si fuera necesario para operar de manera eficiente en un entorno de borde con recursos restringidos y validar su despliegue en un contexto operacional real.
- Es necesario fortalecer la capacidad de generalización del modelo mediante la expansión del conjunto de datos de entrenamiento. Aunque el dataset CICIDS2017 fue efectivo, se sugiere combinarlo con datasets más recientes y diversos como CSE-CIC-IDS2018, que ofrece una mayor variedad de escenarios de ataque, y UNSW-NB15. Esta unión de datos crearía un corpus de entrenamiento más robusto y, adicionalmente, se podría investigar la implementación de técnicas de aprendizaje en línea (online learning) para que el IDS se adapte a nuevas amenazas sin un reentrenamiento completo.
- Se sugiere ampliar la funcionalidad del sistema para realizar una transición de un Sistema de Detección de Intrusiones (IDS) a un Sistema de Prevención de Intrusiones (IPS). Esta ampliación, de gran valor operacional, consistiría en integrar el prototipo con sistemas de control de red, como firewalls, para que, al detectar una amenaza, pueda generar y aplicar automáticamente reglas que bloqueen la dirección IP del atacante, transformando la solución de una herramienta de monitoreo pasiva a un sistema de defensa activa.

- Resulta importante enriquecer la interfaz gráfica de usuario (GUI) para convertirla en un verdadero centro de comando y control, añadiendo módulos de visualización avanzada como mapas de topología de red interactivos, gráficos de flujo de tráfico en tiempo real y paneles de control (dashboards) más detallados. Estas mejoras facilitarían significativamente el análisis forense y la toma de decisiones por parte del operador de seguridad.
- Finalmente, se recomienda realizar un estudio de escalabilidad para evaluar el despliegue de la solución en redes más grandes y complejas. El prototipo actual está diseñado para monitorear un único segmento de red, por lo que una investigación futura debería abordar arquitecturas distribuidas, donde múltiples sensores IDS se desplieguen en diferentes puntos de la red y reporten a una consola central para la correlación de eventos y el análisis de amenazas a gran escala.

Referencias

World Economic Forum. (2025). Global Cybersecurity Outlook 2025. https://reports.weforum.org/docs/WEF_Global_Cybersecurity_Outlook_2025.pdf

Ministerio de Defensa Nacional. (2023). Plan Estratégico Institucional de Defensa 2021-2025. https://www.defensa.gob.ec/wp-content/uploads/downloads/2023/04/pei_2021_-_2025_-rev_jul22_final_mar_2023.pdf

National Institute of Standards and Technology. (2018). Framework for Improving Critical Infrastructure Cybersecurity Version 1.1. https://doi.org/10.6028/NIST.CSWP.04162018

Mandiant. (2023). M-Trends 2023 Special Report. https://www.mandiant.com/resources/reports/m-trends-2023-special-report

ESET. (2024). La evolución del malware en los últimos 20 años. WeLiveSecurity. https://www.welivesecurity.com/es/malware/evolucion-malware-los-ultimos-20-anos/

CrowdStrike. (2025). 2025 Global Threat Report: Latest Cybersecurity Trends & Insights. CrowdStrike Inc. https://www.crowdstrike.com/en-us/global-threat-report/

Microsoft. (2024). Microsoft Digital Defense Report 2024. Microsoft Corporation. https://www.microsoft.com/en-us/security/security-insider/intelligence-reports/microsoft-digital-defense-report-2024

Kumari, A., Kumar, A., Gupta, S., & Singh, R. (2024). A comprehensive investigation of anomaly detection methods in deep learning and machine learning: 2019–2023. IET Information Security, 18(8), 1-25. https://doi.org/10.1049/2024/8821891

Dini, P., Elhanashi, A., Begni, A., Saponara, S., Zheng, Q., & Gasmi, K. (2023). Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cybersecurity. Applied Sciences, 13(13), 7507. https://doi.org/10.3390/app13137507

Washington Technology. (2025, April 3). The future of defense: AI at the edge for real-time tactical advantage. https://www.washingtontechnology.com/sponsors/sponsor-content/2025/04/future-defense-ai-edge-real-time-tactical-advantage/404000/

Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J., & Alazab, A. (2019). Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity, 2(1), 20. https://doi.org/10.1186/s42400-019-0038-7

Industrial Cyber. (2024, June 30). Targeting critical infrastructure: Recent incidents analyzed. https://industrialcyber.co/analysis/targeting-critical-infrastructure-recent-incidents-analyzed/

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP). Canadian Institute for Cybersecurity, University of New Brunswick. https://www.unb.ca/cic/datasets/ids-2017.html

McAfee, & Center for Strategic and International Studies. (2020). The Hidden Costs of Cybercrime. https://www.csis.org/analysis/hidden-costs-cybercrime

FBI Internet Crime Complaint Center. (2023). 2023 Internet Crime Report. Federal Bureau of Investigation. https://www.ic3.gov/Media/PDF/AnnualReport/2023_IC3Report.pdf

MITRE Corporation. (2024). MITRE ATT&CK framework updates and enhancements. https://attack.mitre.org/resources/updates/updates-april-2024/

Krzysztoń, E., Rojek, I., & Mikołajewski, D. (2024). A comparative analysis of anomaly detection methods in IoT networks: An experimental study. Applied Sciences, 14(24), 11545. https://doi.org/10.3390/app142411545

