UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Detección de ataques en un entorno de red de dispositivos IoT

Presentado por Alejandro Diez Bermejo
en Universidad de Burgos — 10 de junio de 2025
Tutor: Daniel Urda Muñoz
Cotutor: Jaime Andrés Rincón Arango

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Daniel Urda Muñoz y D. Jaime Andrés Rincón Arango, profesores del departamento de Digitalización, área de Ciencia de la Computación e Inteligencia Artificial.

Exponen:

Que el alumno D. Alejandro Diez Bermejo, con DNI 71307675Q, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado "Detección de ataques en un entorno de red de dispositivos IoT" de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 10 de junio de 2025

Vº. Bº. del Tutor:

D. Daniel Urda Muñoz

Vº. Bº. del Cotutor:

D. Jaime Andrés Rincón Arango

Resumen

Este proyecto propone el desarrollo de un simulador de una red de dispositivos IoT, orientado a la experimentación con modelos de inteligencia artificial para la detección de ataques. El simulador permitirá construir una topología de red en la que se pueda generar tráfico y simular distintos tipos de ataques a los dispositivos conectados. Además, se entrenará un modelo de red neuronal utilizando aprendizaje federado, un enfoque descentralizado que prioriza la privacidad de los datos locales de cada dispositivo.

Descriptores

internet de las cosas, simulación de red, detección de ataques, inteligencia artificial, aprendizaje federado, redes neuronales, ciberseguridad, emulación de ataques, aprendizaje con preservación de la privacidad, sistemas distribuidos

Abstract

This project proposes the development of a simulator for an IoT device network, aimed at experimenting with artificial intelligence models for attack detection. The simulator will allow the construction of a network topology where traffic can be generated and various types of attacks on connected devices can be simulated. Additionally, a neural network model will be trained using federated learning, a decentralized approach that prioritizes the privacy of each device's local data.

Keywords

internet of things, network simulation, attack detection, artificial intelligence, federated learning, neural networks, cybersecurity, attack emulation, privacy-preserving learning, distributed systems

Índice general

Índice de figuras

Índice de tablas

1. Introducción

1.1. Contexto

En la actualidad, el Internet de las Cosas (IoT) ha experimentado un crecimiento significativo, convirtiéndose en una parte fundamental de nuestras vidas. Los dispositivos IoT están presentes en una amplia gama de sectores, desde los hogares hasta los sistemas industriales, lo que ha incrementado las oportunidades de interconexión y automatización. Sin embargo, este aumento en la cantidad de dispositivos conectados también ha generado un mayor interés por parte de los atacantes en buscar nuevas formas de realizar dichos ataques.

La detención de estos ataques se enfrenta a grandes desafíos debido a la heterogeneidad de los sistemas. Detectar ataques en tiempo real sin comprometer la eficiencia o la privacidad de los datos se ha convertido en una prioridad. En este contexto, la inteligencia artificial (IA) ha destacado como una herramienta poderosa para mejorar esta detección de intrusos, permitiendo analizar grandes cantidades de datos y detectar patrones de comportamiento diferentes que pudieran señalar un ataque.

El aprendizaje federado es una técnica relativamente novedosa que permite entrenar modelos de inteligencia artificial de manera descentralizada, preservando la privacidad de los datos locales en los dispositivos. Esta metodología es especialmente útil en entornos de red IoT, donde los dispositivos pueden generar datos sensibles que no deben compartirse de manera centralizada. A través del aprendizaje federado, es posible entrenar modelos en los dispositivos de forma colaborativa sin la necesidad de compartir los datos de manera directa, lo que reduce los riesgos de privacidad.

Este proyecto se centra en el desarrollo de una herramienta que permita la generación de topologías de red compuestas por dispositivos IoT, así como la simulación de tráfico y la emulación de diferentes tipos de ataques. El objetivo es disponer de un entorno de pruebas en el que se puedan integrar y evaluar distintos modelos de inteligencia artificial para la detección de ataques cibernéticos. Como parte del trabajo, se entrenará un modelo de detección basado en aprendizaje federado, el cual será comparado con su equivalente entrenado mediante un entrenamiento centralizado. Ambos modelos serán entrenados utilizando un conjunto de datos etiquetado que incluye tanto trazas de red benignas como aquellas correspondientes a distintos tipos de ataques.

1.2. Motivación

La realización de este proyecto surge a partir de mi colaboración con el Grupo de Inteligencia Computacional Aplicada (GICAP) [16] de la Universidad de Burgos, donde, durante el último curso del grado, he investigado las posibles ventajas e implementaciones de una técnica relativamente novedosa para el entrenamiento de modelos de inteligencia artificial: el aprendizaje federado. Esta técnica permite entrenar modelos de manera segura y eficaz, priorizando la privacidad y el confinamiento de los datos.

Durante esta etapa, se me propusieron varias líneas de trabajo que me sirvieron de guía para obtener conclusiones y desarrollar una implementación práctica del modelo. Como demostración del trabajo realizado, se planteó la creación de un simulador de red que permitiera visualizar cómo un modelo de inteligencia artificial es capaz de detectar ataques, sin necesidad de desplegar una infraestructura compleja.

La oportunidad de participar en un proyecto tan ambicioso ha sido una gran motivación para el desarrollo de este Trabajo Fin de Grado. Confío en que, en un futuro próximo, esta solución pueda ser aplicada en entornos reales, ya que la seguridad y la protección frente a ataques son aspectos fundamentales en nuestra sociedad tecnológicamente avanzada.

1.3. Solución

La solución propuesta consiste en el desarrollo de un entorno de simulación capaz de generar trazas de red entre dispositivos conectados, así como en el entrenamiento de un modelo de inteligencia artificial mediante aprendizaje federado. Este modelo será entrenado a partir de un conjunto

de datos que contiene diversas trazas de red, etiquetadas según el tipo de ataque al que corresponden.

Para el desarrollo del simulador se emplearán tecnologías como TypeScript y el framework Angular. Por otro lado, el entrenamiento del modelo se llevará a cabo utilizando Python, el framework Flower y la biblioteca TensorFlow, bajo el paradigma del aprendizaje federado.

Para el entrenamiento y validación del modelo se utilizará el conjunto de datos NF-ToN-IoT [21], el cual será particionado en dos partes, permitiendo que cada segmento sea utilizado por un cliente distinto. Cada archivo contendrá un número determinado de trazas de red, tanto benignas como asociadas a distintos tipos de ataques dirigidos a dispositivos IoT. Este conjunto de datos está disponible de forma gratuita a través de la plataforma Kaggle[1] y puede utilizarse con fines científicos y educativos, pero no comerciales.

1.4. Estructura de la memoria

La memoria se organiza en los siguientes apartados:

- **Introducción**: contexto del proyecto, la motivación, la solución propuesta y la estructura de la memoria. También se incluirá información sobre el repositorio del proyecto.
- **Objetivos**: definición de los objetivos generales, técnicos y personales que guían el desarrollo del proyecto.
- **Conceptos teóricos**: descripción detallada de los fundamentos teóricos y conceptos clave relacionados con el desarrollo del proyecto.
- **Técnicas y herramientas**: explicación de las metodologías y tecnologías utilizadas en el desarrollo del proyecto.
- **Desarrollo del proyecto**: explicación de las diferentes etapas de desarrollo del proyecto, incluyendo el proceso de entrenamiento federado, la implementación y las pruebas del modelo, así como la experimentación y los resultados obtenidos.
- **Trabajos relacionados**: revisión de investigaciones y proyectos previos que se relacionan con el tema tratado en este trabajo.

[1] https://www.kaggle.com/datasets/dhoogla/nftoniot?select=NF-ToN-IoT.parquet

- **Conclusiones y líneas futuras**: conclusiones derivadas del trabajo realizado, y propuestas para futuras líneas de investigación y desarrollo en el ámbito.

1.5. Repositorio

El código fuente y los recursos asociados se encuentran disponibles en el repositorio de GitHub[2]. La estructura del repositorio está organizada de tal manera que facilite la comprensión y reproducción del trabajo realizado, incluyendo directorios para las diferentes partes del proyecto, datos utilizados y documentación. Para una comprensión más detallada de la estructura del repositorio, se debe consultar la sección D.2 de los anexos.

[2] https://github.com/AlejaDiez/iot_attack_detection

2. Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir, los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto y los objetivos personales a conseguir por el alumno.

2.1. Objetivos Generales

- Desarrollar un simulador de red de dispositivos que genere tráfico entre ellos, permita lanzar ataques y soporte la integración de modelos de inteligencia artificial basados en TensorFlow para la detección de intrusiones.
- Entrenamiento de un modelo de inteligencia artificial mediante un enfoque federado de entrenamiento.

2.2. Objetivos Técnicos

- Diseñar un simulador de red usando el framework Angular que permita generar tráfico entre dispositivos y facilite la integración de modelos de inteligencia artificial.
- Implementar diversos tipos de ataques en la red simulada para evaluar la capacidad de detección del sistema.
- Dividir un conjunto de datos entre múltiples dispositivos, representando un entorno distribuido.

- Entrenar un modelo de detección de ataques utilizando TensorFlow, distribuido entre dispositivos mediante el framework de aprendizaje federado Flower.
- Recopilar y analizar métricas relevantes durante el entrenamiento federado para evaluar el rendimiento del modelo.
- Detectar ataques en la red analizando los flujos de datos con el modelo entrenado.

2.3. Objetivos Personales

- Aplicar los conocimientos adquiridos durante el Grado en Ingeniería Informática en un proyecto real y práctico.
- Profundizar en el uso de framework de desarrollo web (Angular) y entrenamiento de inteligencia artificial (TensorFlow y Flower).
- Aprender sobre entrenamientos de redes neuronales de forma distributiva e implementación en entornos reales.
- Uso de software de control de versiones como Git junto a herramientas de alojamiento de repositorios como GitHub.

3. Conceptos teóricos

En el marco de este proyecto, se abordan múltiples disciplinas pertenecientes a distintas áreas de la informática, tales como la inteligencia artificial, las redes de dispositivos y la seguridad. La parte de mayor complejidad teórica corresponde al **entrenamiento federado** de un **perceptrón multicapa (MLP)**. Con esta novedosa estrategia, es posible entrenar un modelo de inteligencia artificial de forma descentralizada, preservando la privacidad de los datos al evitar su transferencia a un servidor central. Este aspecto resulta especialmente relevante en contextos relacionados con el flujo de datos en redes, donde la confidencialidad y la seguridad de la información son prioridades fundamentales.

3.1. Aprendizaje Automático

La **inteligencia artificial (IA)** es una rama de la informática que se centra en la creación de máquinas que sean capaces de imitar la inteligencia humana para realizar tareas, y que puedan mejorar sus capacidades según recopilen información.

El **aprendizaje automático (machine learning)** [22] es una rama de la inteligencia artificial que permite a un sistema aprender a partir de un conjunto de datos, identificando patrones y relaciones de forma autónoma, sin necesidad de ser programado con instrucciones específicas para resolver una determinada tarea. Entre las distintas tareas que es capaz de abordar esta tecnología, destaca la **clasificación binaria**, que permite determinar si un conjunto de datos pertenece a una clase o a otra, resultando especialmente útil en la detección de ataques en red.

Estos sistemas emplean diversos algoritmos que se adaptan a las distintas situaciones presentadas por el conjunto de datos utilizado durante el proceso de entrenamiento. Durante cada iteración, se ajustan los parámetros del algoritmo, dando como resultado un **modelo matemático** que es capaz de relacionar las variables de entradas y las salidas deseadas.

