Universidad Nacional de Ingeniería

Facultad de Ingeniería Eléctrica y Electrónica

TESIS

Predicción del tráfico de datos de red en una universidad estatal utilizando modelos de series temporales: ARIMA, LSTM Y Prophet

Para obtener el Título Profesional de Ingeniero de Telecomunicaciones.

Elaborado por

Geanfranco Agustín Solier Riveros

0009-0005-4150-1973

Asesor

Mag. Ing. Victor Andrés Córdova Bernuy

0000-0002-3847-1698

LIMA – PERÚ

2025

Citar/How to cite	Solier Riveros [1]
Referencia/Reference Estilo/Style: IEEE (2020)	[1] G. Solier Riveros, *"Predicción del tráfico de datos de red en una universidad estatal utilizando modelos de series temporales: ARIMA, LSTM y PROPHET*" [Tesis]. Lima (Perú): Universidad Nacional de Ingeniería, 2025.

Citar/How to cite	(Solier, 2025)
Referencia/Reference Estilo/Style: APA (7ma ed.)	Solier, G. (2025). *Predicción del tráfico de datos de red en una universidad estatal utilizando modelos de series temporales: ARIMA, LSTM y PROPHET*. [Tesis, Universidad Nacional de Ingeniería]. Repositorio institucional Cybertesis UNI.

Dedicatoria

La presente tesis se la dedico, en especial, a mis queridos padres Agustín Solier Guevara y Haydeé Rosario Riveros Farfán, quienes han sido mi fuerza, mi inspiración y mi guía en cada etapa de mi vida. Gracias por su amor incondicional, por sus sacrificios y por enseñarme el valor del esfuerzo y la perseverancia.

A mis adorados abuelitos, que con su sabiduría, cariño y enseñanzas marcaron mi corazón y me mostraron que los sueños se alcanzan con dedicación y humildad.

Gracias por ayudarme a cumplir mis objetivos, tanto como persona y como estudiante. Este logro es el reflejo de todo lo que me han brindado, y se los dedico con todo mi amor y gratitud.

Agradecimientos

El principal agradecimiento es a Dios, quien me ha guiado y me ha dado la fortaleza para seguir adelante. Sin su luz y bendiciones, este logro no habría sido posible.

A mi querida familia, por su comprensión, estímulo constante y apoyo incondicional a lo largo de mis estudios.

A mis hermanos, por siempre alentarme, estar a mi lado cuando los necesitaba, nunca dejarme rendir y guiarme con sus enseñanzas, que han sido fundamentales para mi crecimiento personal y académico.

A mi novia, por su comprensión, paciencia y por ser una fuente constante de motivación y alegría durante este proceso.

Expreso mi sincero agradecimiento al Mg. Ing. Víctor Andrés Córdova Bernuy, por su guía, conocimiento y valiosos consejos que hicieron posible la culminación de este trabajo. Su apoyo fue clave para el desarrollo de esta tesis.

También agradezco de manera sincera a los excelentes docentes que estuvieron guiándome y dándome enseñanzas para la elaboración de la presente tesis.

Finalmente, a todas las personas que, de una u otra forma, me apoyaron en la realización de este trabajo, les expreso mi más profundo agradecimiento.

Resumen

La investigación, titulada "Predicción del tráfico de datos de red en una universidad estatal utilizando modelos de series temporales: ARIMA, LSTM y Prophet", tiene como objetivo desarrollar un modelo predictivo para optimizar la gestión del tráfico de datos en la red de la Universidad Nacional de Ingeniería. Este estudio responde al incremento del uso de tecnologías digitales en actividades académicas y administrativas, lo que ha generado problemas de latencia, congestión y caídas del servicio debido a una planificación ineficiente del ancho de banda.

El enfoque metodológico consistió en recopilar datos históricos de tráfico y dividirlos en conjuntos de entrenamiento, validación y prueba. Se implementaron y ajustaron los parámetros de los modelos ARIMA, LSTM y Prophet, considerando las características particulares del tráfico universitario. La evaluación del desempeño se realizó utilizando métricas como MAE, RMSE y MAPE, lo que permitió comparar la precisión y efectividad de cada modelo.

Los resultados demostraron que LSTM fue el modelo más preciso, logrando un MAE de 0.09% en OutTraffic y 0.17% en TotalTraffic, además de una precisión categórica del 92%. ARIMA mostró un desempeño moderado en datos estacionarios, con errores entre 13.37% y 18.20%, mientras que Prophet presentó errores significativamente altos, lo que evidenció su limitada capacidad para modelar el tráfico de red con alta variabilidad.

En conclusión, LSTM demostró ser el modelo más adecuado para la predicción del tráfico de red, permitiendo una planificación más eficiente del ancho de banda y una reducción significativa de la latencia. Esta investigación representa una contribución clave para la gestión proactiva de redes universitarias, recomendándose su integración con sistemas de monitoreo en tiempo real y su aplicación en otras instituciones educativas con desafíos similares.