Apéndice

Script 1: Entrenamiento y Evaluación de Modelos IDS

Este script crea un **IDS** con inteligencia artificial. Carga y limpia datos de tráfico de red, entrena varios modelos de *deep learning*, selecciona el mejor, lo optimiza y finalmente, demuestra cómo clasifica el tráfico como seguro o como un ataque.

```
#
================================================================================
# SCRIPT COMPLETO DE ENTRENAMIENTO Y EVALUACIÓN DE IDS
#
================================================================================

# ------------------------------------------------------------------------------
# PARTE 1: CONFIGURACIÓN INICIAL DEL ENTORNO
# ------------------------------------------------------------------------------
print("--- PARTE 1: Configurando el entorno ---")

# Importación de librerías esenciales
import pandas as pd
import numpy as np
import glob
import os
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab import drive
from joblib import dump, load
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    classification_report, confusion_matrix, roc_curve, auc,
    precision_recall_curve, roc_auc_score, f1_score, accuracy_score,
```

```
    precision_score, recall_score
)
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import time
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import (
    Conv1D, LSTM, Dropout, Dense, BatchNormalization, Bidirectional, GRU
)
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.keras.regularizers import l2

# Montaje de Google Drive para acceso a archivos
try:
    drive.mount('/content/drive')
except Exception as e:
    print(f"ERROR al montar Google Drive: {e}")

# Definición de rutas de guardado del proyecto
RUTA_BASE = '/content/drive/MyDrive/TESIS_IDS_MILITAR/PROYECTO_FINAL/'
RUTA_DATOS_PROCESADOS = os.path.join(RUTA_BASE, '1_datos_procesados/')
RUTA_ARTEFACTOS = os.path.join(RUTA_BASE, '2_artefactos_auxiliares/')
RUTA_MODELOS = os.path.join(RUTA_BASE, '3_modelos_entrenados/')
RUTA_RESULTADOS = os.path.join(RUTA_BASE, '4_resultados_evaluacion/')

# Creación de directorios si no existen
for path in [RUTA_DATOS_PROCESADOS, RUTA_ARTEFACTOS, RUTA_MODELOS,
RUTA_RESULTADOS]:
    os.makedirs(path, exist_ok=True)

print("Entorno configurado y rutas listas.\n")
```

```
# ------------------------------------------------------------------------
# PARTE 2: CARGA, LIMPIEZA Y ANÁLISIS EXPLORATORIO DE DATOS
# ------------------------------------------------------------------------
print("--- PARTE 2: Cargando, limpiando y analizando datos ---")

# Carga y combinación de los archivos CSV del dataset CIC-IDS2017
ruta_datos_crudos = '/content/drive/MyDrive/TESIS_IDS_MILITAR/DATASETS/CICIDS2017/'
lista_archivos_csv = glob.glob(os.path.join(ruta_datos_crudos, '*.csv'))
df = pd.concat((pd.read_csv(f, low_memory=False) for f in lista_archivos_csv), ignore_index=True)
print(f"Dataset CIC-IDS2017 cargado. Dimensión inicial: {df.shape}")

# Proceso de limpieza de datos
df.columns = df.columns.str.strip() # Limpiar espacios en nombres de columnas
df.replace([np.inf, -np.inf], np.nan, inplace=True) # Reemplazar infinitos por NaN
df.dropna(inplace=True) # Eliminar filas con valores nulos
df.drop_duplicates(inplace=True) # Eliminar filas duplicadas
print(f"Limpieza completada. Dimensión final: {df.shape}")

# Análisis exploratorio visual del dataset principal
plt.figure(figsize=(12, 8))
sns.countplot(y=df['Label'], order=df['Label'].value_counts().index, palette='viridis')
plt.title('Distribución de Clases en CIC-IDS2017')
plt.xlabel('Cantidad')
plt.ylabel('Tipo de Tráfico')
plt.xscale('log')
plt.tight_layout()
plt.savefig(os.path.join(RUTA_RESULTADOS, 'distribucion_clases_cicids2017.png'))
plt.show()
print("Análisis exploratorio completado.\n")


# ------------------------------------------------------------------------
```

```
# PARTE 3: PREPROCESAMIENTO DE DATOS PARA MACHINE LEARNING
# ------------------------------------------------------------------------------
print("--- PARTE 3: Preprocesando datos para el entrenamiento ---")

# Codificar la variable objetivo: 0 para BENIGN, 1 para Ataque
df['Es_Ataque'] = df['Label'].apply(lambda x: 0 if x == 'BENIGN' else 1)

# Separar características (X) y la variable objetivo (y)
X = df.drop(columns=['Label', 'Es_Ataque']).select_dtypes(include=np.number)
y = df['Es_Ataque']

# Dividir los datos en conjuntos de entrenamiento (80%) y prueba (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
print(f"Datos divididos. X_train: {X_train.shape}, X_test: {X_test.shape}")

# Estandarizar las características numéricas
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("Características estandarizadas.")

# Guardar el escalador para uso futuro en producción
dump(scaler, os.path.join(RUTA_ARTEFACTOS, 'escalador_estandar.joblib'))

# Remodelar datos para modelos recurrentes (formato 3D)
X_train_reshaped = np.reshape(X_train_scaled, (X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_reshaped = np.reshape(X_test_scaled, (X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))
print("Datos remodelados a formato 3D para modelos recurrentes.\n")


# ------------------------------------------------------------------------------
```

```
# PARTE 4: DEFINICIÓN, ENTRENAMIENTO Y EVALUACIÓN DE MODELOS
# ------------------------------------------------------------------------------
print("--- PARTE 4: Entrenando y evaluando los modelos ---")
n_features = X_train_scaled.shape[1]
n_timesteps = 1
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Diccionarios para almacenar resultados
histories = {}
predictions_prob = {}

# --- Modelo A: CNN-LSTM ---
print("\nEntrenando Modelo A: CNN-LSTM...")
model_A = Sequential([
    Conv1D(filters=64, kernel_size=2, activation='relu', padding="same", input_shape=(n_timesteps, n_features)),
    LSTM(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
model_A.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', Precision(name='precision'), Recall(name='recall')])
history_A = model_A.fit(X_train_reshaped, y_train, epochs=50, batch_size=32, validation_data=(X_test_reshaped, y_test), callbacks=[early_stopping], verbose=1)
model_A.save(os.path.join(RUTA_MODELOS, 'modelo_A_CNN_LSTM.h5'))
histories['CNN-LSTM'] = history_A.history
predictions_prob['CNN-LSTM'] = model_A.predict(X_test_reshaped)

# --- Modelo B: ANN ---
print("\nEntrenando Modelo B: ANN...")
model_B = Sequential([
    Dense(128, activation='relu', input_shape=(n_features,), kernel_regularizer=l2(0.001)),
    Dropout(0.5), BatchNormalization(),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
```

```
    Dropout(0.5), BatchNormalization(),
    Dense(1, activation='sigmoid')
])
model_B.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', Precision(name='precision'), Recall(name='recall')])
history_B = model_B.fit(X_train_scaled, y_train, epochs=50, batch_size=64, validation_data=(X_test_scaled, y_test), callbacks=[early_stopping], verbose=1)