Dentro del aprendizaje automático existen varios tipos de aprendizaje, entre ellos se encuentran el aprendizaje supervisado y el aprendizaje federado, que son los usados durante este proyecto.

Aprendizaje Supervisado

El **aprendizaje supervisado** [17] es una de las principales ramas del aprendizaje automático, y se basa en el entrenamiento de un modelo utilizando un conjunto de datos **etiquetados**, es decir, ejemplos de entrada con la salida esperada. El objetivo del modelo es aprender una función que relacione correctamente las entradas con las salidas correspondientes, para posteriormente poder realizar predicciones sobre un nuevo conjunto de datos no observado.

Durante el proceso de entrenamiento, el modelo realiza predicciones sobre cada uno de los datos de entrada y las compara con las etiquetas correspondientes. Esta comparación se realiza mediante el uso de una **función de pérdida**, que evalúa la solución predicha por el modelo frente a la solución real, devolviendo un valor de error, que se usará con técnicas de **optimización** para ajustar los parámetros del modelo y de esta forma, se irá reduciendo el error progresivamente a medida que se vaya mejorando la precisión del modelo.

Con este tipo de aprendizaje se pueden resolver dos problemas principalmente:

- **Clasificación**: este enfoque consiste en usar el algoritmo entrenado para asignar a nuevos datos una categoría específica. El modelo analiza los **patrones** de los datos de entrada ya etiquetados para extraer la información necesaria para predecir datos futuros.
- **Regresión**: este enfoque se centra en entender la relación entre las variables dependientes e independientes, produciendo así un valor numérico a partir de las variables de entrada que se le asigne.

En el desarrollo de este proyecto, se ha optado por una tarea de **clasificación**, concretamente una **clasificación binaria**. Esta elección permite

que, dado un determinado flujo de red, el modelo sea capaz de **clasificarlo como benigno o malicioso**. Para ello, se ha utilizado un conjunto de datos de entrenamiento compuesto por distintos flujos de red previamente etiquetados, donde cada uno ha sido identificado como **ataque** o **tráfico legítimo**. A través de este proceso, el modelo aprende las características que diferencian ambos tipos de tráfico y, posteriormente, puede predecir flujos no observados. La salida del modelo es un valor numérico entre 0 y 1, que, al aplicar un valor **umbral**, permite distinguir la clase a la que pertenece el flujo.

También se podría haber optado por una tarea de **clasificación multiclase**, la cual habría permitido identificar no solo si un flujo de red corresponde a un ataque, sino también el tipo específico de ataque que se está produciendo. Sin embargo, esta alternativa presenta una mayor complejidad, tanto en el diseño del modelo como en la recolección y etiquetado de los datos, ya que requiere un conjunto de datos amplio y bien balanceado. Debido a la dificultad de obtener este tipo de información, se ha considerado que este enfoque no era viable a corto plazo para los objetivos de este proyecto.

Aprendizaje Federado

El **aprendizaje federado** [23] es un enfoque de aprendizaje automático que permite entrenar un modelo en múltiples dispositivos, sin la necesidad de centralizar los datos. A diferencia de las técnicas tradicionales, donde los datos son recopilados y almacenados en servidores centrales para su posterior uso en entrenamiento de modelos, el aprendizaje federado mantiene los datos **localmente** en los dispositivos de origen, y únicamente se comparten los **parámetros** del modelo.

Existen varios tipos de aprendizaje federado, según la distribución y naturaleza de los datos entre los distintos participantes [18]:

- **Aprendizaje federado horizontal**: se aplica cuando los datos tienen características similares, pero de diferentes usuarios. Por ejemplo, el entrenamiento de un modelo con transacciones bancarias del mismo tipo, pero procedentes de distintos clientes.
- **Aprendizaje federado vertical**: se utiliza cuando los datos presentan características diferentes, pero se refieren a las mismas entidades. Por ejemplo, al combinar la información de compras de una plataforma con las reseñas de usuarios de otra, para entrenar un modelo de recomendación.

- **Aprendizaje federado por transferencia**: emplea un modelo previamente entrenado en un conjunto de datos, que se adapta posteriormente a un conjunto de datos similar para resolver una tarea relacionada. Por ejemplo, utilizar un modelo entrenado para reconocer animales y ajustarlo para identificar especies específicas.

Este enfoque permite realizar un entrenamiento de forma **colaborativa** entre múltiples clientes. Cada cliente entrena un modelo local utilizando sus propios datos. Después de varias iteraciones, el modelo local se envía al servidor central para **actualizar** los parámetros mediante algoritmos como **Federated Averaging** (subsección 3.1) y así poder obtener un modelo global que incluya el proceso de aprendizaje de cada uno de los clientes. Este modelo se vuelve a distribuir entre los clientes, quienes podrán continuar con el entrenamiento usando un modelo actualizado, realizando tantas rondas sucesivas hasta alcanzar la **convergencia** del modelo. Todo este proceso se puede observar de manera gráfica en la siguiente figura 3.1.

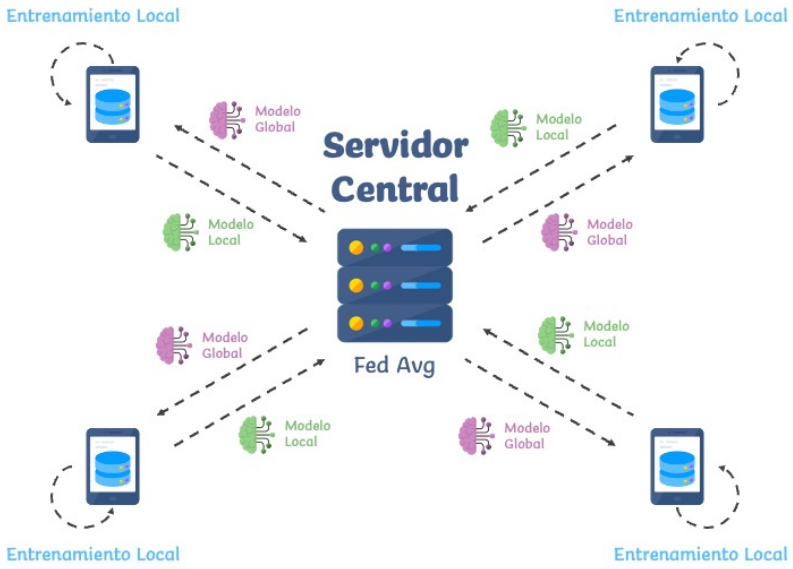

Figura 3.1: Arquitectura de un sistema de aprendizaje federado [25]

Gracias a esta técnica de aprendizaje, los datos de entrenamiento permanecen en todo momento en el cliente, lo que ofrece grandes ventajas en términos de **privacidad**, **seguridad** y cumplimiento de normativa relacionadas con la **Protección de Datos**.

Sin embargo, existen varios **retos** a los que enfrentarse, entre los que destacan [19]:

- **Datos no independientes ni idénticamente distribuidos (no-IID)**: los datos locales de cada cliente pueden tener distribuciones diferentes, lo que dificulta la convergencia del modelo.
- **Conectividad y recursos limitados**: algunos dispositivos pueden tener restricciones de hardware, batería o conectividad que afecten a su capacidad de participar de manera constante.
- **Heterogeneidad del sistema**: la variabilidad entre los dispositivos (arquitectura, capacidad de cómputo, sistema operativo) introduce complejidad en la organización del entrenamiento.

En este proyecto, se ha optado por un enfoque de **aprendizaje federado supervisado**, utilizando una arquitectura de datos **horizontal**. Para simular el entorno federado, se ha dividido el conjunto de datos en partes iguales, proporcionando a cada cliente un subconjunto de datos etiquetados, que se usarán para entrenar el modelo de **clasificación binaria** de manera local. Gracias a esta técnica, es posible desarrollar un modelo de detección de ataques basado en flujos de red, garantizando la privacidad. Esta característica resulta especialmente relevante dada la naturaleza sensible de la información analizada.

Algoritmo Federated Averaging (FedAvg)

El algoritmo **Federated Averaging (FedAvg)** [19], es uno de los enfoques más usados en el ámbito del aprendizaje federado. Su objetivo es permitir el entrenamiento de forma eficiente de modelos sobre datos distribuidos, preservando la privacidad de los usuarios. Este algoritmo combina el entrenamiento local de modelos y la agregación de los modelos locales en el servidor central para generar un modelo global. El procedimiento general del algoritmo es el siguiente:

1. El **servidor central** inicializa el modelo global con unos parámetros iniciales w_0.
2. En cada ronda de entrenamiento:
   a) Se selecciona aleatoriamente una fracción de clientes disponibles.

b) El servidor envía el modelo global actual a los clientes seleccionados.

c) Cada **cliente** entrena el modelo localmente usando sus propios datos durante varias iteraciones, y actualiza sus parámetros a w_{t+1}^k.

d) Los modelos locales se devuelven al servidor.

e) El servidor realiza un **promedio ponderado** de los parámetros, considerando el número de ejemplos de entrenamiento de cada cliente:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$$

donde n_k es el número de muestras del cliente k, y n es el total de muestras de todos los clientes participantes en esa ronda.

3. Este proceso se repite durante múltiples rondas hasta alcanzar la **convergencia** del modelo global.

Este algoritmo permite que los clientes realicen múltiples pasos de entrenamiento local antes de comunicarse con el servidor, lo que reduce significativamente el número de rondas necesarias. Además, ha demostrado ser eficaz incluso cuando los datos están distribuidos de forma no balanceada y no-IID, como suele ocurrir en escenarios reales.

3.2. Perceptrón Multicapa (MLP)

Un **Perceptrón Multicapa (MLP)** [12] es un tipo de **Red Neuronal Artificial (ANN)** que consta de varias capas de neuronas interconectadas entre sí, logrando poder detectar patrones complejos en los datos (figura 3.2).

Las capas se pueden identificar según su tipo:

- La **capa de entrada** recibe los datos de entrada de la red neuronal, y tendrá tantas neuronas como inputs necesite el modelo.
- Las **capas ocultas** son capas intermedias que se encuentran entre la capa de entrada y la capa de salida.
- La **capa de salida** genera la salida final de la red neuronal. Esta capa tendrá tantas neuronas como valores de salida se necesite.

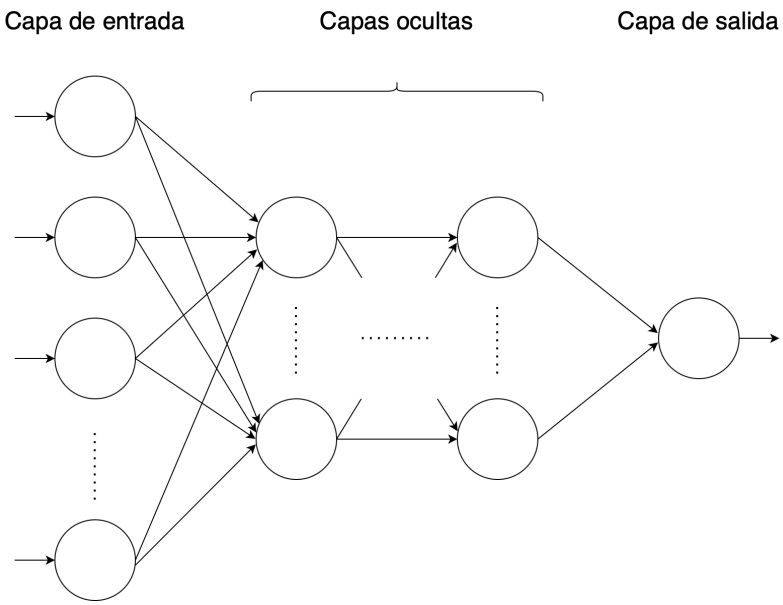

Figura 3.2: Esquema de un Perceptrón Multicapa

Cada capa está formada por un conjunto de neuronas o nodos. Los **pesos** y el **sesgo (bias)** son parámetros que se usan para ajustar la salida de cada neurona en función de las entradas que esta reciba (figura 3.3).