Palabras clave: predicción de tráfico, ARIMA, LSTM, Prophet, modelos de series temporales, gestión de red.

Abstract

The research, titled "Network Traffic Prediction in a Public University Using Time Series Models: ARIMA, LSTM, and Prophet", aims to develop a predictive model to optimize network traffic management at the National University of Engineering. This study addresses the increasing use of digital technologies in academic and administrative activities, which has led to issues such as latency, congestion, and service disruptions due to inefficient bandwidth planning.

The methodological approach involved collecting historical traffic data and splitting it into training, validation, and testing datasets. The ARIMA, LSTM, and Prophet models were implemented and fine-tuned based on the unique characteristics of university network traffic. Their performance was evaluated using MAE, RMSE, and MAPE, allowing for a comparative analysis of their prediction accuracy and efficiency.

The results indicated that LSTM was the most accurate model, achieving a MAE of 0.09% in OutTraffic and 0.17% in TotalTraffic, with a categorical accuracy of 92%. ARIMA exhibited moderate performance in stationary data, with errors ranging from 13.37% to 18.20%, whereas Prophet showed significantly high errors, highlighting its limited capability to model network traffic with high variability.

In conclusion, LSTM proved to be the most suitable model for network traffic prediction, enabling more efficient bandwidth planning and a significant reduction in latency. This research makes a valuable contribution to proactive network management in academic environments, recommending its integration with real-time monitoring systems and expansion to other educational institutions facing similar challenges.

Keywords: traffic prediction, ARIMA, LSTM, Prophet, time series models, network management.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Introducción

Hoy en día, las redes de datos cumplen un papel fundamental en el desarrollo de trabajos que van más allá de lo académico, tales como los trabajos administrativos y de investigación en las universidades. Las tecnologías digitales acrecentaron una mayor demanda de conexión estable y eficiente, por lo que el aumento del tráfico de la red es inevitable. La gestión de este tráfico es un problema que concierne a todos recurren a la Universidad como un espacio donde trabajar y experimentar académicamente. Esta no es excepción para nadie. Si se espera que el servicio de la red tenga calidad, es necesario que el ancho de banda esté disponible y sea aprovechado por todos los estudiantes, docentes y administrativos que alberga la universidad.

La Universidad Nacional de Ingeniería es una prestigiosa institución de educación superior con un enfoque tecnológico que encara tales retos en un entorno en donde la digitalización ha apresurado el paso, además su influencia se repite en el de todas las áreas. La adopción de más herramientas digitales ha incrementado el tráfico en la red. Esto repercute en los ámbitos académico, administrativo e incluso de investigación. De este modo ante cada problema planteado, se requerirá la presentación de una solución innovadora para así mitigar la congestión y la latencia, además de anticiparse a todas las necesidades futuras de tecnología.

En este marco, la presente investigación, titulada "Predicción del tráfico de datos de red en una universidad estatal utilizando modelos de series temporales: ARIMA, LSTM y Prophet", plantea un análisis estricto con el empleo de series temporales en modelos avanzados. Se han implementado unos tres enfoques predictivos clave, además de ser comparados. Específicamente son los modelos ARIMA, LSTM y Prophet. ARIMA es reconocido por su habilidad para representar tendencias lineales además de estacionales; LSTM, es destacado por capturar toda clase de patrones complejos junto con relaciones de muy larga duración; por otro lado, Prophet, es diseñado para identificar en los datos sus tendencias con alta variabilidad además de verificar la estacionalidad.

El principal objetivo de la presente investigación es aumentar la precisión al pronosticar el flujo de información en la red de la Universidad Nacional de Ingeniería, proveyendo herramientas analíticas que hagan posible mejorar la planificación del ancho de banda, así como prever aumentos de tráfico de forma eficaz. Para lograr esto, se ha desarrollado un marco metodológico bastante detallado. Dicho marco permite la selección, ajuste y evaluación de modelos para analizar el comportamiento de series temporales en los entornos universitarios, así como optimizar la gestión del ancho de banda de una manera mucho más eficiente.

Los resultados de esta investigación establecen normas técnicas que permiten optimizar las estructuras de telecomunicaciones con respecto a la administración del ancho de banda, impulsando la aplicación de tácticas de monitoreo predictivo para disminuir la saturación, consolidar la resiliencia de la red, y también aumentar la calidad de planificación operativa. Además, este estudio aporta cierto marco de referencia fundamental. Tiene lugar en el campo de la modelización predictiva aplicada a muchas redes de datos dentro de entornos académicos, proveyendo una base teórica, así como práctica para la integración hacia estrategias avanzadas sobre gestión del tráfico de red.

Este trabajo representa una contribución significativa para la optimización de infraestructuras de telecomunicaciones, así como para la gestión eficiente del tráfico de datos en entornos universitarios. Se valora el respaldo de la institución en cuanto a proporcionar los medios, data y equipos técnicos, lo cual facilitó que la investigación se realizara de manera exhaustiva y con el rigor metodológico requerido. Los resultados conseguidos dan un marco útil para investigaciones futuras en el campo de la modelización predictiva y administración de redes con respecto al ancho de banda.