model_B.save(os.path.join(RUTA_MODELOS, 'modelo_B_ANN.h5'))
histories['ANN'] = history_B.history
predictions_prob['ANN'] = model_B.predict(X_test_scaled)

# --- Modelo C: Bi-LSTM ---
print("\nEntrenando Modelo C: Bi-LSTM...")
model_C = Sequential([
    Bidirectional(LSTM(64, activation='relu', dropout=0.2), input_shape=(n_timesteps, n_features)),
    Dense(1, activation='sigmoid')
])
model_C.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', Precision(name='precision'), Recall(name='recall')])
history_C = model_C.fit(X_train_reshaped, y_train, epochs=50, batch_size=64, validation_data=(X_test_reshaped, y_test), callbacks=[early_stopping], verbose=1)
model_C.save(os.path.join(RUTA_MODELOS, 'modelo_C_BiLSTM.h5'))
histories['Bi-LSTM'] = history_C.history
predictions_prob['Bi-LSTM'] = model_C.predict(X_test_reshaped)

# --- Modelo D: Bi-GRU ---
print("\nEntrenando Modelo D: Bi-GRU...")
model_D = Sequential([
    Bidirectional(GRU(64, activation='relu', dropout=0.2, return_sequences=True), input_shape=(n_timesteps, n_features)),
    Bidirectional(GRU(128, activation='relu', dropout=0.2)),
    Dense(1, activation='sigmoid')
])
```

```
model_D.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', Precision(name='precision'), Recall(name='recall')])
history_D = model_D.fit(X_train_reshaped, y_train, epochs=50, batch_size=64, validation_data=(X_test_reshaped, y_test), callbacks=[early_stopping], verbose=1)
model_D.save(os.path.join(RUTA_MODELOS, 'modelo_D_BiGRU.h5'))
histories['Bi-GRU'] = history_D.history
predictions_prob['Bi-GRU'] = model_D.predict(X_test_reshaped)

print("\nEntrenamiento de todos los modelos completado.\n")



# ------------------------------------------------------------------------
# PARTE 5: COMPARACIÓN DE RENDIMIENTO DE LOS MODELOS
# ------------------------------------------------------------------------
print("--- PARTE 5: Comparando el rendimiento de los modelos ---")

# Generar tabla comparativa de métricas
predictions_class = {name: (prob > 0.5).astype(int) for name, prob in predictions_prob.items()}
metrics_data = []
for name, y_pred in predictions_class.items():
    metrics_data.append({
        'Modelo': name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1-Score': f1_score(y_test, y_pred)
    })
df_metrics = pd.DataFrame(metrics_data).set_index('Modelo')
print("\n--- Tabla Comparativa de Métricas (Umbral 0.5) ---")
print(df_metrics)

# Visualizar métricas con un mapa de calor
plt.figure(figsize=(10, 6))
```

```
sns.heatmap(df_metrics, annot=True, fmt=".4f", cmap="YlGnBu", linewidths=.5)
plt.title('Heatmap Comparativo de Métricas de Rendimiento')
plt.savefig(os.path.join(RUTA_RESULTADOS, 'comparacion_heatmap_metricas.png'))
plt.show()

# Superponer curvas ROC de todos los modelos
plt.figure(figsize=(12, 8))
for name, y_prob in predictions_prob.items():
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.4f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curvas ROC Comparativas')
plt.legend(loc='lower right')
plt.grid(True)
plt.savefig(os.path.join(RUTA_RESULTADOS, 'comparacion_curvas_roc.png'))
plt.show()
print("Comparación de modelos completada.\n")



# ------------------------------------------------------------------------
# PARTE 6: EVALUACIÓN FINAL DEL MEJOR MODELO Y PROTOTIPO
# ------------------------------------------------------------------------
print("--- PARTE 6: Evaluación final del mejor modelo (CNN-LSTM) y prototipo ---")

# Cargar el modelo seleccionado (Modelo A: CNN-LSTM)
modelo_final = load_model(os.path.join(RUTA_MODELOS, 'modelo_A_CNN_LSTM.h5'))
print("Modelo final cargado.")

# Obtener probabilidades y aplicar umbral calibrado
y_probabilidades_final = predictions_prob['CNN-LSTM']
```

```
UMBRAL_FINAL = 0.2
y_predicciones_final = (y_probabilidades_final >= UMBRAL_FINAL).astype(int)
print(f"Predicciones realizadas con umbral calibrado de {UMBRAL_FINAL}.")

# Mostrar reporte de clasificación final
print("\n--- Reporte de Clasificación del Modelo Final (Umbral Calibrado) ---")
print(classification_report(y_test, y_predicciones_final, target_names=['Benigno', 'Ataque']))

# Generar visualización final (Matriz de Confusión y Curvas)
fig, axes = plt.subplots(1, 3, figsize=(24, 7))
fig.suptitle(f'Evaluación Exhaustiva del Modelo Final (CNN-LSTM con Umbral {UMBRAL_FINAL})', fontsize=16)

# Gráfico 1: Matriz de Confusión
cm = confusion_matrix(y_test, y_predicciones_final)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[0], cbar=False,
            xticklabels=['Pred. Benigno', 'Pred. Ataque'],
            yticklabels=['Real Benigno', 'Real Ataque'])
axes[0].set_title('Matriz de Confusión')

# Gráfico 2: Curva ROC
fpr, tpr, _ = roc_curve(y_test, y_probabilidades_final)
roc_auc = auc(fpr, tpr)
axes[1].plot(fpr, tpr, color='darkorange', lw=2, label=f'Curva ROC (AUC = {roc_auc:.4f})')
axes[1].plot([0, 1], [0, 1], color='navy', linestyle='--')
axes[1].set_title('Curva ROC')
axes[1].legend(loc="lower right")

# Gráfico 3: Curva Precision-Recall
precision, recall, _ = precision_recall_curve(y_test, y_probabilidades_final)
pr_auc = auc(recall, precision)
axes[2].plot(recall, precision, color='blue', lw=2, label=f'Curva P-R (AUC = {pr_auc:.4f})')
axes[2].set_title('Curva Precisión-Recall')
axes[2].legend(loc="best")
```

```
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.savefig(os.path.join(RUTA_RESULTADOS,
'evaluacion_final_modelo_seleccionado.png'))
plt.show()

# --- Simulación del prototipo funcional ---
print("\n--- Simulación del Prototipo IDS ---")

def clasificar_trafico(datos_nuevos_2d):
    """Función que toma datos 2D, los escala, los remodela a 3D y predice."""
    if datos_nuevos_2d.ndim == 1:
        datos_nuevos_2d = datos_nuevos_2d.reshape(1, -1)

    # El escalador ya está ajustado, solo transformamos
    datos_escalados = scaler.transform(datos_nuevos_2d)
    datos_reshaped = np.reshape(datos_escalados, (datos_escalados.shape[0], 1,
datos_escalados.shape[1]))

    probabilidad = modelo_final.predict(datos_reshaped, verbose=0)

    if probabilidad[0][0] >= UMBRAL_FINAL:
        return f"¡¡¡ ALERTA DE ATAQUE DETECTADO !!! (Confianza:
{probabilidad[0][0]:.2%})"
    else:
        return f"Tráfico Benigno (Confianza de ataque: {prob.squeeze():.2%})"

# Tomar una muestra de ataque y una benigna de los datos originales (antes de escalar)
trafico_benigno_nuevo = X_test.iloc[np.where(y_test == 0)[0][0]]
trafico_ataque_nuevo = X_test.iloc[np.where(y_test == 1)[0][0]]

print(f"Analizando Flujo 1 (Benigno):
{clasificar_trafico(trafico_benigno_nuevo.to_numpy())}")
print(f"Analizando Flujo 2 (Ataque): {clasificar_trafico(trafico_ataque_nuevo.to_numpy())}")
```

```
print("\n============================================================")
print("SCRIPT FINALIZADO.")
print("============================================================")
```