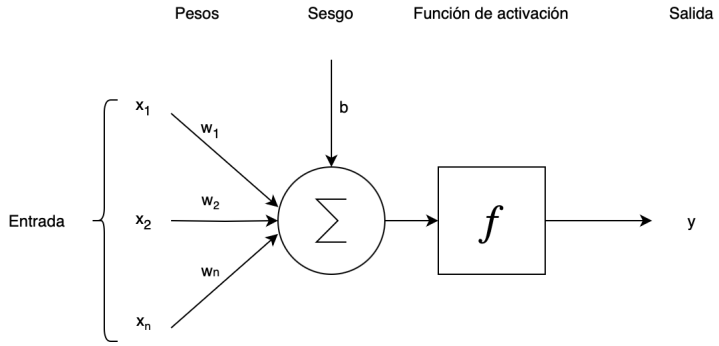

Figura 3.3: Esquema de una neurona

El propósito de una neurona artificial es realizar una combinación lineal de sus entradas, ponderadas por unos pesos, y sumar el sesgo. A este resultado se le aplica la **función de activación** (subsección 3.2) $f(x)$, para generar la salida de la neurona. Durante el **entrenamiento** (subsección 3.2), los pesos y el sesgo se ajustan mediante algoritmos de optimización, con

el objetivo de que la neurona aprenda a producir las salidas deseadas en función de sus entradas.

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

Función de activación

La **función de activación** es una parte fundamental de la neurona, ya que introduce la no linealidad al modelo. Sin el uso de estas funciones, la neurona simplemente sería un elemento de combinación lineal, limitando la capacidad de aprender de patrones complejos. También, permite filtrar la salida de la neurona, estableciendo un rango fijo de salidas para que no se produzcan valores excesivamente altos o bajos.

Existen diversas funciones de activación, y su elección depende del tipo de problema que se desea resolver.

- **ReLU** (Rectified Linear Unit)

  $$f(x) = max(0, x)$$

  Es una función simple de activación que permite activar la neurona cuando la salida sea positiva y desactivarla en caso contrario.

- **Sigmoide**

  $$f(x) = \frac{1}{1 + e^{-x}}$$

  Es una función que convierte la salida de una neurona en un valor dentro del rango $[0, 1]$, lo que la hace especialmente útil en problemas de clasificación binaria. Se utiliza con frecuencia en la capa de salida para determinar si una instancia pertenece a la clase 0 o a la clase 1.

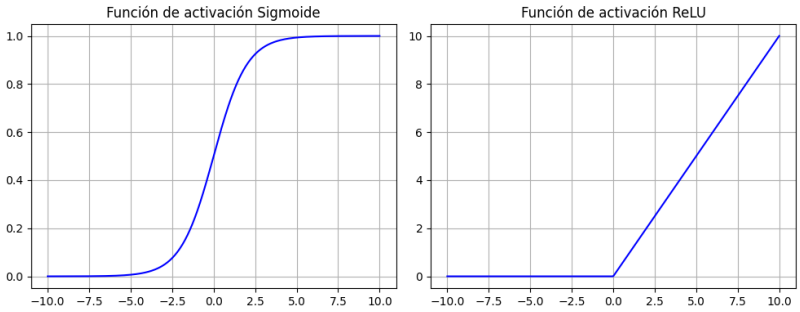

Figura 3.4: Funciones de activación

Función de pérdida

La **función de pérdida** es un componente fundamental durante el proceso de entrenamiento de un modelo, ya que proporciona una medida cuantitativa que permite evaluar la diferencia entre las predicciones del modelo y los valores reales del conjunto de entrenamiento. Este valor se utiliza en el proceso de **entrenamiento** (subsección 3.2) de la red neuronal, con el objetivo de minimizar la pérdida y, de este modo, permitir que el modelo aprenda a partir de los datos.

Existen diversas funciones de pérdida, y su elección depende del tipo de problema que se desea resolver.

- **Entropía Cruzada Binaria**

$$\ell(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

  Es una función utilizada en problemas de clasificación binaria. Compara la probabilidad predicha por el modelo, $\hat{y}$, con la etiqueta real, y, penalizando las predicciones incorrectas y recompensando a las correctas.

Función de costo

La **función de costo** es una medida global para evaluar el rendimiento del modelo durante el entrenamiento. Se obtiene al calcular el promedio de las funciones de pérdida (subsección 3.2) individuales sobre todas las muestras del conjunto de entrenamiento. Esta función proporciona un valor de error total del modelo y es la que se **minimiza** en el proceso de entrenamiento.

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{y}_i)$$

Proceso de entrenamiento

La **función de costo** (subsección 3.2) proporciona una medida global para evaluar el rendimiento del modelo sobre el conjunto de datos de entrenamiento. Durante esta etapa, el objetivo principal es **minimizar** dicho valor mediante el ajuste de los parámetros θ del modelo, con el fin de que las predicciones generadas por el modelo se aproximen lo máximo posible a los valores reales.

Para alcanzar este objetivo, se emplean **algoritmos de optimización** que permiten reducir el valor de la función de costo, favoreciendo así la **convergencia** del modelo. Un proceso de optimización adecuado ayuda a mejorar la capacidad de generalización del modelo, evitando problemas como el **sobreajuste**[3] o el **subajuste**[4]. Sin embargo, la convergencia del modelo no es un proceso rápido ni sencillo, ya que en muchos casos puede requerir un tiempo considerable, así como la posibilidad de tener un conjunto de datos de alta calidad.

Una práctica común en este proceso es dividir el conjunto de entrenamiento en **lotes más pequeños** (batches). En lugar de procesar todos los datos a la vez o uno a uno, se utiliza una estrategia intermedia. Este enfoque permite un equilibrio entre eficiencia computacional y estabilidad del gradiente, ya que se actualizan los parámetros del modelo tras procesar cada lote, lo que acelera el entrenamiento y suaviza la variabilidad del aprendizaje.

El procedimiento general del algoritmo de entrenamiento es el siguiente:

1. Se inicializa el modelo con unos parámetros iniciales.

2. Se realiza la **propagación hacia adelante** (Forward Propagation), que consiste en transmitir los datos de entrada a través de las distintas capas de la red neuronal. En cada capa, cada una de las neuronas calculan su salida aplicando la función de activación, y estas salidas se usan como entradas para la siguiente capa.

3. Se calcula la **función de costo** para calcular el error entre las predicciones del modelo y las etiquetas.

[3] El sobreajuste (overfitting) ocurre cuando el modelo aprende demasiado bien los datos de entrenamiento, incluyendo el ruido, lo que deteriora su rendimiento en datos nuevos.

[4] El subajuste (underfitting) ocurre cuando el modelo no logra capturar la complejidad de los datos, lo que resulta en un rendimiento bajo tanto en el conjunto de entrenamiento como en el de prueba.

4. Una vez obtenido el costo, se empieza con la **propagación hacia atrás** (Backward Propagation), que calcula el **gradiente** de la función del costo con respecto a cada parámetro de la red neuronal. Este gradiente indica la dirección y magnitud con la que tiene que ajustarse los parámetros para minimizar el error y así ir modificando cada parámetro a lo largo de las capas en función del impacto de cada parámetro en el error global.

5. Con el gradiente calculado, se realiza la **actualización de los pesos y sesgos** de la red neuronal, según el algoritmo de optimización utilizado.

6. Para hacer que la red neuronal llegue a un estado avanzado de entrenamiento, es necesario realizar esta secuencia de pasos varias veces, hasta alcanzar la **convergencia** del modelo.

Métricas de evaluación

Para evaluar y comparar el rendimiento de los modelos de aprendizaje automático, es necesario utilizar diversas métricas que permitan poder obtener una visión general del desempeño del modelo durante el aprendizaje.

Exactitud (Accuracy)

Mide la proporción de predicciones correctas realizadas por el modelo, sobre el conjunto total de muestras evaluadas.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Es un indicador general del rendimiento, pero no puede llegar a ser útil en conjuntos de datos desbalanceados, donde predecir la clase mayoritaria puede dar lugar a una alta precisión sin una detección significativa [2].

Pérdida (Loss)

Cuantifica como de alejadas están las predicciones del modelo con respecto a las etiquetas reales. Cuando los valores sean más bajos, indican un mejor ajuste del modelo [6].

Precisión (Precision)

Indica cual es la proporción de instancias positivas correctamente predichas por el modelo sobre el número total de instancias predichas como

positivas.

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

Una alta precisión implica menos falsas alarmas, lo cual es importante en sistemas donde los falsos positivos implican un coste [2].

Sensibilidad (Recall)

Es la proporción de instancias positivas reales que fueron correctamente identificadas por el modelo.

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

Una alta sensibilidad es esencial ya que indica que el modelo está detectando la mayoría de los casos positivos reales [2].

Puntuación F1 (F1-scrore)

Medida equilibrada que junta la precisión y sensibilidad en una sola.

$$F1\text{-}score = 2 \cdot \frac{Precisión \cdot Sensibilidad}{Precisión + Sensibilidad}$$

Es especialmente útil cuando el conjunto de datos está desbalanceado y se deben considerar tanto los falsos positivos como los falsos negativos [2].

Curva ROC

Gráfica que representa la tasa de verdaderos positivos frente a la tasa de falsos positivos para distintos umbrales. El área bajo la curva (AUC) resume el rendimiento en todos los umbrales. Un modelo con un AUC cercano a 1.0, significa que el modelo es altamente eficaz [3].

Matriz de Confusión

Se representa mediante una tabla que muestra el número de predicciones correctas e incorrectas para cada clase. Permite un análisis detallado de los errores del modelo [10].

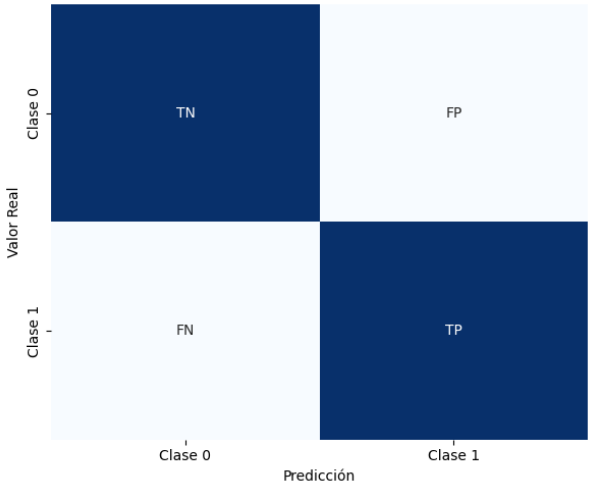

Figura 3.5: Matriz de Confusión

3.3. Redes de dispositivos

Las **redes de dispositivos** es una rama fundamental de la informática y las telecomunicaciones que se centra en la conexión entre dispositivos, la comunicación entre ellos y la gestión de recursos compartidos. Su objetivo principal es facilitar el intercambio de información y la colaboración entre diferentes dispositivos.

Estas redes pueden estar compuestas por ordenadores, pequeños dispositivos, sensores, dispositivos móviles, entre otros. Con el auge del **Internet de las Cosas (IoT)**, estas redes se han ido expandiendo y utilizando con mayor frecuencia, desde entornos domésticos hasta aplicaciones industriales. Esta expansión ha conllevado una evolución constante tanto en arquitectura como en los medios de transmisión, con el objetivo de garantizar una conexión eficiente, escalable y segura.