Capítulo I. Parte introductoria del trabajo

1.1 Generalidades

La predicción, así como evaluación del tráfico de red es un elemento esencial en la gestión de infraestructuras de redes actuales, en especial en sitios de alta densidad poblacional como son las universidades. Muchas aplicaciones permiten anticipar el flujo de datos dentro de una red, asegurando de esta manera la asignación óptima de los recursos. De esta manera se está reduciendo de manera significativa la latencia y previniendo toda congestión. Tradicionalmente, la gestión de red un enfoque reactivo, esto hace que solo se dé solución a los problemas solo en el momento en que suceden. No obstante, a medida que el tráfico de datos de red experimenta un rápido crecimiento en el número de dispositivos, aplicaciones y usuarios conectado, las soluciones que anticipan problemas y se ajustan rápidamente se convierten en indispensables, con el propósito de mantener la eficiencia y la escalabilidad de la red.

Es así como esta investigación titulada “Predicción del tráfico de datos de red en una universidad estatal utilizando modelos de series temporales: ARIMA, LSTM y Prophet,” plantea esta problemática aplicando el análisis, implementación y la evaluación de tres modelos de series temporales. Los modelos mencionados permitirán analizar patrones históricos de tráfico y predecir comportamientos futuros. De esta manera se optimizará la gestión de recursos de red.

Estos modelos se definen:

- ARIMA (Promedios Móviles Integrados Autorregresivos): Es un modelo que maneja los datos lineales y aquellos que tienen patrones estacionales definidos.
- LSTM (Redes Neuronales de Memoria a Largo y Corto Plazo): Este modelo captura relaciones complejas y relaciones no lineales de secuencias temporales.
- Prophet: Este modelo maneja cambios abruptos en las series temporales, analiza tendencias y estacionalidades.

Debido a la creciente demanda de conectividad en red de la Universidad Nacional de Ingeniería (UNI) se requiere de soluciones que ayuden a predecir dichas demandas. En este contexto se gestiona el tráfico de grandes volúmenes de datos, estos son originados por las actividades académicas, administrativas, plataformas de aprendizaje en línea, sistemas de gestión institucional y herramientas de investigación. De esta manera el desarrollo de modelos predictivos permite el análisis de patrones de consumo, periodos, picos de tráfico y la detección de anomalías. Todo esto contribuirá a reducir la congestión, mejorar la experiencia digital, y la satisfacción general del usuario final.

En el presente estudio se hará uso de los modelos ARIMA, LSTM y Prophet los cuales serán comparados tanto en técnicas tradicionales como en avanzadas para predecir el tráfico de red en el campus universitario. Estos resultados permiten mejorar el rendimiento de la red universitaria y proporcionar un marco practico de referencia para otras instituciones que enfrentan los mismos desafíos.

En Perú, las instituciones de educación superior empiezan a adoptar soluciones basadas en el análisis avanzado de datos. Factores como limites presupuestales, carencia de personal capacitado y la privacidad en los datos dificultan la implementación de modelos predictivos. En contraste a ello, el enorme crecimiento de herramientas de inteligencia artificial permite a las instituciones de educación superior adoptar estas soluciones en beneficio de una mejor gestión de la red.

Esta investigación propone un modelo practico y escalable con la finalidad de satisfacer dichas demandas, tanto en el ámbito técnico y de contexto. De esta manera se busca además modernizar la infraestructura digital en las universidades del Perú, gestionando proactivamente la red universitaria, en contraste de solo reaccionar a los problemas luego de que estos suceden.

La presente investigación se divide en seis capítulos. El Primer Capítulo describe el problema, los objetivos de la investigación y las hipótesis rectoras. El Segundo Capítulo presenta todos los fundamentos teóricos y estudios previos sobre técnicas predictivas que apoyan la parte experimental. El Tercer Capítulo explica los métodos de investigación, este

capítulo incluye la preparación de datos y la configuración experimental. El Cuarto Capítulo analiza y discute los resultados obtenidos en la implementación y evaluación de los modelos predictivos. El Quinto Capítulo brinda un resumen de las conclusiones y propone líneas de investigación que pueden ser abordadas en un futuro. Finalmente, el Sexto Capitulo nos brinda recomendaciones para gestionar eficazmente las redes universitarias.

En líneas generales, el presente estudio contribuye de manera significativa a la integración de soluciones tecnológicas avanzadas en las redes universitarias, afrontando los desafíos en el marco de la transformación digital.

1.2 Descripción del problema de investigación

En años recientes debido al crecimiento de la tecnología digital que es usada en ámbitos de la docencia, investigación y administración de universidades buscan mejorar la gestión eficiente de su tráfico de red, siendo esta aún más relevante. Instituciones de educación superior como la Universidad Nacional de Ingeniería enfrentan continuamente retos que dificultan una gestión de manera eficiente el ancho de banda para que estudiantes, profesores y personal tengan acceso fiable tanto a las plataformas en línea como a los recursos digitales. La gran cantidad de datos generados diariamente por miles de usuarios conectados en simultaneo con la red hace que predecir el ancho de banda sea incierto y esto se incrementa debido al uso de aulas en línea, videoconferencias, bases de datos y herramientas de investigación especializadas.