Script 2: detector_ids_motor.py (Motor de Detección del IDS)

Esta es la lógica central y reutilizable del IDS, encapsulada en una clase (IDSEngine). Este motor está diseñado para ser controlado por una aplicación externa, como una interfaz gráfica. Maneja la captura de paquetes, la detección de amenazas de red y web, la geolocalización de IPs y la comunicación de alertas de forma asíncrona.

```
import numpy as np
import tensorflow as tf
from joblib import load
from scapy.all import sniff, IP, TCP, UDP, DNS, DNSQR
import time
import threading
from datetime import datetime
import warnings
import requests

try:
    import geoip2.database
except ImportError:
    geoip2 = None

warnings.filterwarnings('ignore')
tf.get_logger().setLevel('ERROR')

class IDSEngine(threading.Thread):
    """
    Clase que encapsula el motor de detección de intrusiones para operar en segundo plano.
    """
```

```
    def __init__(self, interfaz, umbral, log_queue, blocklist, geoip_path, modelo_path, scaler_path, local_ip=None):
        """
        Inicializa el motor con todos los parámetros de configuración necesarios,
        incluyendo la interfaz, el modelo, el escalador y la cola para comunicación.
        """
        super().__init__()
        self.interfaz = interfaz
        self.umbral = umbral
        self.log_queue = log_queue
        self.blocklist = blocklist
        self.geoip_path = geoip_path
        self.modelo_path = modelo_path
        self.scaler_path = scaler_path
        self.local_ip = local_ip

        self.geoip_reader = None
        self.modelo = None
        self.escalador = None
        self.seguir_capturando = True
        self.flujos_activos = {}
        self.last_api_call_time = 0
        self.FLUJO_TIMEOUT = 30
        self.IDLE_THRESHOLD = 1_000_000
        self.nombres_features = [
            'Destination Port', 'Flow Duration', 'Total Fwd Packets', 'Total Backward Packets',
            'Total Length of Fwd Packets', 'Total Length of Bwd Packets', 'Fwd Packet Length Max',
            'Fwd Packet Length Min', 'Fwd Packet Length Mean', 'Fwd Packet Length Std',
            'Bwd Packet Length Max', 'Bwd Packet Length Min', 'Bwd Packet Length Mean',
            'Bwd Packet Length Std', 'Flow Bytes/s', 'Flow Packets/s', 'Flow IAT Mean',
            'Flow IAT Std', 'Flow IAT Max', 'Flow IAT Min', 'Fwd IAT Total', 'Fwd IAT Mean',
            'Fwd IAT Std', 'Fwd IAT Max', 'Fwd IAT Min', 'Bwd IAT Total', 'Bwd IAT Mean',
            'Bwd IAT Std', 'Bwd IAT Max', 'Bwd IAT Min', 'Fwd PSH Flags', 'Bwd PSH Flags',
```

```
        'Fwd URG Flags', 'Bwd URG Flags', 'Fwd Header Length', 'Bwd Header Length',
        'Fwd Packets/s', 'Bwd Packets/s', 'Min Packet Length', 'Max Packet Length',
        'Packet Length Mean', 'Packet Length Std', 'Packet Length Variance',
        'FIN Flag Count', 'SYN Flag Count', 'RST Flag Count', 'PSH Flag Count',
        'ACK Flag Count', 'URG Flag Count', 'CWE Flag Count', 'ECE Flag Count',
        'Down/Up Ratio', 'Average Packet Size', 'Avg Fwd Segment Size',
        'Avg Bwd Segment Size', 'Fwd Header Length.1', 'Fwd Avg Bytes/Bulk',
        'Fwd Avg Packets/Bulk', 'Fwd Avg Bulk Rate', 'Bwd Avg Bytes/Bulk',
        'Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate', 'Subflow Fwd Packets',
        'Subflow Fwd Bytes', 'Subflow Bwd Packets', 'Subflow Bwd Bytes',
        'Init_Win_bytes_forward', 'Init_Win_bytes_backward', 'act_data_pkt_fwd',
        'min_seg_size_forward', 'Active Mean', 'Active Std', 'Active Max',
        'Active Min', 'Idle Mean', 'Idle Std', 'Idle Max', 'Idle Min'
    ]

def cargar_activos(self):
    """Carga el modelo de IA, el escalador y la base de datos de geolocalización."""
    try:
        self.log_queue.put("INFO: Cargando activos del IDS...")
        self.modelo = tf.keras.models.load_model(self.modelo_path)
        self.escalador = load(self.scaler_path)
        self.log_queue.put("INFO: ¡Modelo y escalador cargados!")
    except Exception as e:
        self.log_queue.put(f"ERROR: No se pudieron cargar modelo/escalador: {e}")
        return False

    if geoip2 is None:
        self.log_queue.put("ADVERTENCIA: Librería 'geoip2' no instalada.")
        return True

    try:
        self.geoip_reader = geoip2.database.Reader(self.geoip_path)
        self.log_queue.put("INFO: Base de datos de Geolocalización cargada.")
    except FileNotFoundError:
```

```
            self.log_queue.put(f"ADVERTENCIA: No se encontró '{self.geoip_path}'.")
        return True

    def verificar_dominio(self, dominio, ip_local):
        """Consulta la API de VirusTotal para verificar si un dominio es malicioso."""
        if dominio.endswith(('.local', '.localdomain')) or '.' not in dominio:
            return

        ahora = time.time()
        if ahora - self.last_api_call_time < 15:
            return
        self.last_api_call_time = ahora

        api_key = "f26f1995f571c4577509da7d35038464fbcb0d39378bdf109e7accc040e238af"
        if api_key == "AQUÍ_VA_TU_CLAVE_DE_API_DE_VIRUSTOTAL":
            self.log_queue.put("ADVERTENCIA: La clave de API de VirusTotal no ha sido configurada.")
            return

        url = f"https://www.virustotal.com/api/v3/domains/{dominio}"
        headers = {"x-apikey": api_key}

        try:
            response = requests.get(url, headers=headers)
            if response.status_code == 200:
                stats = response.json().get('data', {}).get('attributes', {}).get('last_analysis_stats', {})
                malicious_votes = stats.get('malicious', 0)
                if malicious_votes > 0:
                    razon = f"{malicious_votes} motor(es) lo marcaron como malicioso."
                    mensaje = {'tipo': 'WEB_ALERT', 'dominio': dominio, 'razon': razon, 'ip_local': ip_local}
                    self.log_queue.put(mensaje)
        except requests.RequestException as e:
            self.log_queue.put(f"INFO: No se pudo contactar con la API de VirusTotal: {e}")
```

```
def _geolocate_ip(self, ip):
    """Obtiene la geolocalización (ciudad, país) de una dirección IP pública."""
    if not self.geoip_reader or not ip or ip.startswith(('192.168.', '10.', '172.16.')):
        return "N/A (IP Privada)"
    try:
        response = self.geoip_reader.city(ip)
        city = response.city.name or ''
        country = response.country.name or ''
        return f"{city}, {country}".strip(", ")
    except geoip2.errors.AddressNotFoundError:
        return "N/A (No Encontrada)"

# --- LÓGICA PRINCIPAL DE CAPTURA Y PROCESAMIENTO ---

def run(self):
    """
    Método principal del hilo. Carga los activos y entra en un bucle de
    captura de paquetes y revisión de timeouts hasta que se le indique parar.
    """
    if not self.cargar_activos():
        self.log_queue.put("ERROR: El motor no puede iniciar.")
        return
    self.log_queue.put(f"INFO: Iniciando escucha en '{self.interfaz}'...")
    while self.seguir_capturando:
        try:
            sniff(iface=self.interfaz, prn=self.procesar_paquete, store=0, timeout=5)
            self.revisar_timeouts()
        except Exception as e:
            self.log_queue.put(f"ERROR en sniff: {e}")
            time.sleep(2)
    self.limpieza_final()
    self.log_queue.put("INFO: El motor de captura se ha detenido.")
```