Modelo OSI

El **Modelo OSI (Open Systems Interconnection)** [14] es un modelo conceptual que expone un protocolo estándar de comunicación, para que otros dispositivos sean capaces de comunicarse entre ellos.

Este modelo se basa en dividir el sistema de comunicación en siete capas distintas, organizadas de forma jerárquica y cada una apilada sobre otra. Cada capa tiene una función específica y se encarga de ofrecer servicios a

la capa superior, usando los servicios que proporcionan las capas inferiores. De este modo, se puede establecer una arquitectura modular que facilita el diseño, implementación y mantenimiento de sistemas más complejos.

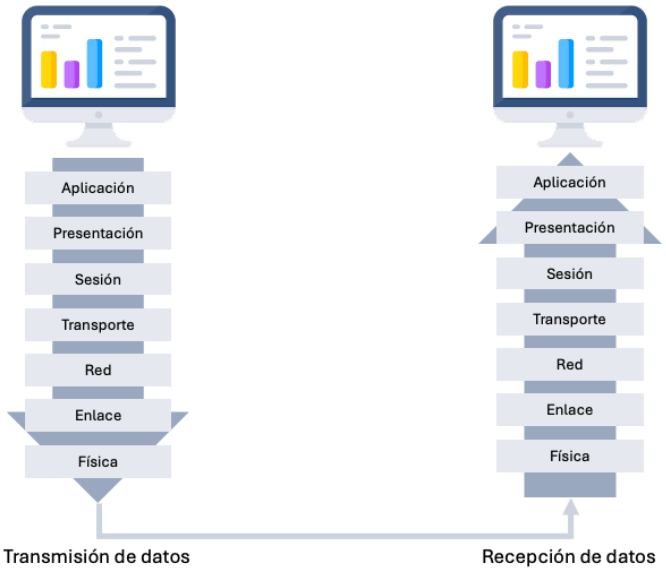

Figura 3.6: Modelo OSI [25]

Para que un mensaje se distribuya de un dispositivo a otro, los datos deben atravesar cada una de las capas en orden descendente desde el emisor, y una vez en el receptor, deben atravesar las capas en orden ascendente (figura 3.6). Esto garantiza que cada capa cumpla su función específica en la preparación, transmisión, recepción y reconstrucción de los datos.

Capa de aplicación

Esta capa es la única que interactúa con los datos del usuario. El software, interactúa con esta capa para poder iniciar las comunicaciones. Esta capa se encarga de la gestión de protocolos y la manipulación de los datos, con los que luego el software va a trabajar.

Existen múltiples protocolos:

- **Protocolo de Transferencia de Hipertexto (HTTP)**: se usa para las comunicaciones entre navegadores web y servidores.
- **Protocolo Simple de Transferencia de Correo (SMTP)**: se usa para el envío de correos electrónicos a través de redes.

- **Protocolo de Transferencia de Archivos (FTP)**: se usa para la transferencia de archivos entre un cliente y un servidor.
- **Sistema de Nombre de Dominios (DNS)**: sistema que traduce nombres de dominio legibles por humanos a direcciones IP.

Capa de presentación

Esta capa es la responsable de preparar los datos para que los pueda usar la capa de aplicación.

- **Traducción**: capaz de codificar y descodificar los datos recibidos, para que las demás capas sean capaces de entenderlos.
- **Cifrado**: se responsabiliza de encriptar en el extremo del emisor, así como de desencriptar en el extremo del receptor.
- **Compresión de datos**: ayuda a comprimir los datos producidos en la capa de aplicación, para mejorar la velocidad de transferencia y minimizar la cantidad de datos transferidos.

Capa de sesión

Se responsabiliza de abrir y cerrar las comunicaciones entre los dispositivos. Esta capa mantiene la conexión abierta tanto tiempo como sea necesario para completar la comunicación, además de cerrar la sesión para ahorrar recursos. También sincroniza la transferencia de datos utilizando puntos de control.

Capa de transporte

En esta capa se encarga de las comunicaciones de extremo a extremo entre dos dispositivos. Esto implica, que antes de realizar el envío en la capa de red, realiza una fragmentación de los datos en trozos más pequeños llamados segmentos. En la capa de transporte del receptor, se encarga de rearmar los segmentos para volver a construir los datos. Además, se encarga del control de flujo y el control de errores.

En esta capa se incluye varios protocolos:

- **Protocolo de Control de Transmisión (TCP)**: está orientado a garantizar una conexión fiable y ordenada de los datos. Utiliza mecanismos de control de errores, control de flujo y retransmisión de paquetes perdidos.

- **Protocolo de Datagrama de Usuario (UDP)**: está orientado a la eficiencia y velocidad, ya que no establece la conexión entre los dispositivos y por lo tanto no garantiza de que lleguen los datos completos u ordenados.

Capa de red

Esta capa se responsabiliza de facilitar la transferencia de datos entre dos redes diferentes, en caso de que los dos dispositivos se encuentren en la misma red, no sería necesaria esta capa. Esta capa divide los segmentos de la capa de transporte en unidades más pequeñas denominadas paquetes. También busca la mejor ruta para que los paquetes lleguen a su destino (enrutamiento).

En esta capa se incluyen varios protocolos:

- **Protocolo de Internet (IP)**: es un identificador numérico único en la red que se asigna a cada dispositivo. Permite la identificación y localización de los dispositivos en la red para comunicarse entre ellos.
- **Protocolo de Mensajes de Control de Internet (ICMP)**: se usa para enviar mensajes de diagnóstico y control entre dispositivos.

Capa de enlace de datos

Similar a la capa de red, excepto que esta capa facilita la transferencia de datos entre dos dispositivos dentro la misma red. Obtiene los paquetes de la capa de red y los divide en partes más pequeñas denominadas tramas. También es responsable del control de flujo y la gestión de errores.

- **Control de Acceso al Medio (MAC)**: identificador físico único que está asignado a la tarjeta de red y se usa para identificar de forma única a los dispositivos de una red local.

Capa física

En esta capa se incluye el equipo físico que permite la transferencia de datos. En esta capa tiene lugar la conversión de los datos en una secuencia de bits (0 y 1), para poder realizar la transferencia de los datos a través del medio.

Ataques en Redes

Los ataques sobre redes son acciones maliciosas con el objetivo de interrumpir, interceptar, modificar o dañar la comunicación o los recursos de la red. Estos ataques pueden comprometer la **confidencialidad**, **integridad** o **disponibilidad** de los datos y el sistema [15]. Según el objetivo de estos ataques, se pueden dividir en distintos grupos:

Obtener acceso no autorizado

Estos ataques tienen como objetivo intentar acceder a recursos o sistemas sin tener permiso. Los atacantes suelen aprovechar las vulnerabilidades del software, usar contraseñas robadas u otras técnicas.

- **Puerta trasera (Backdoor)**: vulnerabilidades que permiten el acceso remoto mediante aplicaciones diseñadas para ello.
- **Ataques de contraseñas**: ataques dirigidos a obtener contraseñas por fuerza bruta o sniffing.
- **Escaneo (Scanning)**: técnicas de rastreo para descubrir servicios vulnerables o mal configurados.

Interrumpir el servicio

El objetivo de estos ataques es hacer que un sistema o red deje de funcionar correctamente, debido a sobrecarga de peticiones o paquetes maliciosos.

- **Denegación de Servicio (DoS)**: se intenta saturar un sistema mediante un flujo de paquetes ilegítimo [13]. Puede haber varias maneras de generar dicho tráfico:
  - **Desbordamiento de búfer**: consiste en saturar los recursos de una máquina como la memoria, tiempo de CPU o espacio disponible en el disco duro. Logrando que el sistema se ralentice y produzca comportamientos indeseados.
  - **Inundación**: consiste en saturar un sistema con una cantidad abrumadora de paquetes. Para que se logre este ataque, el atacante debe tener más ancho de banda que el atacado.
- **Denegación de Servicio Distribuido (DDoS)**: se satura un sistema debido a tráfico ilegítimo que provienen desde múltiples dispositivos

infectados. Tiene el mismo comportamiento que el ataque DoS, pero se ejecuta en varios dispositivos al mismo tiempo, produciendo un mayor daño al sistema.

Robar o corromper datos

La función de estos ataques es interceptar o modificar la información transmitida por la red.

- **Ataque de Intermediario (MITM)**: intercepta la comunicación entre dos dispositivos para espiar o modificar mensajes.
- **Inyecciones (Injection)**: inserción de comandos que manipulan la ejecución de programas.
- **Scripting entre sitios (XSS)**: envío de scripts maliciosos a través de aplicaciones web a los navegadores de los usuarios.

Destruir o dañar sistemas

Se busca causar daño permanente a sistemas o datos.

- **Ransomware**: cifra archivos del sistema y exige un rescate para su recuperación.

3.4. Sistema de detección de intrusiones

Un **sistema de detección de intrusiones** es un componente clave en la seguridad de una red. Se encarga de detectar el acceso no autorizado, comportamientos anómalos o ataques en el tráfico de red procedentes de dispositivos conectados. Estos necesitan analizar el tráfico en tiempo real, para que después puedan evaluar posibles patrones que indiquen algún indicio de amenaza.

Paquetes a Flujos

El **tráfico de red** está compuesto por paquetes individuales, cada uno contienen una carga útil además de cabeceras que se encargan de identificar el tipo de paquete, estructura, direcciones... Sin embargo, el análisis de un paquete aislado no es muy significativo en cuanto a un sistema de detección

de intrusiones, ya que no se muestra un contexto global del funcionamiento de la red.

Por esta razón, este tipo de sistemas usa lo que se denomina **flujo de red** [5], que es una secuencia de paquetes que comparten ciertos atributos importantes como las direcciones IPs, los puertos, el protocolo y el sentido del flujo, durante un determinado intervalo de tiempo.

Posibles campos que puede incluir un flujo:

- IP de origen y destino
- Puertos de origen y destino
- Protocolo de la capa de transporte
- Número de paquetes y bytes transmitidos
- Tiempo de inicio y fin del flujo
- Banderas del protocolo TCP y otros posibles datos importantes

Gracias a esta representación de los paquetes, se permite a los dispositivos de detección analizar mejor los posibles patrones que pueden surgir en una red. Esto es muy importante a la hora de la detección de ataques mediante técnicas de inteligencia artificial, ya que así se ofrece a los modelos una visión más amplia del tráfico de la red, fundamental para la identificación de comportamientos anómalos.

En el desarrollo de este proyecto, el modelo de detección entrenado se ha basado en flujos de red, no sobre paquetes. Esta elección se debe a temas de **eficiencia**, ya que se reduce enormemente la cantidad de instancias del conjunto de datos. Además, la mayoría de ataques en redes se realizan a lo **largo del tiempo** y no en un instante puntual, haciendo que el análisis basado en flujos resulte más adecuado.

4. Técnicas y herramientas

4.1. Técnicas utilizadas

En el desarrollo de este proyecto, se utilizó la metodología SCRUM para gestionar la evolución del proyecto y asegurar la calidad del resultado final.

SCRUM

La metodología SCRUM, es un marco de trabajo ágil pensado para la gestión de proyectos complejos. Esta metodología se estructura en ciclos de trabajo llamados sprints, de una duración aproximada de dos semanas. En cada sprint se entrega un producto funcional, lo que permite una evolución continua del proyecto.