Los responsables de TI de las universidades pese a los numerosos esfuerzos por mejorar la infraestructura de red ven difícil predecir los picos de tráfico y gestionar los recursos de forma eficaz. A menudo la red presenta baja calidad, así como demoras e interrupciones para los usuarios lo que impacta de forma negativa en lo académico y lo administrativo. De este modo el rápido avance del aprendizaje híbrido y a distancia genera en la demanda de conexión de red un gran aumento y saturación.

La forma actual de gestionar la red en las universidades es reactiva, de esta manera solo se actúa cuando surgen inconvenientes. Pese a ello este método resulta insuficiente para abordar los problemas de congestión y asignación optima de recursos de forma

proactiva. Por consiguiente, resulta esencial planificar y ejecutar adecuadamente modelos predictivos capaces de pronosticar el tráfico de red y de esta manera los administradores de red podrán anticipar y resolver problemas incluso antes de que estos afecten a los usuarios.

La presente investigación enfrenta estos desafíos mediante el uso de modelos predictivos de series temporales tales como ARIMA, LSTM y Prophet. Estos enfoques procesan datos históricos de tráfico para anticipar futuras tendencias lo que facilita a los administradores de red una asignación proactiva y eficiente del ancho de banda. No obstante llevar estos modelos a escenarios reales implica ciertos retos como manejar la variabilidad del tráfico, asegurar la escalabilidad ante el aumento de la demanda y proteger la privacidad y seguridad de los datos.

La ausencia de métodos predictivos avanzados para analizar el tráfico de red en entornos universitarios junto con la creciente necesidad de proteger la integridad y privacidad de los datos evidencia la urgencia de adoptar modelos predictivos sustentados en técnicas de series temporales. Estas soluciones favorecen una gestión de red más proactiva y eficiente permitiendo la asignación dinámica de recursos y de esta manera mejorar el rendimiento y adaptando la red a las cambiantes exigencias de conectividad.

1.2.1 Problema general

1. ¿Cuál es el nivel de precisión que se puede alcanzar en la predicción del tráfico de datos en la red de la Universidad Nacional de Ingeniería utilizando modelos de series temporales como ARIMA, LSTM y Prophet, y cómo estos modelos contribuyen a la planificación de ancho de banda en la red universitaria?

1.2.2 Problemas Específicos

1. ¿Cuáles son los patrones de tráfico predominantes en la red de la Universidad Nacional de Ingeniería y cómo pueden utilizarse para estimar la demanda futura de ancho de banda mediante modelos predictivos basados en series temporales?

2. ¿Cuál de los modelos de series temporales (ARIMA, LSTM o Prophet) ofrece el mejor desempeño en términos precisión y error de predicción (MAE, RMSE, MAPE) para la estimación del tráfico de la red universitaria?
3. ¿En qué medida la implementación de un modelo predictivo basado en series temporales permite mejorar la detección de picos de tráfico y optimizar la planificación del ancho de banda en la red universitaria?

1.2.3 Variables

1. Variable Dependiente: Predicción del tráfico de datos de red.
2. Variable Independiente: Modelos predictivos y sus configuraciones (ARIMA, LSTM, Prophet).

1.3 Objetivos

Los objetivos de esta tesis se agrupan en dos categorías, las cuales se describen a continuación.

1.3.1 Objetivo general

Desarrollar y evaluar modelos predictivos basados en series temporales (ARIMA, LSTM y Prophet) para la predicción del tráfico de datos en la red de la Universidad Nacional de Ingeniería, con el fin de mejorar la precisión en la estimación del tráfico y optimizar la planificación del ancho de banda.

1.3.2 Objetivos específicos

1. Analizar y caracterizar los patrones de tráfico en la red universitaria, identificando las tendencias y fluctuaciones que permitan estimar la demanda futura de ancho de banda mediante modelos de series temporales.
2. Comparar el desempeño de los modelos ARIMA, LSTM y Prophet en términos de error de predicción (MAE, RMSE, MAPE) y determinar cuál de ellos es el más adecuado para la estimación del tráfico en la red universitaria.
3. Implementar y validar el modelo predictivo más preciso y eficiente, evaluando su precisión en la estimación del tráfico de datos y su capacidad para anticipar la demanda futura de ancho de banda.