```
def stop(self):
    """Señal para detener el bucle de captura de forma segura."""
    self.log_queue.put("INFO: Señal de detención recibida...")
    self.seguir_capturando = False

def limpieza_final(self):
    """Procesa los flujos que quedaron activos al momento de detener el motor."""
    self.log_queue.put("INFO: Limpiando flujos restantes...")
    for id_flujo in list(self.flujos_activos.keys()):
        self.calcular_y_clasificar(id_flujo, "Cierre del programa")

def crear_id_flujo(self, paquete):
    """Crea un identificador único para un flujo de red."""
    if IP in paquete and (TCP in paquete or UDP in paquete):
        proto = "TCP" if TCP in paquete else "UDP"
        ip_o, ip_d = sorted((paquete[IP].src, paquete[IP].dst))
        if ip_o == paquete[IP].src:
            p_o, p_d = paquete.sport, paquete.dport
        else:
            p_o, p_d = paquete.dport, paquete.sport
        return f"{ip_o}:{p_o}-{ip_d}:{p_d}-{proto}"
    return None

def calcular_y_clasificar(self, id_flujo, razon_cierre):
    """
    Extrae las características de un flujo terminado, determina la IP del
    potencial atacante y envía los datos al clasificador.
    """
    hora_actual = datetime.now()
    flujo = self.flujos_activos.pop(id_flujo, None)
    if not flujo or len(flujo['paquetes']) < 2:
        return

    features = {name: 0.0 for name in self.nombres_features}
```

```
lista_paquetes = flujo['paquetes']

ip_atacante = "Desconocida"
ip_victima = "Desconocida"
try:
    partes_id = id_flujo.split('-')
    ip1_str, p1_str = partes_id[0].split(':')
    ip2_str, p2_str = partes_id[1].split(':')
    p1, p2 = int(p1_str), int(p2_str)

    if self.local_ip and self.local_ip in [ip1_str, ip2_str]:
        ip_victima = self.local_ip
        ip_atacante = ip2_str if ip1_str == ip_victima else ip1_str
    else:
        ip_atacante = flujo['ip_cliente']

    features['Destination Port'] = p1 if ip1_str == ip_victima else p2
except Exception:
    pass

paquetes_fwd = [p for p in lista_paquetes if p['direccion'] == 'fwd']
paquetes_bwd = [p for p in lista_paquetes if p['direccion'] == 'bwd']
duracion_s = flujo['ultimo_timestamp'] - flujo['timestamp_inicio']
features['Flow Duration'] = duracion_s * 1_000_000
features['Total Fwd Packets'] = len(paquetes_fwd)
features['Total Backward Packets'] = len(paquetes_bwd)

longitudes = [p['longitud'] for p in lista_paquetes]
longitudes_fwd = [p['longitud'] for p in paquetes_fwd]
longitudes_bwd = [p['longitud'] for p in paquetes_bwd]

features['Total Length of Fwd Packets'] = float(sum(longitudes_fwd))
features['Total Length of Bwd Packets'] = float(sum(longitudes_bwd))
if longitudes_fwd:
```

```
            features['Fwd Packet Length Max'] = float(np.max(longitudes_fwd))
            features['Fwd Packet Length Min'] = float(np.min(longitudes_fwd))
            features['Fwd Packet Length Mean'] = float(np.mean(longitudes_fwd))
            features['Fwd Packet Length Std'] = float(np.std(longitudes_fwd)) if len(longitudes_fwd) > 1 else 0.0
        if longitudes_bwd:
            features['Bwd Packet Length Max'] = float(np.max(longitudes_bwd))
            features['Bwd Packet Length Min'] = float(np.min(longitudes_bwd))
            features['Bwd Packet Length Mean'] = float(np.mean(longitudes_bwd))
            features['Bwd Packet Length Std'] = float(np.std(longitudes_bwd)) if len(longitudes_bwd) > 1 else 0.0

        if duracion_s > 1e-6:
            features['Flow Bytes/s'] = sum(longitudes) / duracion_s
            features['Flow Packets/s'] = len(lista_paquetes) / duracion_s

        timestamps = sorted([p['timestamp'] for p in lista_paquetes])
        iats = np.diff(timestamps) * 1_000_000
        if len(iats) > 0:
            features['Flow IAT Mean'] = float(np.mean(iats))
            features['Flow IAT Std'] = float(np.std(iats)) if len(iats) > 1 else 0.0
            features['Flow IAT Max'] = float(np.max(iats))
            features['Flow IAT Min'] = float(np.min(iats))

        features_array = np.array([features.get(name, 0.0) for name in self.nombres_features])
        self.clasificar_flujo(features_array, ip_atacante, hora_actual)

    def clasificar_flujo(self, features, ip_atacante, timestamp):
        """
        Usa el modelo para predecir la probabilidad de ataque y, si supera el umbral,
        envía una alerta formateada a la cola de mensajes.
        """
        if features.ndim == 1: features = features.reshape(1, -1)
        features_scaled = self.escalador.transform(features)
```

```
        features_reshaped = np.reshape(features_scaled, (features_scaled.shape[0], 1, features_scaled.shape[1]))
        probabilidad = self.modelo.predict(features_reshaped, verbose=0)[0][0]
        if probabilidad >= self.umbral:
            geo_info = self._geolocate_ip(ip_atacante)
            mensaje = {"tipo": "ALERTA", "ip": ip_atacante, "confianza": f"{probabilidad:.2%}", "geo": geo_info, "timestamp": timestamp}
            self.log_queue.put(mensaje)

    def procesar_paquete(self, paquete):
        """
        Función 'callback' para cada paquete. Intercepta consultas DNS para
        verificar dominios y agrupa el resto de paquetes en flujos.
        """
        if IP not in paquete: return

        if paquete.haslayer(DNS) and paquete.haslayer(DNSQR) and UDP in paquete and paquete[UDP].dport == 53:
            try:
                dominio = paquete[DNSQR].qname.decode().rstrip('.')
                ip_solicitante = paquete[IP].src
                verificador_thread = threading.Thread(target=self.verificar_dominio, args=(dominio, ip_solicitante), daemon=True)
                verificador_thread.start()
            except Exception:
                pass

        if paquete[IP].src in self.blocklist or paquete[IP].dst in self.blocklist: return

        id_flujo = self.crear_id_flujo(paquete)
        if not id_flujo: return

        timestamp_actual = float(paquete.time)
```

```
        if id_flujo not in self.flujos_activos:
            self.flujos_activos[id_flujo] = {
                'paquetes': [], 'timestamp_inicio': timestamp_actual,
                'ip_cliente': paquete[IP].src, 'protocolo': "TCP" if TCP in paquete else "UDP"
            }

        flujo = self.flujos_activos[id_flujo]
        flujo['ultimo_timestamp'] = timestamp_actual

        direccion = 'fwd' if paquete[IP].src == flujo['ip_cliente'] else 'bwd'
        flags_str = str(paquete[TCP].flags) if TCP in paquete else ''

        info_paquete = {'timestamp': timestamp_actual, 'longitud': len(paquete), 'flags': flags_str,
'direccion': direccion}
        flujo['paquetes'].append(info_paquete)

        if flujo['protocolo'] == "TCP" and ('F' in flags_str or 'R' in flags_str):
            if id_flujo in self.flujos_activos:
                self.calcular_y_clasificar(id_flujo, "Flag FIN/RST")

    def revisar_timeouts(self):
        """Finaliza y procesa los flujos inactivos."""
        timestamp_actual = time.time()
        flujos_a_eliminar = [id for id, flujo in self.flujos_activos.items() if timestamp_actual -
flujo['ultimo_timestamp'] > self.FLUJO_TIMEOUT]
        for id_flujo in flujos_a_eliminar:
            if id_flujo in self.flujos_activos:
                self.calcular_y_clasificar(id_flujo, f"Timeout ({self.FLUJO_TIMEOUT}s)")
```