Tras cada iteración, se valoraba el trabajo realizado mediante reuniones o comunicación por correo electrónico, con el fin de informar sobre la evolución del proyecto y evaluar los posibles cambios en los requisitos de este. Una vez finalizadas estas reuniones, se procedía a la planificación de la siguiente iteración, que consistía en generar las historias de usuario y tareas que se realizarían en la próxima iteración, además de realizar las estimaciones de tiempo de cada una de ellas.

Para un análisis más detallado del proceso de gestión del proyecto, debe consultar el Apéndice A.2 de los anexos.

4.2. Herramientas utilizadas

Durante el desarrollo de este proyecto, se han ido utilizado diversas herramientas para facilitar el desarrollo, la documentación y gestión en la realización del trabajo.

Herramientas de desarrollo

Visual Studio Code

Herramienta de software que permite escribir, editar, compilar y depurar código de múltiples lenguajes de programación y frameworks.

Figura 4.1: Logotipo de Visual Studio Code

Chromium

Navegador web que se ha utilizado para probar, depurar y ejecutar el simulador de red. Este navegador web sirve como base para multitud de navegadores web, dando una mayor compatibilidad de uso para el simulador.

Figura 4.2: Logotipo de Chromium

Google Colab

Entorno de desarrollo en la nube basado en Python, que permite ejecutar código en diferentes CPU y GPU. Se ha usado para utilizar la biblioteca TensorFlow Federated.

Figura 4.3: Logotipo de Google Colab

Git

Sistema de control de versiones que permite llevar un registro de los cambios realizados en el código. Permite obtener un control preciso del desarrollo del proyecto.

Figura 4.4: Logotipo de Git

Python

Lenguaje de programación de alto nivel, interpretado y multiparadigma que ha sido usado en todo lo referente a el entrenamiento de redes neuronales y evaluación de estas.

Figura 4.5: Logotipo de Python

Entre las bibliotecas y frameworks que se han usado, destacar:

- **TensorFlow/Keras**: plataforma de código abierto que permiten la creación, entrenamiento y evaluación de redes neuronales mediante aprendizaje automático. Se ha usado para diseñar el Perceptrón Multicapa.

- **Flower**: framework de código abierto que se ha usado para la implementación del aprendizaje federado y permite el entrenamiento de modelos de inteligencia artificial de forma distribuida.
- **TensorFlow Federated**: extensión de TensorFlow que está diseñada para la implementación de algoritmos de aprendizaje federado con modelos generados en TensorFlow. Esta biblioteca se usó como primera toma de contacto con el aprendizaje federado.
- **TensorFlowJS**: extensión de TensorFlow que permite convertir modelos de TensorFlow generados en Python a modelos TensorFlow compatibles con JavaScript.
- **Numpy**: biblioteca fundamental para manejar grandes conjuntos de números en Python de forma eficiente.
- **Scikit Learn**: biblioteca que incluye herramientas para clasificación, regresión, evaluación... de modelos de inteligencia artificial.
- **Matplotlib y Seaborn**: bibliotecas de visualización de datos, ofreciendo la facilidad de generar gráficas a partir de datos.
- **Argparse**: biblioteca que permite el uso avanzado de los argumentos que se pasa al script de Python.

JavaScript

Lenguaje de programación de alto nivel, interpretado y orientado a objetos, que ha sido usado en todo lo referente al simulador de red. Para poder usar JavaScript ha sido necesario instalar Node, que es el intérprete disponible para entornos de servidor o escritorio.

Figura 4.6: Logotipo de JavaScript

Entre las bibliotecas y frameworks que se han usado, destacar:

- **Typescript**: es una extensión de JavaScript que permite añadir tipado estático, interfaces, clases y otros elementos propios de lenguajes de

programación. Esto permite mejorar la calidad del código y facilita el desarrollo además de detectar errores en tiempo de compilación.

- **Angular**: framework que tiene como base TypeScript y permite construir aplicaciones de una sola página. Proporciona una gran variedad de herramientas y usa adicionalmente lenguajes como HTML y CSS para la gestión de estructura y estilo de la interfaz.
- **RxJS**: biblioteca de programación reactiva en JavaScript basada en observables, que es muy usada en Angular.
- **TensorFlow.js**: biblioteca que permite entrenar e implementar modelos de aprendizaje automático usando como lenguaje base JavaScript.
- **JS-YAML**: biblioteca que permite la gestión de archivos *.yaml*.
- **JSZip**: biblioteca que permite descomprimir archivos *.zip*.
- **Tailwind CSS**: framework de CSS que permite diseñar interfaces usando clases predefinidas sin necesidad de escribir un código CSS complejo.
- **Spartan UI**: colección de componentes para aplicaciones de Angular basada en la biblioteca shadcn/ui para Next.
- **Lucide Icons**: colección de iconos de código abierto basados en Feather Icons.

Herramientas de documentación y gestión

GitHub

Plataforma que permite alojar repositorios Git, facilitando así la colaboración entre desarrolladores mediante el control de versiones, issues, revisiones de código... Además, con **GitHub Projects** se puede generar tableros para organizar tareas en diferentes sprints y hacer un seguimiento de su progreso. También ofrece la posibilidad de publicar páginas web estáticas, funcionando como un servicio de hosting mediante **GitHub Pages**.

Figura 4.7: Logotipo de GitHub

Overleaf y LaTeX

Es un editor colaborativo en línea para crear documentos en LaTeX, un lenguaje de marcado que permite generar documentos en PDF con un estilo predefinido.

Figura 4.8: Logotipo de Overleaf

Draw.io

Herramienta que ayuda a crear diagramas de flujo, esquemas... Funciona en el navegador y permite multitud de opciones de personalización e integraciones con otras plataformas.

Figura 4.9: Logotipo de Draw

Zotero

Gestor de referencias bibliográficas que permite recopilar, organizar y citar fuentes de manera sencilla.

Figura 4.10: Logotipo de Zotero

5. Aspectos relevantes del desarrollo del proyecto

En este capítulo se recogen los aspectos más importantes durante el desarrollo del proyecto. Se incluyen los detalles más relevantes de las fases de análisis, diseño, implementación y resultados.

5.1. Desarrollo del simulador IoT

El simulador IoT desarrollado durante este proyecto tiene como objetivo emular un entorno de red realista en el que múltiples dispositivos se comunican a través de un router central, generando tráfico para que pueda ser analizado mediante un modelo de inteligencia artificial. A continuación, se describen las distintas fases de su desarrollo.

Elección de tecnología

Durante esta fase inicial del proyecto, se valoraron distintas alternativas para realizar el simulador. Entre las opciones consideradas se encontraban el desarrollo como programa de escritorio, aplicación móvil o aplicación web. Finalmente opte por la versión web, que, aunque es más limitada en cuanto a recursos, da una gran versatilidad muy grande a la hora de usar el programa, eliminando la necesidad de configuración por parte del usuario.

En cuanto a la tecnología utilizada, elegí el framework **Angular**, ya que permite construir aplicaciones de una sola página (SPA) de forma bastante estructurada, permitiendo separar la lógica de la parte visual de una forma bastante sencilla.

Arquitectura del simulador de red

La arquitectura del simulador forma la parte más importante del software y ha sido diseñada con el objetivo de reproducir (de forma sencilla) el funcionamiento de una red de dispositivos conectados. Para ello, tomé como referencia el modelo **OSI**, explicado en la Sección 3.3, como guía conceptual para estructurar las distintas capas de comunicación.

Cada dispositivo dentro del simulador actúa como un nodo capaz de enviar y recibir paquetes. Estos paquetes contienen información que simula distintos niveles del modelo OSI, incluyendo direcciones de red, protocolos y datos de aplicación. La comunicación entre dispositivos se realiza a través de un nodo central que simula el comportamiento de un router, y es el encargado de transferir los paquetes, asignar direcciones IP... Además, se incluye una abstracción de conexión entre nodos, lo que permite introducir parámetros como la latencia y el ancho de banda, que son gestionados mediante tiempos de espera.

La arquitectura se ha diseñado siguiendo principios de modularidad y separación de responsabilidades. Cada componente está representado por una clase independiente, lo que facilita su mantenimiento, ampliación y reutilización. La implementación detallada de estos componentes se describe en el Apéndice C de los anexos.

Interfaz gráfica

Una vez finalizada la estructura interna del simulador mediante clases, el siguiente paso fue diseñar una interfaz gráfica que permitiera interactuar con el sistema de forma intuitiva y visual. La interfaz era un aspecto importante, ya que tendría que permitir observar el comportamiento del simulador en tiempo real.

Para que el desarrollo y posterior mantenimiento sea sencillo, se ha implementado un patrón Modelo-Vista-Controlador (MVC), aunque no de forma estricta. Los modelos encapsulan tanto el estado como la lógica de cada entidad de red. La vista, se encarga de la representación gráfica de este modelo y la interacción con el usuario.

Además, existe un controlador global encargado de coordinar la comunicación entre las distintas entidades de la red y gestionar la interacción entre la interfaz gráfica y los modelos. Este controlador actúa como intermediario, permitiendo que la interfaz pueda observar y modificar el estado de la red sin acoplarse directamente a los modelos internos.

Utilidades

Una vez terminada la implementación de la interfaz gráfica, surgía la necesidad de implementar funciones, que sean útiles durante la ejecución de la simulación.

Una de las más relevantes fue la posibilidad de importar una biblioteca externa de comandos, ataques e interceptores definidos por el usuario. Esta funcionalidad permite extender el comportamiento del simulador sin necesidad de modificar su código fuente. Cada una de estas funciones externas recibe como parámetro el dispositivo que las ejecuta. En el caso de los comandos y ataques, también reciben como argumento las direcciones IP de destino. Por otro lado, los interceptores reciben el paquete que han interceptado. Para distinguir el tipo de cada función, es necesario que en el nombre lleve un prefijo que la identifique. Para realizar esta tarea de análisis del script, se han utilizado expresiones regulares, permitiendo obtener el nombre de la función y los parámetros de esta. Para consultar como construir una biblioteca para el simulador, visitar el Apéndice E de los anexos.

Otra funcionalidad importante es la implementación de un gestor de cambios que permita realizar deshacer y rehacer los cambios realizados. Para evitar modificar los métodos existentes, este servicio se implementó utilizando una clase estática encargada de almacenar los estados del simulador. Además, se crearon decoradores con el objetivo de invocar una función que guarde el estado actual cada vez que se ejecute un método anotado. Para evitar almacenar demasiadas instancias del simulador, el guardado se realiza de forma controlada (se guarda un estado si han pasado al menos 500 ms desde el último cambio), además se limita el número de estados guardados.

Predicción con Redes Neuronales

Una de las funcionalidades más importantes del simulador es la integración de modelos de inteligencia artificial para realizar predicciones sobre los flujos de red generados en la simulación. Desde el diseño inicial, se planteó como requisito fundamental ofrecer al usuario la posibilidad de cargar sus propios modelos, permitiendo así una total libertad en cuanto a la elección de la arquitectura de estos.

Estos requisitos dan una serie de problemas que se han intentado solucionar durante el desarrollo. En primer lugar, los modelos suelen ser entrenados en Python utilizando TensorFlow, por lo que se necesita convertirlos a un formato compatible con la biblioteca JavaScript.js. Para ello, se ha empleado la herramienta `tensorflowjs`, que permite la transformación de modelos

entrenados en Python a versiones compatibles para el uso en navegador usando JavaScript.

Otro desafío importante era definir cómo dar la posibilidad al usuario de usar sus modelos durante la captura de paquetes. La solución buscada fue permitir al usuario incluir un script propio que se encargue de ejecutar el modelo, y que implemente una función específica que el simulador pueda invocar para obtener una predicción.