1.3.3 Indicadores de logros de Objetivos

Tabla 1

Indicadores de Logro de los Objetivos

Objetivo Especifico	Indicador de Logro	Métrica
Analizar y caracterizar los patrones de tráfico en la red universitaria para estimar la demanda futura de ancho de banda mediante técnicas de análisis de series temporales.	Identificación de tendencias, estacionalidad y fluctuaciones del tráfico en la red universitaria, permitiendo la predicción de la demanda de ancho de banda.	MAE, RMSE, MAPE: MAE ≤ 15%, RMSE ≤ 20%,
Comparar el desempeño de los modelos ARIMA, LSTM y Prophet en términos de error de predicción (RMSE, MAE, MAPE) y determinar cuál de ellos es el más adecuado para la estimación del tráfico en la red universitaria.	Evaluación del desempeño de ARIMA, LSTM y Prophet, seleccionando el modelo con mejor precisión y menor error en la predicción.	Comparación de MAE, RMSE, MAPE: Selección del modelo con menor error y mejor desempeño.
Implementar y validar el modelo predictivo más preciso y eficiente, evaluando su precisión en la estimación del tráfico de datos y su capacidad para anticipar la demanda futura de ancho de banda.	Implementación del modelo más eficiente, demostrando su capacidad para predecir picos de tráfico y optimizar la planificación del ancho de banda.	Reducción del error en la predicción de picos de tráfico en al menos un 18.7% (LSTM), comparación con ARIMA y Prophet

Nota: Elaboración propia (Solier, G., 2025).

1.4 Hipótesis

Las hipótesis de esta tesis se agrupan en dos categorías, las cuales se describen a continuación.

1.4.1 Hipótesis general

La implementación de un modelo predictivo basado en series temporales ARIMA, LSTM y Prophet reduce el error de predicción del tráfico de datos hasta en 15%, mejorando la precisión en la estimación del tráfico de datos y facilitando la planificación del ancho de banda en la red universitaria.

1.4.2 Hipótesis específicas

1. Los modelos predictivos basados en series temporales identificarán patrones de tráfico recurrentes con una precisión superior al 90% y estimarán la demanda futura de ancho de banda con un error promedio (MAE, RMSE) menor al 10%.
2. Entre los modelos ARIMA, LSTM y Prophet, el modelo LSTM es el más adecuado para la predicción del tráfico de red en la Universidad Nacional de Ingeniería, reduce el MAE hasta en 15% con una precisión superior al 92%.
3. La integración del modelo LSTM permitirá mejorar la detección de picos de tráfico y optimizar la planificación del ancho de banda, reduciendo los errores de estimación en al menos un 20% en comparación con ARIMA y Prophet.

1.4.3 Alcance

El presente estudio titulado "Predicción y evaluación del tráfico de red universitario utilizando modelos de series temporales: ARIMA, LSTM y Prophet" cubre un amplio espectro y aporta beneficios relevantes para múltiples actores involucrados. Entre ellos:

1. Administradores de redes universitarias: En la Universidad Nacional de Ingeniería los encargados de la administración podrán mejorar significativamente la gestión y eficiencia del tráfico de datos en la red institucional. Al aplicar modelos predictivos basados en series temporales posibilita anticiparse a los patrones de uso, detectar zonas con mayor carga y distribuir los recursos de manera más inteligente.
   Esto permitirá ofrecer a los estudiantes, académicos y personal administrativo un servicio de red estable, uniforme y receptivo.
2. Universidades y centros educativos: Las instituciones de educación superior que dependen de su conectividad a la red para desarrollar sus procesos tanto académicos como administrativos, se beneficiaran al tener una infraestructura de red optima y sin retrasos. De esta manera al implementar modelos tales como ARIMA, LSTM o Prophet se permitirá anticipar las necesidades de ancho

de banda para futuras adquisiciones, además permitirá posibles cuellos de botella y de esta manera optimizar la gestión de todo el sistema. Esto nos permitirá tener un servicio más ágil y eficaz para el usuario final.

3. Estudiantes y personal académico: Mayor velocidad, menor latencia y mayor estabilidad que permitirán un rendimiento optimo son los beneficios de contar con un ancho de banda optimo y esto favorecerá a todos los usuarios de la red universitaria. Dichas mejoras se verán reflejadas en una creciente mejora en la taza de aprendizaje, investigación y docencia; ya que aumentara el acceso digital de recursos compartidos en línea sin latencia.
4. Investigadores y académicos: Comprender la gestión y predicción de las redes de internet es una excelente contribución al análisis mediante series temporales del tráfico de red en entornos universitarios. Estos resultados serán de muy buena utilidad para alentar a la innovación futura en las redes de datos, aprendizaje automático y tecnologías avanzadas en el ámbito de gestión de la infraestructura de red.

En síntesis, la presente tesis titulada "Predicción del tráfico de datos de red en una universidad estatal utilizando modelos de series temporales: ARIMA, LSTM y Prophet" permite analizar un amplio espectro de aplicaciones en beneficio de la optimización de la infraestructura de la red universitaria. Calidad de servicios de internet en instituciones de educación superior y contribuir al área de conocimiento como redes de datos e inteligencia artificial.