Script 3: gui_ids.py (Interfaz Gráfica de Usuario con Tkinter)

Este script crea una interfaz gráfica de usuario (GUI) utilizando la librería Tkinter. Permite al usuario controlar el motor del IDS (IDSEngine), visualizar alertas de red y web en tiempo real,

gestionar listas blancas, ver un historial de alertas y realizar acciones como bloquear IPs o buscar información adicional.

```
import tkinter as tk
from tkinter import scrolledtext, messagebox, ttk, filedialog
import threading
import queue
import psutil
import os
import subprocess
import sqlite3
from datetime import datetime
import re
from ipaddress import ip_interface

try:
    import whois
except ImportError: whois = None
try:
    import geoip2.database
except ImportError: geoip2 = None

from detector_ids_motor import IDSEngine

class App:
    def __init__(self, root):
        """
        Constructor de la aplicación. Inicializa la ventana principal, las variables de estado,
        la base de datos, las listas blanca/negra y llama a la creación de widgets.
        """
        self.root = root
        self.root.title("IDS Prototipo Militar - Centro de Comando v4")
        self.root.geometry("1200x800")
        self.ids_thread, self.db_conn = None, None
```

```
        self.log_queue = queue.Queue()
        self.threat_widgets, self.threat_counts = {}, {}
        self.web_threat_widgets = {}
        self.whitelist_file, self.blocklist_file = "lista_blanca.txt", "lista_negra.txt"
        self.whitelist = self._load_list_from_file(self.whitelist_file)
        self.blocklist = self._load_list_from_file(self.blocklist_file)
        self.machine_ip = None
        self.local_network_ips = set()
        self.web_threat_frame = None
        self._setup_database()
        self.create_widgets()
        self.process_log_queue()

    def _setup_database(self):
        """Conecta o crea la base de datos SQLite para almacenar el historial de alertas."""
        try:
            self.db_conn = sqlite3.connect('historial_alertas.db', check_same_thread=False)
            cursor = self.db_conn.cursor()
            cursor.execute('CREATE TABLE IF NOT EXISTS alertas (id INTEGER PRIMARY
KEY AUTOINCREMENT, timestamp TEXT, ip_atacante TEXT, confianza TEXT,
geolocalizacion TEXT)')
            self.db_conn.commit()
            self.log_queue.put("INFO: Base de datos de alertas lista.")
        except Exception as e: self.log_queue.put(f"ERROR: No se pudo configurar la base de
datos: {e}")

    # --- FUNCIONES AUXILIARES DE MANEJO DE ARCHIVOS ---
    def _load_list_from_file(self, filename):
        """Carga una lista (blanca o negra) desde un archivo de texto."""
        if not os.path.exists(filename):
            with open(filename, 'w') as f: pass
            return set()
        try:
            with open(filename, 'r') as f: return set(line.strip() for line in f if line.strip())
```

```
        except Exception as e:
            self.log_queue.put(f"ERROR: No se pudo leer {filename}: {e}")
            return set()

    def _add_ip_to_file(self, ip, filename):
        """Añade una dirección IP a un archivo de texto."""
        try:
            with open(filename, 'a') as f: f.write(ip + '\n')
            os.chmod(filename, 0o666)
        except Exception as e: self.log_queue.put(f"ERROR: No se pudo escribir en {filename}: {e}")

    # --- CREACIÓN Y CONFIGURACIÓN DE LA INTERFAZ GRÁFICA ---
    def create_widgets(self):
        """Define y organiza todos los elementos visuales de la aplicación (botones, paneles, etc.)."""
        top_frame = ttk.Frame(self.root, padding="10 5 10 5")
        top_frame.pack(fill=tk.X)
        main_paned_window = ttk.PanedWindow(self.root, orient=tk.HORIZONTAL)
        main_paned_window.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        ttk.Label(top_frame, text="Interfaz:").pack(side=tk.LEFT, padx=(0, 5))
        try: self.interfaces = [iface for iface in psutil.net_if_addrs().keys() if iface != 'lo']
        except: self.interfaces = []
        self.iface_var = tk.StringVar(value=self.interfaces[0] if self.interfaces else "")
        self.iface_menu = ttk.Combobox(top_frame, textvariable=self.iface_var, values=self.interfaces, state="readonly", width=15)
        self.iface_menu.pack(side=tk.LEFT, padx=5)
        self.start_button = ttk.Button(top_frame, text="Iniciar Captura", command=self.start_ids)
        self.start_button.pack(side=tk.LEFT, padx=10)
        self.stop_button = ttk.Button(top_frame, text="Detener Captura", command=self.stop_ids, state=tk.DISABLED)
        self.stop_button.pack(side=tk.LEFT)
```

```
        self.export_button = ttk.Button(top_frame, text="Exportar Registro", command=self.export_log)
        self.export_button.pack(side=tk.RIGHT, padx=5)
        self.whitelist_button = ttk.Button(top_frame, text="Gestionar Lista Blanca", command=self.show_whitelist_window)
        self.whitelist_button.pack(side=tk.RIGHT, padx=5)
        self.history_button = ttk.Button(top_frame, text="Ver Historial de Alertas", command=self.show_history_window)
        self.history_button.pack(side=tk.RIGHT, padx=5)

        log_frame = ttk.Frame(main_paned_window, padding=5)
        ttk.Label(log_frame, text="Registro de Actividad", font=("Helvetica", 12, 'bold')).pack(anchor=tk.W)
        self.log_text = scrolledtext.ScrolledText(log_frame, wrap=tk.WORD, state=tk.DISABLED, font=("Consolas", 9))
        self.log_text.pack(fill=tk.BOTH, expand=True, pady=5)
        self.log_text.tag_config('alerta', foreground='red', font=('Helvetica', 9, 'bold'))
        self.log_text.tag_config('info', foreground='blue')
        self.log_text.tag_config('exito', foreground='green')
        main_paned_window.add(log_frame, weight=3)

        right_pane = ttk.PanedWindow(main_paned_window, orient=tk.VERTICAL)

        net_threat_container = ttk.Frame(right_pane, padding=5)
        ttk.Label(net_threat_container, text="Panel de Amenazas de Red", font=("Helvetica", 12, 'bold')).pack(anchor=tk.W)
        canvas_net = tk.Canvas(net_threat_container, borderwidth=0, highlightthickness=0)
        scrollbar_net = ttk.Scrollbar(net_threat_container, orient="vertical", command=canvas_net.yview)
        self.threat_frame = ttk.Frame(canvas_net)
        self.threat_frame.bind("<Configure>", lambda e: canvas_net.configure(scrollregion=canvas_net.bbox("all")))
        canvas_net.create_window((0, 0), window=self.threat_frame, anchor="nw")
        canvas_net.configure(yscrollcommand=scrollbar_net.set)
```

```
        canvas_net.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        scrollbar_net.pack(side=tk.RIGHT, fill=tk.Y)
        right_pane.add(net_threat_container, weight=1)

        web_threat_container = ttk.Frame(right_pane, padding=5)
        ttk.Label(web_threat_container, text="Panel de Amenazas Web", font=("Helvetica", 12,
'bold')).pack(anchor=tk.W)
        canvas_web = tk.Canvas(web_threat_container, borderwidth=0, highlightthickness=0)
        scrollbar_web = ttk.Scrollbar(web_threat_container, orient="vertical",
command=canvas_web.yview)
        self.web_threat_frame = ttk.Frame(canvas_web)
        self.web_threat_frame.bind("<Configure>", lambda e:
canvas_web.configure(scrollregion=canvas_web.bbox("all")))
        canvas_web.create_window((0, 0), window=self.web_threat_frame, anchor="nw")
        canvas_web.configure(yscrollcommand=scrollbar_web.set)
        canvas_web.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        scrollbar_web.pack(side=tk.RIGHT, fill=tk.Y)
        right_pane.add(web_threat_container, weight=1)

        main_paned_window.add(right_pane, weight=1)