Despliegue para la utilización

Para facilitar el acceso al simulador, he habilitado una dirección web que permite acceder al simulador. El simulador está alojado en GitHub Pages, lo que permite desplegar la versión final del proyecto de forma gratuita y accesible. En concreto, la página sirve el contenido de la rama release del repositorio, donde se encuentra el código ya compilado y preparado para producción.

`https://alejadiez.github.io/iot_attack_detection`

Un aspecto importante a tener en cuenta es la forma en que Angular gestiona las rutas. Por defecto, Angular utiliza un enrutamiento que provoca errores al recargar la página. Para evitar este problema, se optó por utilizar el sistema de rutas mediante hashes (HashLocationStrategy).

5.2. Desarrollo del aprendizaje federado

El objetivo de este desarrollo es la creación de un script que permita desplegar un entorno de entrenamiento distribuido, tanto en clientes como en el servidor. A continuación, se describen las distintas fases de su desarrollo.

Investigación sobre Aprendizaje Federado

Durante esta etapa inicial, se realizó una investigación para comprender en profundidad el concepto de aprendizaje federado y cómo era posible entrenar un modelo en entornos distribuidos. El objetivo principal fue entender cómo en que consiste entrenar un modelo de redes neuronales sin centralizar los datos y cómo, posteriormente, se iban a mezclar todos los modelos generados para obtener un modelo global.

La investigación englobó una multitud de consultas en diversas fuentes, sobre todo en recursos de **Google AI**, que fue una empresa pionera en la

implementación de esta técnica en entornos reales. Se analizaron artículos, documentación oficial, vídeos e implementaciones reales, lo que permitió obtener una visión sobre el funcionamiento de esta novedosa técnica. También, se evaluaron las ventajas y desventajas, tales como la seguridad de los datos, el uso de diferentes datasets, la complejidad de coordinación entre clientes y servidor, la heterogeneidad de los datos (non-IID) ...

Pruebas de entornos federados con datasets simples

Después de la primera fase de investigación sobre el aprendizaje federado, llega el momento de intentar explorar varias bibliotecas que permitan realizar un aprendizaje federado basándose en un modelo de TensorFlow.

La primera opción considerada fue **TensorFlow Federated (TFF)**, una extensión de la biblioteca TensorFlow para Python que permite entrenar un modelo Keras en una arquitectura federada. Tras varias pruebas e intentos, se logró entrenar un modelo básico que consistía en una regresión lineal utilizando una función matemática, siendo un ejemplo muy simple que pretendía ser un acercamiento a un entrenamiento distribuido. Sin embargo, la implementación usando esta biblioteca presentó muchos problemas a la hora de ejecutarse sobre dispositivos con arquitectura ARM, lo que dificultó su uso práctico. Además, TFF está orientada principalmente a fines académicos y de simulación, por lo que no resultó adecuada para entrenamientos reales en múltiples dispositivos.

Teniendo en cuenta las limitaciones encontradas en TFF, se decidió explorar alternativas más adecuadas para nuestro objetivo. Una de las alternativas, mejor valoradas por la comunidad fue Flower, un framework que permite el entrenamiento de modelos de inteligencia artificial sin importar la tecnología usada. Gracias a esto, fue posible el entrenamiento del mismo modelo usado durante la anterior prueba, pero esta vez usando diferentes dispositivos físicos, permitiendo llevar a cabo una prueba real de aprendizaje federado.

Análisis del dataset NF-ToN-IoT y adaptación para entornos federados

Uno de los objetivos principales del proyecto es entrenar un modelo ya definido, que permita detectar ataques en una red, sobre un entorno distribuido. Para ello, se ha usado el dataset **NF-ToN-IoT** [20].

NF-ToN-IoT es un conjunto de datos basados en el conjunto ToN-IoT, creado por la Universidad de Nueva Gales del Sur (UNSW) en Sídney. Fue creado para tener un amplio conjunto de flujos de red que permitiera el entrenamiento en la detección de ataques en entornos de re de dispositivos IoT. Fue desarrollado en el laboratorio de ciberseguridad IoT de UNSW Canberra con una emulación de entornos IoT realistas y complejos.

Este dataset contiene flujo de red procesado y etiquetado, con una proporción de 19.6 % de tráfico benigno y 80.4 % de tráfico malicioso (Tabla 5.1).

Etiqueta	**Distribución**	**Descripción**
Benigno	270.279	Tráfico normal sin actividad maliciosa
Puerta trasera	17.247	Acceso remoto no autorizado mediante programas ocultos
DoS	17.717	Saturación de recursos para interrumpir servicios
DDoS	326.345	DoS desde múltiples fuentes distribuidas
Inyección	468.539	Ingreso de datos maliciosos para alterar la ejecución de programas
Hombre en el medio	1.295	Intercepción de comunicaciones entre dos partes
Contraseñas	156.299	Ataques para obtener contraseñas (fuerza bruta, sniffing)
Ransomware	142	Cifrado de archivos y exigencia de rescate para su recuperación
Escaneo	21.467	Exploración de red para identificar dispositivos y servicios
XSS	99.944	Inyección de scripts maliciosos en páginas web

Tabla 5.1: Distribución del tráfico en el conjunto de datos NF-ToN-IoT

Para el entrenamiento del modelo, se tuvo que extraer las características más representativas y se normalizaron los datos (media cero y varianza unitaria) además de transformar las etiquetas en una nueva etiqueta que indicara si el flujo pertenecía a un ataque o no, logrando así un conjunto de datos apto para una clasificación binaria. Este conjunto de datos se dividió en tres subconjuntos:

- 70 % para entrenamiento
- 15 % para validación
- 15 % para selección de umbral

Dado que el objetivo es entrenar un modelo de forma distribuida, el subconjunto de entrenamiento se tuvo que dividir de manera equitativa en dos subconjuntos, uno para cada cliente, simulando así un dataset para un entorno federado.

Como resultado de todos los pasos descritos, se obtuvieron los siguientes subconjuntos del dataset principal que se puede observar en la Tabla 5.2.

Subconjunto	**Distribución**		**Benignos**	**Ataques**
Entrenamiento 1	35 %	405.297	17,11 %	82,89 %
Entrenamiento 2	35 %	405.298	17,14 %	82,86 %
Validación	15 %	173.699	17,28 %	82,72 %
Umbral	15 %	173.700	17,06 %	82,94 %
Total	**100 %**	**1.157.994**		

Tabla 5.2: Partición del conjunto de datos y distribución entre clientes

Entrenamiento del modelo en entornos federados

Una vez que se tiene el conjunto de datos procesado, se procede a entrenar el modelo. Para ello, primero se debe de disponer de una configuración base del modelo que se debe entrenar. En este caso, se utilizó el modelo ya desarrollado por otros integrantes del Grupo de Inteligencia Computacional Aplicada (GICAP) de la Universidad de Burgos.

El modelo empleado es un **Perceptrón Multicapa (MLP)** con la siguiente arquitectura:

- **Capa de entrada** de 10 neuronas, una neurona por característica analizada:
  - `L4_SRC_PORT`: Puerto de origen a nivel de capa 4.
  - `L4_DST_PORT`: Puerto de destino a nivel de capa 4.

  - `PROTOCOL`: Protocolo de red utilizado (por ejemplo, TCP, UDP).
  - `L7_PROTO`: Protocolo a nivel de la capa de aplicación.
  - `IN_BYTES`: Número de bytes entrantes.
  - `OUT_BYTES`: Número de bytes salientes.
  - `IN_PKTS`: Número de paquetes entrantes.
  - `OUT_PKTS`: Número de paquetes salientes.
  - `TCP_FLAGS`: Indicadores de control TCP codificados en bits.
  - `FLOW_DURATION_MILLISECONDS`: Duración del flujo en milisegundos.

- **Primera capa oculta** con 64 neuronas y función de activación **ReLU**.
- **Segunda capa oculta** con 32 neuronas y función de activación **ReLU**.
- **Capa de salida** con función de activación **Sigmoide**.

Para el entrenamiento, se utilizó el optimizador **Adam** con una tasa de aprendizaje de 0.001. La función de pérdida empleada fue la **entropía cruzada binaria**, y la métrica principal de evaluación fue la **exactitud**.

Con todo esto, se desarrolló un script que permitía lanzar el entorno federado tanto en el lado del servidor como clientes y lo único que había que seleccionar es la IP del servidor para que se pudieran realizar comunicaciones entre ellos.

Con toda la configuración preparada, se desarrolló un script que permitía lanzar el entorno federado, tanto en el servidor como en los clientes. Era necesario obtener la dirección IP del servidor para indicar a los dos clientes el servidor con el que comunicarse.

El flujo de entrenamiento seguido fue el siguiente:

1. Evaluación inicial del modelo en el servidor.
2. El servidor selecciona a los clientes participantes y les envía el modelo global.
3. Cada cliente entrena el modelo localmente durante **4 epochs** y envía el modelo entrenado de vuelta al servidor.

4. El servidor agrega los modelos recibidos y genera un nuevo modelo global.

5. El nuevo modelo se envía a otro subconjunto de clientes para la evaluación[5].

6. Los clientes evalúan el rendimiento del modelo global y devuelven los resultados al servidor. Al mismo tiempo el servidor evalúa también el modelo global.

7. Se repite el proceso desde el paso 2 hasta completar un total de 12 rondas.

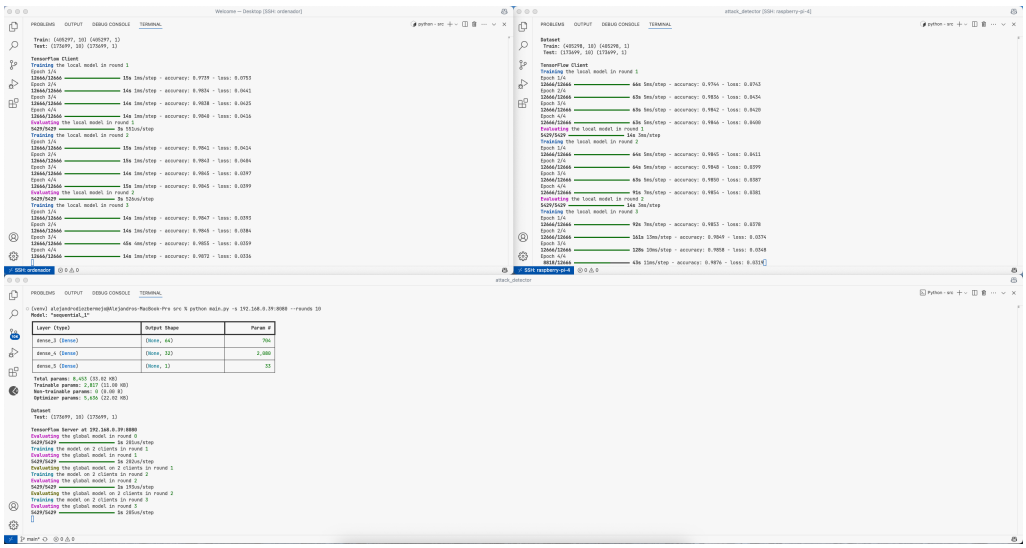

Figura 5.1: Proceso de entrenamiento de forma distribuida

Evaluación del modelo

Una vez finalizado el proceso de entrenamiento, se procede a evaluar el modelo para analizar su desempeño y la capacidad de generalización. Como ya se mencionó en la Sección 3.2, se van a usar diferentes métricas para evaluar la precisión y eficacia del modelo.