1.5 Matriz de Operacionalización de variables de la investigación: "Predicción del tráfico de datos de red en una universidad estatal utilizando modelos de series temporales: ARIMA, LSTM Y PROPHET"

Tabla 2

Dimensiones e indicadores de las variables usadas

Variable	Dimensión	Indicador	Instrumentos
Variable Independiente: Modelos predictivos y sus configuraciones.	Modelos de predicción.	- Parámetros del modelo ARIMA (p,d, q)	- Software de análisis: Python con statsmodels, TensorFlow/Keras y Prophet.
		- Arquitectura y configuración de LSTM (capas, tamaño de ventana, tasa de aprendizaje).	- Software de análisis: Python con statsmodels, TensorFlow/Keras y Prophet.
		- Configuración de Prophet (estacionalidad, días festivos, tendencias).	- Software de análisis: Python con statsmodels, TensorFlow/Keras y Prophet.
	Configuración de los modelos	- Selección de hiperparámetros óptimos.	- Herramientas de ajuste de parámetros: Scikit-learn para optimización de hiperparámetros.
		- Ajustes en la metodología de entrenamiento y validación.	- Herramientas de ajuste de parámetros: Scikit-learn para optimización de hiperparámetros.
Variable Dependiente: Tráfico de datos de red (Promedio de bits por segundo)	Precisión de predicción	- Error Absoluto Medio (MAE).	- Herramientas de ajuste de parámetros: Scikit-learn para optimización de hiperparámetros.
		- Raíz del Error Cuadrático Medio (RMSE).	- Herramientas de ajuste de parámetros: Scikit-learn para optimización de hiperparámetros.
		- Error Porcentual Absoluto Medio (MAPE).	- Herramientas de ajuste de parámetros: Scikit-learn para optimización de hiperparámetros.
	Error de predicción	- Diferencia entre valores predichos y reales.	- Dataset histórico de tráfico de red (medido en promedio de bits por segundo).

Variable	Dimensión	Indicador	Instrumentos
		- Desempeño de las predicciones frente a valores reales del tráfico de red.	- Dataset histórico de tráfico de red (medido en promedio de bits por segundo
	Desempeño del modelo	- Comparación entre modelos (ARIMA, LSTM, Prophet) en términos de precisión.	- Herramientas de simulación: Scripts en Python y gráficos para evaluación visual.
		- Tiempo de ejecución y complejidad computacional.	- Herramientas de simulación: Scripts en Python y gráficos para evaluación visual.

Nota: Elaboración propia (Solier, G., 2025).

1.6 Antecedentes investigativos

En el contexto de las metodologías predictivas y de uso de modelos complejos de inteligencia artificial para conseguir la predicción de fenómenos, se han realizado trabajos de investigación los cuales ponen en comparación diferentes técnicas cuya finalidad es obtener el modelo que refleje mayor precisión y eficiencia en la predicción de dichos datos.

Un estudio relevante y notorio en este contexto es el de Chilón Ayay y Anghie Lizzeth (2023), el cual lleva por título "Redes neuronales recurrentes y modelos ARIMA para el pronóstico de la inflación en el Perú", fue desarrollado en la reconocida Universidad Nacional de Trujillo. En esta investigación bastante exhaustiva, los autores se centraron en la comparación entre un par de metodologías predictivas ampliamente conocidas: los modelos ARIMA y las redes neuronales recurrentes, específicamente el modelo LSTM, para el pronóstico preciso de la inflación en el Perú. El estudio abarcó una muestra de datos justo desde enero del año 2000 hasta el mes de diciembre del año 2021, y al diseñarse los modelos para cada serie temporal, se procedió entonces a realizar las predicciones correspondientes para el año posterior, es decir en el 2022. Los resultados demostraron que el modelo LSTM tuvo un error de evaluación bastante más bajo en comparación con ARIMA, empleando métricas tales como RMSE, MAE, MAPE, EMC e incluso MPE. Este estudio destaca la gran capacidad de las redes neuronales en contextos

determinados. También destaca su amplia aplicabilidad para predecir una gran cantidad de tráfico de datos en redes universitarias.

En el contexto especifico de la gestión eficiente del tráfico de datos y el ancho de banda en redes universitarias se han realizado investigaciones destacadas orientadas hacia la optimización de recursos mediante el uso de tecnologías avanzadas tales como las redes neuronales. Sangama Reyna en colaboración con Amaya Murayari (2019), en su tesis titulada "Redes neuronales artificiales para la mejora de la gestión del consumo de ancho de banda de Internet en la Municipalidad Distrital de Belén", demostraron de manera clara cómo estas redes neuronales pueden ofrecer predicciones bastante precisas sobre el consumo del ancho de banda, optimizando su planificación y distribución. Este estudio basado en un enfoque eminentemente descriptivo y con un diseño correlacional es un referente importante y notorio en la manera de usar redes neuronales artificiales en instituciones educativas, facilitando una mejor gestión del tráfico de datos en dichos establecimientos.