        self.status_var = tk.StringVar(value="Estado: Detenido")
        self.statusbar = ttk.Label(self.root, textvariable=self.status_var, relief=tk.SUNKEN,
anchor=tk.W, padding=2)
        self.statusbar.pack(side=tk.BOTTOM, fill=tk.X)

    # --- VENTANAS ADICIONALES Y GESTIÓN DE WIDGETS ---

    def update_web_threat(self, ip_local, dominio, razon):
        """Crea o actualiza un widget en el panel de amenazas web."""
        if dominio in self.web_threat_widgets:
            widget_dict = self.web_threat_widgets[dominio]
            widget_dict['count'] += 1
            label_text = (f"IP Local: {ip_local}\nSitio Peligroso: {dominio}\n"
```

```
                f"Razón: {razon} | Alertas: {widget_dict['count']}")
            widget_dict['label'].config(text=label_text)
        else:
            web_threat_widget = ttk.Frame(self.web_threat_frame, padding=5, relief=tk.RIDGE, borderwidth=1)
            web_threat_widget.pack(fill=tk.X, pady=2, padx=2)
            style = ttk.Style()
            style.configure('WebAlert.TFrame', background='light salmon')
            web_threat_widget.config(style='WebAlert.TFrame')
            count = 1
            label_text = (f"IP Local: {ip_local}\nSitio Peligroso: {dominio}\n"
                f"Razón: {razon} | Alertas: {count}")
            info_label = ttk.Label(web_threat_widget, text=label_text, font=("Helvetica", 10), background='light salmon')
            info_label.pack(side=tk.LEFT, expand=True, fill=tk.X, anchor=tk.W)
            self.web_threat_widgets[dominio] = {'frame': web_threat_widget, 'label': info_label, 'count': count}

    def show_whitelist_window(self):
        """Muestra una nueva ventana para añadir o quitar IPs de la lista blanca."""
        wl_window = tk.Toplevel(self.root)
        wl_window.title("Gestionar Lista Blanca")
        wl_window.geometry("400x350")
        wl_window.transient(self.root)
        wl_window.grab_set()

        frame = ttk.Frame(wl_window, padding="10")
        frame.pack(fill="both", expand=True)
        list_frame = ttk.Frame(frame)
        list_frame.pack(fill="both", expand=True, pady=5)
        listbox = tk.Listbox(list_frame, selectmode=tk.SINGLE)
        listbox.pack(side=tk.LEFT, fill="both", expand=True)
        scrollbar = ttk.Scrollbar(list_frame, orient="vertical", command=listbox.yview)
        scrollbar.pack(side=tk.RIGHT, fill="y")
```

```
        listbox.config(yscrollcommand=scrollbar.set)
        for ip in sorted(list(self.whitelist)): listbox.insert(tk.END, ip)

        entry_frame = ttk.Frame(frame)
        entry_frame.pack(fill="x", pady=5)
        ttk.Label(entry_frame, text="IP a añadir:").pack(side=tk.LEFT)
        ip_entry = ttk.Entry(entry_frame)
        ip_entry.pack(side=tk.LEFT, fill="x", expand=True, padx=5)

        def add_ip():
            ip = ip_entry.get().strip()
            if ip and ip not in self.whitelist:
                self.whitelist.add(ip)
                listbox.insert(tk.END, ip)
                ip_entry.delete(0, tk.END)
                self.log_queue.put(f"INFO: {ip} añadida a la lista blanca.")
        add_button = ttk.Button(entry_frame, text="Añadir", command=add_ip)
        add_button.pack(side=tk.LEFT)

        def remove_ip():
            selected = listbox.curselection()
            if selected:
                selected_ip = listbox.get(selected[0])
                if messagebox.askyesno("Confirmar", f"¿Quitar {selected_ip} de la lista blanca?",
parent=wl_window):
                    self.whitelist.remove(selected_ip)
                    listbox.delete(selected[0])
                    self.log_queue.put(f"INFO: {selected_ip} quitada de la lista blanca.")

        def save_and_close():
            try:
                with open(self.whitelist_file, 'w') as f:
                    for ip in sorted(list(self.whitelist)): f.write(ip + '\n')
                self.log_queue.put(f"ÉXITO: Lista blanca guardada en '{self.whitelist_file}'.")
```

```
                wl_window.destroy()
            except Exception as e:
                messagebox.showerror("Error al Guardar", f"No se pudo escribir en el archivo: {e}", parent=wl_window)

        button_frame = ttk.Frame(frame)
        button_frame.pack(fill="x", pady=10)
        remove_button = ttk.Button(button_frame, text="Quitar Seleccionada", command=remove_ip)
        remove_button.pack(side=tk.LEFT, expand=True)
        save_button = ttk.Button(button_frame, text="Guardar y Cerrar", command=save_and_close)
        save_button.pack(side=tk.RIGHT, expand=True)

    def show_history_window(self):
        """Muestra una ventana con el historial de alertas consultando la base de datos."""
        history_window = tk.Toplevel(self.root)
        history_window.title("Historial de Alertas")
        history_window.geometry("800x500")

        frame = ttk.Frame(history_window, padding="10")
        frame.pack(fill="both", expand=True)

        cols = ('ID', 'Timestamp', 'IP Atacante', 'Confianza', 'Geolocalización')
        tree = ttk.Treeview(frame, columns=cols, show='headings')
        for col in cols: tree.heading(col, text=col)
        tree.pack(fill="both", expand=True)

        def populate_tree():
            for i in tree.get_children(): tree.delete(i)
            try:
                cursor = self.db_conn.cursor()
                cursor.execute("SELECT * FROM alertas ORDER BY timestamp DESC")
                for row in cursor.fetchall(): tree.insert("", "end", values=row)
```

```
            except Exception as e:
                messagebox.showerror("Error", f"No se pudo leer el historial: {e}", parent=history_window)

        refresh_button = ttk.Button(frame, text="Refrescar", command=populate_tree)
        refresh_button.pack(pady=10)
        populate_tree()

    # --- FUNCIONES DE LÓGICA Y CONTROL DEL IDS ---

    def _get_interface_details(self, iface):
        """Obtiene la IP de la máquina y las IPs de la red local para evitar falsos positivos."""
        self.machine_ip = None
        self.local_network_ips = set()
        self.log_queue.put(f"INFO: Obteniendo detalles de la interfaz '{iface}'...")
        try:
            result = subprocess.run(["ip", "address", "show", "dev", iface], capture_output=True, text=True, check=True)
            inet_line = re.search(r"inet ([\d\.]+/\d+)", result.stdout)
            if not inet_line: return

            interface = ip_interface(inet_line.group(1))
            self.machine_ip = str(interface.ip)
            net = interface.network
            self.local_network_ips.add(self.machine_ip)
            self.local_network_ips.add(str(net.network_address))
            self.local_network_ips.add(str(net.broadcast_address))
            self.log_queue.put(f"INFO: IP de la máquina: {self.machine_ip}")
        except Exception as e:
            self.log_queue.put(f"ERROR al obtener detalles de la interfaz: {e}")

    def update_threat(self, ip, confianza, geo):
        """Crea o actualiza un widget en el panel de amenazas de red."""
        if ip in self.whitelist or ip in self.local_network_ips: return
```

```
        if ip not in self.threat_widgets:
            self.threat_widgets[ip] = {'count': 0}
            ip_frame = ttk.Frame(self.threat_frame, padding=5, relief=tk.RIDGE, borderwidth=1)
            ip_frame.pack(fill=tk.X, pady=2, padx=2)

            label_text = f"IP: {ip} ({geo})\nConfianza: {confianza} | Alertas: 1"
            info_label = ttk.Label(ip_frame, text=label_text, font=("Helvetica", 10))
            info_label.pack(side=tk.LEFT, expand=True, fill=tk.X, anchor=tk.W)

            safe_button = ttk.Button(ip_frame, text="No Malicioso", command=lambda i=ip: self.mark_as_safe(i))