Para ello, se va a usas el archivo de métricas que se ha generado durante el entrenamiento. Este archivo contiene todos los resultados obtenidos durante

[5]En este caso no es necesario evaluar el modelo en cada cliente, ya que se usa el mismo conjunto de evaluación.

el entrenamiento y la evaluación del modelo. Además, el mismo script que se ha usado para entrenar el modelo, permite generar gráficas a partir de este archivo, lo que ofrece una manera más visual el comportamiento que ha tenido el modelo durante todas las fases de su evolución.

Esta la siguiente Tabla 5.3 se pueden observar los resultados obtenidos durante el entrenamiento y la evaluación del modelo distribuido.

Entrenamiento	**Cliente 1**	**Cliente 2**
Exactitud	0,9919	0,9918
Pérdida	0,0242	0,0236
Evaluación		**Servidor**
Exactitud		0,9916
Pérdida		0,0244
Precisión		0,9915
Sensibilidad		0,9916
Puntuación F1		0,9915
Área bajo la curva		0,9992

Tabla 5.3: Métricas de rendimiento obtenidas durante el ajuste del modelo

Teniendo como base estas métricas, se observa que el modelo presenta un rendimiento sobresaliente en todas las evaluaciones, lo que indica que hemos logrado obtener un modelo lo suficientemente avanzado como para distinguir con alta precisión flujos maliciosos frente a flujos benignos.

Sin embargo, debido a la naturaleza de este tipo de entrenamiento, surgieron algunos inconvenientes durante el proceso, los cuales se intentarán analizar mediante representaciones gráficas.

Como se puede observar en la Figura 5.2, durante las rondas 6 y 8 se presentan valores inusuales en la evaluación que realiza el servidor en comparación con la pérdida de entrenamiento de cada uno de los clientes. Esto puede deberse a uno de los principales problemas mencionados en la Sección 3.1, relacionado con la heterogeneidad de los datos. Esta heterogeneidad puede provocar que, en determinadas rondas, el modelo colapse parcialmente durante el entrenamiento, generando estos desajustes. No obstante, en las rondas posteriores el comportamiento se estabiliza, permitiendo alcanzar un modelo con mayor convergencia.

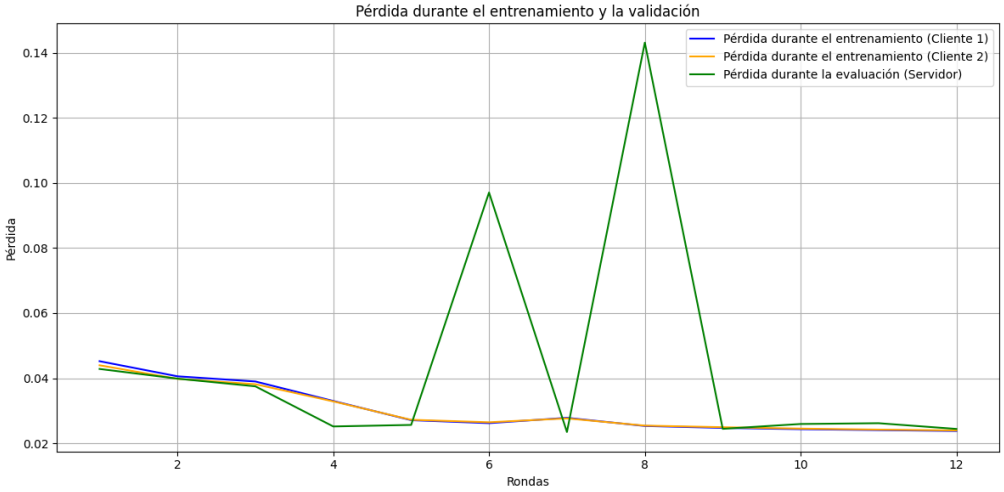

Figura 5.2: Pérdida durante el entrenamiento y la validación

Para visualizar mejor el comportamiento del modelo, en la Figura 5.3 se muestra la matriz de confusión, la cual muestra que el modelo es capaz de predecir correctamente la mayoría de los casos. Cabe destacar que el modelo tiende a detectar con mayor precisión los ataques frente a los flujos benignos, esto podría deberse a un desbalanceo en el conjunto de datos, donde podría ser necesario introducir una mayor cantidad de muestras de flujos benignos para mejorar la representatividad y el equilibrio entre clases.

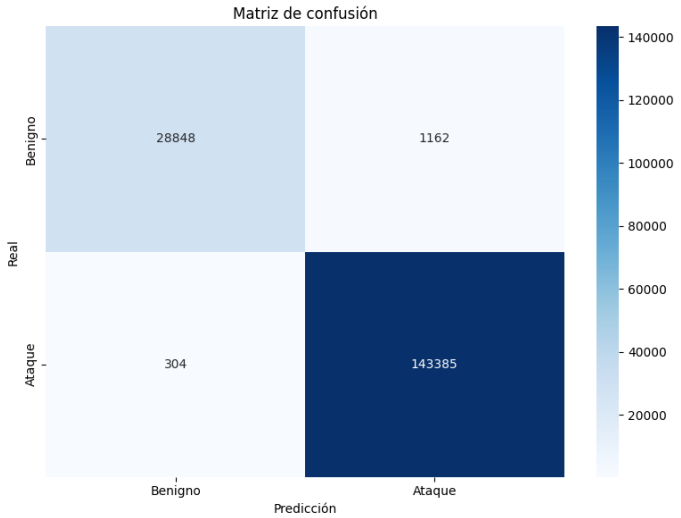

Figura 5.3: Matriz de confusión del modelo

Pruebas del modelo

Con el fin de validar el modelo[6] en un entorno más próximo al uso real, se usó el Simulador IoT desarrollado en este mismo proyecto para simular flujos de paquetes benignos y ataques.

Para ello, se realizó un script en JavaScript que permitiera adaptar el modelo al simulador, analizando cada uno de los flujos que se transmitían de un nodo a otro. Este script mantiene un registro de cada uno de los tráficos activos y, cuando alguno de los tráficos permanece cerrado durante 8 segundos, se cierra y se extraen sus características principales.

Cada flujo cerrado se transforma en un conjunto de características numéricas, las cuales se normalizan utilizando la media y desviación estándar utilizadas durante la fase de entrenamiento. Estas características incluyen:

- Puertos de origen y destino
- Protocolo de transporte
- Tamaño de paquetes transmitidos durante el flujo
- Número de paquetes transmitidos durante el flujo
- Banderas TCP
- Duración total del flujo

Una vez realizada la transformación de los datos, se introducen al modelo para que genere una predicción en el rango $[0, 1]$. Si el resultado es superior a 0.5, este flujo se clasifica como ataque y en caso contrario se considera como flujo benigno.

[6]Es necesario convertir el modelo generado en TensorFlow Python usando la herramienta tensorflowjs para que sea compatible con la biblioteca TensorFlow.js

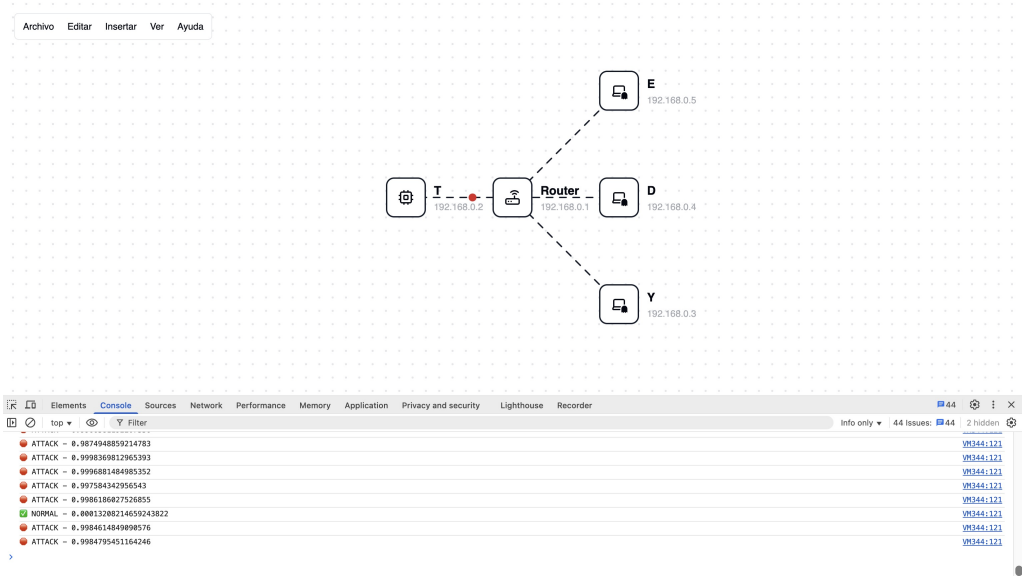

Figura 5.4: Ataque DDoS en el Simulador IoT

Para realizar las pruebas fue necesario generar varias simulaciones de flujo de datos y combinarlas entre sí para que el modelo pueda detectar los ataques.

Ataque DoS

Simula el envío masivo de paquetes TCP con la bandera SYN, simulando un SYN flood. Para realizar este ataque se envía una cantidad de paquetes TCP con la bandera SYN activada, con puertos de origen y valores de TTL aleatorios. También se generó un ataque con paquetes UDP, para comprobar si cambiaba en algo. Si lo que se quiere es simular un ataque DDoS, se lanzara el mismo ataque desde diferentes dispositivos simultáneamente.

Ping

Se intenta simular la transmisión de un comando ping. Consiste en enviar un paquete ICMP, con código 8 (Echo Request). Después el otro dispositivo le responde con un paquete ICMP y código 0 (Echo Reply).

Three-Way Handshake

Se intenta simular el inicio de una conexión confiable entre dispositivos mediante el protocolo TCP. Se envía un paquete TCP con la bandera SYN.

Después, el otro dispositivo le responde con un paquete TCP con las banderas SYN y ACK. Finalmente, se responde con un paquete TCP con la bandera ACK, confirmando así la conexión.

Video Streaming

Se intenta simular la transmisión de un video desde un servidor que envía datos de forma continua. Consiste en enviar durante un tiempo definido, paquetes cada cierto tiempo con un contenido aleatorio en un rango de $[512, 2056]$ caracteres. Este tráfico simula un flujo de datos benigno para compartir un vídeo en tiempo real, como se haría durante una videollamada. Se usa un paquete TCP, con bandera AKC y se envía hacia el puerto de destino 443 y desde un puerto aleatorio.

6. Trabajos relacionados

Como se comentó en la introducción, la detección de ataques en redes de dispositivos es una tarea de gran relevancia en la sociedad. Cada vez más dispositivos se conectan a la red y, con ello, aumenta su vulnerabilidad frente a posibles ataques. En este contexto, surgen múltiples alternativas para hacer frente a estos problemas. Durante el desarrollo de este proyecto, se ha propuesto una estrategia para paliar esta situación.

En este apartado se exponen los distintos artículos y recursos que más han influenciado en el desarrollo del proyecto, así como el artículo académico que se escribió a partir de la investigación realizada sobre el aprendizaje federado.