En una línea similar, Urdaneta Montiel (2020), en su denominada “Análisis de tráfico en una red LAN aplicando la tecnología de redes neuronales", exploró la viabilidad de poder utilizar redes neuronales para el análisis del tráfico de datos en redes locales (LAN). Este estudio se llevó a cabo en el Colegio Universitario "Dr. Rafael Belloso Chacín", empleó un diseño explicativo además de ser cuasiexperimental. Los resultados obtenidos en este estudio mostraron claramente una fiabilidad bastante alta del 90% en cuanto a la capacidad predictiva ofrecida por las redes neuronales. Este nivel de precisión destaca especialmente su gran efectividad para poder prever y gestionar adecuadamente el comportamiento del tráfico de datos dentro de redes locales. Además, este resultado apoya considerablemente su aplicabilidad también en entornos específicos tales como las redes universitarias.

Un antecedente significativo es el trabajo de Guerrero Meza y Renteros Parra (2023), titulado "Predicción de la demanda eléctrica de los edificios de la Facultad de Derecho y Edificio E de la UDEP mediante el uso de redes neuronales LSTM y TCN", realizado en la Universidad de Piura. En esta investigación, se compararon dos modelos

de predicción de series temporales: LSTM y TCN, para predecir la demanda eléctrica diaria de edificios académicos. Los resultados mostraron que el modelo TCN presentó un mejor desempeño con un MAPE de 8.35%. Este estudio demuestra cómo las redes neuronales pueden ser aplicadas para optimizar recursos críticos en infraestructuras educativas, una idea directamente paralela a la optimización del tráfico de datos en redes universitarias.

Para culminar se presenta un antecedente internacional destacado, se encuentra el trabajo de Villarroya Sánchez, C. (2021), titulado "Modelo basado en redes neuronales para la predicción de tráfico en la ciudad de Valencia", desarrollado en la reconocida Universitat Politècnica de València. El mencionado estudio se centró en la predicción del flujo vehicular en la ciudad de Valencia, España, se empleó específicamente las redes neuronales LSTM. La recolección de datos fue mediante el uso de espiras electromagnéticas distribuidas estratégicamente a lo largo de la ciudad de esta manera fue integrado a posterior en un modelo de predicción a corto plazo. Así se hace posible anticipar posibles congestionamientos y la hacer una gestión más eficiente del tráfico. Se resalta la importancia y efectividad de este estudio en la utilización de redes LSTM para predecir patrones complejos y no lineales y así definir su utilidad práctica para la optimización del tráfico vehicular. Su enfoque metodológico y los resultados obtenidos son un referente clave para desarrollar la investigación presente y de tal manera validar la aplicabilidad de redes LSTM en la predicción haciendo uso de datos temporales en escenarios complejos.

De esta manera los antecedentes mencionados muestran como las redes LSTM y ARIMA que son técnicas avanzadas de aprendizaje automático y pueden aplicarse en diversos contextos de optimización de recursos, que van desde aplicarlo en el análisis y predicción de redes vehiculares hasta infraestructuras de datos en redes universitarias.

Capítulo II. Marco teórico y conceptual

2.1 Marco teórico

Mediante una búsqueda detallada en diversas fuentes especializadas se han identificado varios trabajos de investigación que abordan problemas enfocados en resolver problemas asociados a la predicción y gestión del tráfico en redes empleando modelos de inteligencia artificial basados en redes neuronales. Se presenta de esta manera un análisis detallado de cada uno de estos estudios.

En primer lugar, se destaca el estudio de Villarroya Sánchez, C. (2021), titulado "Modelo basado en redes neuronales para la predicción de tráfico en la ciudad de Valencia", que plantea un modelo predictivo para tráfico urbano utilizando redes neuronales LSTM (Memoria a Largo y Corto Plazo). Esta investigación empleo datos obtenidos mediante espiras electromagnéticas instaladas estratégicamente en diversos puntos de la ciudad de Valencia con el objetivo principal de predecir con precisión el flujo vehicular a lo largo de distintos horarios del día. Este estudio resalta específicamente la eficacia de las redes LSTM para procesar series temporales debido a su capacidad para capturar patrones complejos y mantener relaciones en secuencias prolongadas de datos. El modelo propuesto ha demostrado ser altamente útil en la anticipación de congestiones vehiculares, facilitando de esta manera una mejor gestión del tráfico, así como el enrutamiento eficiente de vehículos y la reducción efectiva de la congestión vehicular urbana. Aunque el enfoque particular del estudio es el tráfico vehicular los resultados obtenidos también pueden ser abordados en la gestión del tráfico de redes de datos en instituciones educativas debido a que estas redes también presentan patrones variables y complejos en su comportamiento temporal.