            safe_button.pack(side=tk.RIGHT, padx=2)
            block_button = ttk.Button(ip_frame, text="Bloquear", command=lambda i=ip: self.block_ip(i))
            block_button.pack(side=tk.RIGHT, padx=2)
            info_button = ttk.Button(ip_frame, text="Más Info", command=lambda i=ip: self.get_ip_info(i))
            info_button.pack(side=tk.RIGHT, padx=2)

            self.threat_widgets[ip].update({'frame': ip_frame, 'label': info_label, 'buttons': [block_button, info_button, safe_button]})

        self.threat_widgets[ip]['count'] += 1
        label_text = f"IP: {ip} ({geo})\nConfianza: {confianza} | Alertas: {self.threat_widgets[ip]['count']}"
        self.threat_widgets[ip]['label'].config(text=label_text)

    # --- ACCIONES DEL USUARIO ---

    def mark_as_safe(self, ip):
        """Añade una IP a la lista blanca y deshabilita sus botones en la GUI."""
        self.whitelist.add(ip)
        self._add_ip_to_file(ip, self.whitelist_file)
```

```
        self.log_queue.put(f"INFO: IP {ip} marcada como no maliciosa.")
        if ip in self.threat_widgets:
            style = ttk.Style()
            style.configure(f'Safe.{ip}.TFrame', background='pale green')
            self.threat_widgets[ip]['frame'].config(style=f'Safe.{ip}.TFrame')
            for button in self.threat_widgets[ip]['buttons']: button.config(state=tk.DISABLED)

    def block_ip(self, ip):
        """Usa 'iptables' para bloquear una IP en el firewall del sistema."""
        self.blocklist.add(ip)
        self._add_ip_to_file(ip, self.blocklist_file)
        try:
            subprocess.run(['sudo', 'iptables', '-A', 'INPUT', '-s', ip, '-j', 'DROP'], check=True)
            self.log_queue.put(f"ÉXITO: IP {ip} bloqueada en el firewall.")
            messagebox.showinfo("Firewall", f"La IP {ip} ha sido bloqueada.")
        except Exception as e:
            messagebox.showerror("Error de Firewall", f"No se pudo bloquear la IP {ip}.\n"
                                 f"Asegúrate de ejecutar con 'sudo'. Error: {e}")

    def get_ip_info(self, ip):
        """Obtiene y muestra información 'Whois' para una dirección IP."""
        if whois is None: return
        try:
            domain_info = whois.whois(ip)
            info = "\n".join([f"{k.replace('_', ' ').title()}: {v}" for k, v in domain_info.items() if v])
            messagebox.showinfo(f"Información de {ip}", info if info else "No se encontró información.")
        except Exception as e:
            messagebox.showerror("Error de Whois", f"No se pudo obtener información: {e}")

    def process_log_queue(self):
        """
        Bucle que se ejecuta periódicamente para procesar mensajes del motor del IDS
        (alertas, info, errores) y mostrarlos en la GUI.
```

```
        """
        try:
            while True:
                msg = self.log_queue.get_nowait()
                autoscroll = self.log_text.yview()[1] == 1.0
                tag = 'normal'

                if isinstance(msg, dict) and msg.get("tipo") == "ALERTA":
                    msg_str = f"!!! ALERTA DE RED: {msg['ip']} (Confianza: {msg['confianza']})
!!!"
                    tag = 'alerta'
                    self.update_threat(msg['ip'], msg['confianza'], msg.get('geo', 'N/A'))
                    self._log_alert_to_db(msg)
                elif isinstance(msg, dict) and msg.get("tipo") == "WEB_ALERT":
                    if msg['dominio'] not in self.web_threat_widgets:
                        messagebox.showwarning("Alerta de Navegación", f"Detectado intento de
conexión a sitio malicioso:\n\n"
                                               f"Dominio: {msg['dominio']}\nCausa: {msg['razon']}")
                    self.update_web_threat(msg.get('ip_local', 'N/A'), msg['dominio'], msg['razon'])
                    msg_str = f"!!! ALERTA WEB: {msg['dominio']} ({msg['razon']}) !!!"
                    tag = 'alerta'
                else:
                    msg_str = str(msg)
                    if msg.startswith("INFO:"): tag = 'info'
                    elif msg.startswith("ÉXITO:"): tag = 'exito'

                self.log_text.config(state=tk.NORMAL)
                self.log_text.insert(tk.END, msg_str + '\n', tag)
                self.log_text.config(state=tk.DISABLED)
                if autoscroll: self.log_text.see(tk.END)
        except queue.Empty: pass
        finally: self.root.after(100, self.process_log_queue)

    def export_log(self):
```

```
        """Permite al usuario guardar el contenido del registro de actividad en un archivo."""
        content = self.log_text.get("1.0", tk.END)
        filename = filedialog.asksaveasfilename(defaultextension=".log", filetypes=(("Log files", "*.log"),("All files", "*.*")))
        if filename:
            with open(filename, 'w', encoding='utf-8') as f: f.write(content)
            messagebox.showinfo("Exportar", f"Registro guardado en {filename}")

    def _log_alert_to_db(self, data):
        """Inserta los detalles de una nueva alerta en la base de datos SQLite."""
        if not self.db_conn: return
        try:
            self.db_conn.cursor().execute("INSERT INTO alertas (timestamp, ip_atacante, confianza, geolocalizacion) VALUES (?, ?, ?, ?)",
                                          (data['timestamp'].strftime('%Y-%m-%d %H:%M:%S'), data['ip'], data['confianza'], data['geo']))
            self.db_conn.commit()
        except Exception as e: self.log_queue.put(f"ERROR de DB: {e}")

    # --- CONTROL DEL CICLO DE VIDA DEL IDS ---

    def start_ids(self):
        """
        Inicia el motor del IDS en un hilo separado con la configuración seleccionada
        en la interfaz gráfica.
        """
        selected_iface = self.iface_var.get()
        if not selected_iface: return

        self.whitelist = self._load_list_from_file(self.whitelist_file)
        self._get_interface_details(selected_iface)

        modelo_path, scaler_path = 'modelo_A_optimizado_earlystopping.h5', 'escalador_estandar.joblib'
```

```
        if not os.path.exists(modelo_path) or not os.path.exists(scaler_path):
            messagebox.showerror("Error", f"Asegúrate de que '{modelo_path}' y '{scaler_path}'
estén presentes.")
            return

        self.start_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.NORMAL)
        self.status_var.set(f"Estado: Iniciando motor en {selected_iface}...")

        self.ids_thread = IDSEngine(
            interfaz=selected_iface, umbral=0.2, log_queue=self.log_queue,
            blocklist=self.blocklist, geoip_path='GeoLite2-City.mmdb',
            modelo_path=modelo_path, scaler_path=scaler_path, local_ip=self.machine_ip
        )
        self.ids_thread.start()

    def stop_ids(self):
        """Detiene el hilo del motor del IDS."""
        if self.ids_thread and self.ids_thread.is_alive(): self.ids_thread.stop()
        self.stop_button.config(state=tk.DISABLED)
        self.start_button.config(state=tk.NORMAL)
        self.status_var.set("Estado: Detenido")

    def on_closing(self):
        """
        Se ejecuta al cerrar la ventana. Detiene el IDS y cierra la conexión
        a la base de datos de forma segura.
        """
        if messagebox.askokcancel("Salir", "¿Seguro que quieres salir?"):
            if self.ids_thread and self.ids_thread.is_alive(): self.ids_thread.stop()
            if self.db_conn: self.db_conn.close()
            self.root.destroy()

if __name__ == "__main__":
```

```
"""Punto de entrada para lanzar la aplicación gráfica."""
root = tk.Tk()
app = App(root)
root.protocol("WM_DELETE_WINDOW", app.on_closing)
root.mainloop()
```