- **Communication-Efficient Learning of Deep Networks from Decentralized Data**: artículo que introduce el concepto de Federated Learning, además de analizar sus ventajas y desventajas. Este trabajo fue una primera toma de contacto con las bases teóricas del aprendizaje federado. El desarrollo de este proyecto se ha basado en este artículo, aplicándolo a un área más específica, como es la detección de ataques en redes. (Fuente [19])
- **TensorFlow Federated Tutorial Session**: conferencia que muestra el concepto de aprendizaje federado y su aplicación con la biblioteca TensorFlow Federated. Este recurso fue clave en los primeros pasos de implementación de un entorno federado, aunque después no se haya usado en la realización del TFG. (Fuente [24])
- **Listado de herramientas para aprendizaje federado**: blog que compara las distintas alternativas que existen en el mercado de código

abierto para la implementación de aprendizaje federado. Permitió evaluar qué tecnología era la más flexible para el desarrollo del proyecto. (Fuente [11])

- **Documentación TensorFlow**: documentación oficial de la biblioteca TensorFlow. Es la Fuente principal de ayuda en la que me he apoyado en los momentos donde han surgido problemas con el uso de la biblioteca. (Fuente [7])

- **Documentación TensorFlowJS**: documentación oficial de la biblioteca TensorFlowJS. Esta documentación me ha ayudado a la implementación de los modelos de inteligencia artificial en el simulador web y a la solución de los problemas que he tenido durante la implementación. (Fuente [9])

- **Documentación TensorFlow Federated**: documentación oficial de la biblioteca TensorFlow Federated. Fue la base para la implementación de las primeras pruebas realizadas sobre una arquitectura de entrenamiento federado. (Fuente [8])

- **Documentación Flower**: documentación oficial de la biblioteca Flower. Me sirvió de gran ayuda en el desarrollo del script realizado para la implementación de un aprendizaje federado sobre un caso concreto. Tiene muchos ejemplos usando diferentes bibliotecas de entrenamiento de redes neuronales, entre las que se encuentra TensorFlow. (Fuente [4])

- **Curso de Angular**: curso sobre desarrollo de aplicaciones web con el framework Angular. Aunque el curso incluye la creación de aplicaciones web, que poco tienen que ver con el simulador realizado durante el proyecto, ha sido de gran ayuda para comprender las nociones básicas del framework, así como las buenas prácticas de desarrollo. (Fuente [1])

Federated Learning for the Detection of Attacks on IoT Devices

Durante mi participación en la beca de colaboración dentro de un grupo de investigación universitario, desarrollé un trabajo centrado en el estudio y aplicación de técnicas de aprendizaje automático para la detección de ataques sobre redes de dispositivos IoT. El objetivo principal del proyecto fue adaptar y entrenar un modelo ya existente utilizando un enfoque distribuido basado en aprendizaje federado.

Como resultado de este trabajo, se elaboró un artículo académico cuyo propósito fue introducir el concepto de aprendizaje federado y comparar su eficacia con la de un modelo entrenado de forma tradicional. La investigación se centró en evaluar el rendimiento de ambos modelos mediante diversas métricas, obteniendo resultados bastante buenos, especialmente teniendo en cuenta los desafíos del uso de técnicas federadas.

Para dar visibilidad al trabajo realizado, el artículo fue enviado al congreso internacional CISIS 2025 (International Conference on Computational Intelligence in Security for Information Systems), encontrándose actualmente en proceso de revisión.

7. Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Durante el periodo que ha durado este proyecto, se ha desarrollado un simulador de rede de dispositivos IoT, con objetivo de ser un espacio de pruebas para distintos modelos de redes neuronales destinados a la detección de ataques. Además, se ha logrado el entrenamiento de un modelo de inteligencia artificial de forma distribuida, priorizando en todo momento la privacidad de los datos.

Se puede concluir que el objetivo general del proyecto se ha cumplido de manera satisfactoria. Se ha logrado desarrollar un simulador básico de red de dispositivos IoT, capaz de generar un flujo de paquetes y, mediante la implementación de un modelo de red neuronal, detectar estos ataques simulados.

En cuanto al entrenamiento de la red neuronal para la detección de ataques, aún queda un largo camino por recorrer hasta que el modelo sea plenamente funcional y efectivo. La complejidad está en la obtención de datos de calidad, dificultando el aprendizaje. Por ello, se requieren más iteraciones para mejorar su precisión y capacidad de generalización.

Una vez finalizado el proyecto, es importante destacar algunos puntos clave que he tenido en cuenta:

- Tener una base teórica en los temas tratados durante el proyecto (artículos de investigación y conocimientos del grado).

- Tener claro que ofrece cada tecnología que he usado, para ahorrar tiempo durante el desarrollo.
- Una buena planificación del trabajo, con los objetivos claros y materiales necesarios.

Gracias a la realización de este proyecto, me ha permitido profundizar en diversas áreas del ámbito de la inteligencia artificial y el desarrollo de software. Para ser más concretos, he tenido la oportunidad de investigar y aplicar técnicas avanzadas de entrenamiento de redes neuronales, como el aprendizaje federado. Esta técnica me ha permitido comprender mejor la importancia de tener un buen conjunto de datos y como este es capaz de influir sobre la calidad del modelo resultante.

Asimismo, el proyecto ha sido un entorno perfecto para mejorar mis habilidades en el desarrollo de software, consolidando conocimientos sobre las herramientas que he usado.

Por otro lado, gestionar un proyecto de esta magnitud me ha llevado a mejorar mis habilidades organizativas. He aprendido a planificar tareas, establecer prioridades, gestionar tiempos y adaptarme a imprevistos que he logrado solucionar buscando alternativas.

7.2. Líneas futuras

A pesar de los logros alcanzados por este proyecto, todavía quedan varios aspectos a mejorar para obtener un proyecto más consolidado.

- **Topologías de red más personalizadas**: Una posible línea de mejora del simulador consiste en ampliar su capacidad para representar topologías de red más complejas y personalizadas. Esto podría incluir la incorporación de múltiples subredes interconectadas y nuevos dispositivos de gestión, como switches o puntos de acceso. Además, la integración de inteligencia artificial generativa podría permitir la generación automática de redes complejas.
- **Mayor realismo en la red**: otra posible mejora para el simulador consistiría en la implementación de más protocolos o aspectos que permitan tener un entorno de red más realista.
- **Posibilidad de introducir parámetros a funciones externas**: una mejora que facilitaría la prueba del simulador sería la posibilidad

de introducir parámetros adicionales a las funciones externas creadas por el usuario.

- **Actualización del dataset**: una mejora significativa para incrementar el rendimiento del modelo consistiría en entrenarlo con conjuntos de datos más amplios y representativos. Se podrían emplear versiones más recientes del dataset utilizado en este proyecto.
- **Exploración de diferentes arquitecturas de redes neuronales**: otro aspecto a investigar sería el uso de diferentes arquitecturas de redes neuronales, para poder comparar entre diferentes alternativas, cuál sería la más adecuada para este proyecto.
- **Entorno de entrenamiento más realista**: para avanzar en la investigación del entrenamiento distribuido, una posible línea de trabajo futura consistiría en simular un entorno más cercano a escenarios reales. Esto implicaría entrenar el modelo con un mayor número de clientes, distribuidos en varias subredes. Esto permitiría evaluar el comportamiento del sistema en condiciones de mayor escala y diferencias. Además, sería interesante estudiar el impacto del uso de datos no independientes e idénticamente distribuidos en el rendimiento del modelo. Por último, también se podría analizar las implicaciones que lleva el entrenamiento del modelo en la propia red, en consumo de ancho de banda, latencia o congestión.

Bibliografía

[1] Angular moderno. https://cursos.devtalles.com/courses/angular-moderno. [Último acceso 1 de Junio de 2025].

[2] Classification: Accuracy, recall, precision, and related metrics | Machine Learning | Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall. [Último acceso 9 de Mayo de 2025].

[3] Classification: ROC and AUC | Machine Learning | Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. [Último acceso 9 de Mayo de 2025].

[4] Flower. https://flower.ai/docs/. [Último acceso 14 de Marzo de 2025].

[5] Gestión de recursos de red mediante flujos. https://docs.oracle.com/cd/E56339_01/html/E53790/gitiz.html#scrolltoc. [Último acceso 5 de Junio de 2025].

[6] Linear regression: Loss | Machine Learning | Google for Developers. https://developers.google.com/machine-learning/crash-course/linear-regression/loss. [Último acceso 9 de Mayo de 2025].

[7] Tensorflow. https://www.tensorflow.org/api_docs/python/tf. [Último acceso 12 de Marzo de 2025].

[8] TensorFlow Federated. https://www.tensorflow.org/federated/api_docs/python/tff. [Último acceso 12 de Marzo de 2025].

[9] Tensorflow.js. https://js.tensorflow.org/api/latest/. [Último acceso 28 de Abril de 2025].

[10] Thresholds and the confusion matrix | Machine Learning | Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/thresholding. [Último acceso 9 de Mayo de 2025].

[11] Top 7 Open-Source Frameworks for Federated Learning. https://www.apheris.com/resources/blog/top-7-open-source-frameworks-for-federated-learning, September 2024. [Último acceso 4 de Febrero de 2025].

[12] Carolina Bento. Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. https://medium.com/data-science/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141, September 2021. [Último acceso 19 de Mayo de 2025].

[13] Cloudflare. Ataque de denegación de servicio. https://www.cloudflare.com/es-es/learning/ddos/glossary/denial-of-service/. [Último acceso 20 de Mayo de 2025].

[14] Cloudflare. ¿qué es el modelo osi? https://www.cloudflare.com/es-es/learning/ddos/glossary/open-systems-interconnection-model-osi/. [Último acceso 20 de Mayo de 2025].

[15] Fortinet. ¿qué es un ciberataque y los tipos de ataques en la red? https://www.fortinet.com/lat/resources/cyberglossary/types-of-cyber-attacks.html. [Último acceso 20 de Mayo de 2025].

[16] GICAP. Grupo de inteligencia computacional aplicada. https://gicap.ubu.es/main/home.shtml, 2008. [Último acceso 6 de Mayo de 2025].

[17] IBM. ¿qué es el aprendizaje supervisado? https://www.ibm.com/es-es/topics/supervised-learning, December 2024. [Último acceso 14 de Mayo de 2025].

[18] Kelvin. Introduction to federated learning and challenges. https://towardsdatascience.com/introduction-to-federated-learning-and-challenges-ea7e02f260ca/, October 2020. [Último acceso 28 de Enero de 2025].

[19] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. http://arxiv.org/abs/1602.05629, January 2023. [Último acceso 12 de Abril de 2025].

[20] The University of Queensland, Australia Brisbane St Lucia, QLD 4072 +61 7 3365 1111 Other Campuses: UQ Ipswich, U. Q. Gatton, UQ Herston Maps, and Directions © 2013 The University of Queensland. Ml-based nids datasets. https://www.itee.uq.edu.au/research/cyber-security/research-areas. [Último acceso 25 de Abril de 2025].

[21] Mohanad Sarhan, Siamak Layeghy, Nour Moustafa, and Marius Portmann. Netflow datasets for machine learning-based network intrusion detection systems. In Zeng Deze, Huan Huang, Rui Hou, Seungmin Rho, and Naveen Chilamkurti, editors, *Big Data Technologies and Applications*, pages 117–135, Cham, 2021. Springer International Publishing.

[22] Google Cloud Tech. ¿qué es el aprendizaje automático? tipos y usos. https://cloud.google.com/learn/what-is-machine-learning. [Último acceso 14 de Mayo de 2025].

[23] Google Cloud Tech. What is federated learning? https://www.youtube.com/watch?v=X8YYWunttOY, February 2021. [Último acceso 19 de Febrero de 2025].

[24] Google TechTalks. Tensorflow federated tutorial session. https://www.youtube.com/watch?v=JBNas6Yd30A. [Último acceso 24 de Febrero de 2025].

[25] Vectorslab. Iconos en Flaticon. https://www.flaticon.es/autores/vectorslab.