En segundo lugar, el estudio de Fleischer (2017) titulado "*Using deep learning techniques to predict network traffic on the South African Research and Education Network*" , analiza la eficacia de utilizar redes neuronales artificiales en la predicción del tráfico de red, esta tarea es esencial para optimizar y mantener redes extensas y federadas, como

es el caso específico de la Red Sudafricana de Investigación y Educación (SANREN). El estudio enfatiza de manera especial en que las redes neuronales demuestran mayor precisión y eficiencia en comparación con métodos estadísticos tradicionales, debido a su habilidad para detectar dependencias de largo plazo y de patrones temporales complejos y poco usuales en el tratamiento de datos de series temporales. En particular se resalta la arquitectura LSTM como especialmente prometedora para este tipo específico de predicción. No obstante Fleischer concluye que aún se requiere profundizar más en la investigación sobre las redes LSTM en particular cómo mejorar su rendimiento y evaluar con mayor detalle su aplicabilidad en contextos específicos como el de SANREN. Estas conclusiones subrayan la relevancia y potencial aplicabilidad de dichas técnicas en redes universitarias amplias y altamente complejas.

En tercer lugar, Osorio D. (2021), en su investigación titulada "Predicción del consumo del ancho de banda de las aplicaciones web en la nube nativa basada en machine learning", analiza comparativamente tres algoritmos de redes neuronales para predecir el tráfico de red en aplicaciones web en la nube. Este trabajo subraya la importancia de anticipar el consumo de ancho de banda para optimizar los recursos de red y mejorar la calidad del servicio, especialmente en pequeñas y medianas empresas. Aunque su enfoque está dirigido a aplicaciones web en la nube, los principios y metodologías empleadas son altamente relevantes para la gestión de redes en universidades, donde el uso de servicios basados en la nube es cada vez más común.

En cuarto lugar, Millán Alonso (2006), en su tesis titulada "Predicción de tráfico en redes de telecomunicaciones basada en técnicas de inteligencia analítica", presenta una metodología para estimar el tráfico esperado en redes de telecomunicaciones. Este trabajo utiliza técnicas de inteligencia analítica y redes neuronales para predecir la carga de tráfico en una red de datos. Los resultados muestran la eficacia del modelo propuesto para identificar variables relevantes y realizar predicciones precisas sobre la utilización de enlaces en una red. No obstante, la orientación del estudio es aplicado a las redes de

telecomunicación, por lo cual requiere de adaptaciones específicas para ser llevado a una red universitaria.

En síntesis, los estudios en mención ofrecen distintas perspectivas para el uso de redes neuronales y técnicas de inteligencia artificial en el ámbito de predicción de tráfico en redes. El aporte de todas estas investigaciones en cuanto ideas y metodología es muy valioso, pero aun así presenta limitaciones al momento de abordarlas en un entorno universitario. Las redes universitarias presentan un tráfico variado y cambiante que implica desafíos propios. De esta manera la presente tesis abordara tales retos mediante la evaluación de los modelos avanzados ARIMA, LSTM y *Prophet*.

2.2 Marco conceptual

En esta sección, se explican los principales términos y definiciones clave que sustentan el marco teórico, los cuales son esenciales para comprender y aplicar correctamente las metodologías y técnicas utilizadas en este estudio

2.2.1 Redes Neuronales Artificiales (Artificial Neural Network, "ANN")

Redes neuronales artificiales (RNAs) son modelos computacionales inspirados en el funcionamiento del sistema nervioso biológico. Estas estructuras de procesamiento de información consisten en unidades interconectadas, llamadas neuronas artificiales, que trabajan en conjunto para realizar tareas específicas. Cada neurona artificial realiza operaciones matemáticas en la entrada recibida, produciendo una salida que se transmite a otras neuronas a través de conexiones ponderadas. El aprendizaje en redes neuronales artificiales se logra mediante la adaptación de estos pesos sinápticos en respuesta a la retroalimentación del sistema, permitiendo la capacidad de generalización y la resolución de problemas complejos en diversas áreas como reconocimiento de patrones, procesamiento del lenguaje natural y visión por computadora.

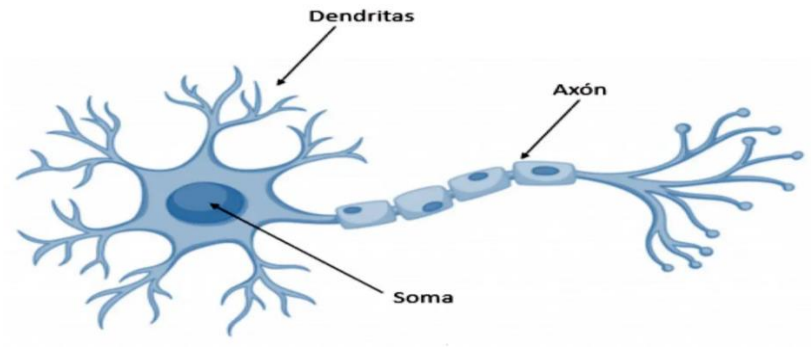
2.2.2 Neuronas Artificiales

En el caso del cerebro humano este está conformado por millones de neuronas interconectadas, en el caso de neuronas artificiales se trata de recrear las neuronas usando herramientas computacionales, que es usado para la resolución de problemas complejos.

En la Figura 1 y Figura 2 se muestra la comparación de una Neurona Artificial y una Neurona Biológica.

Figura 1

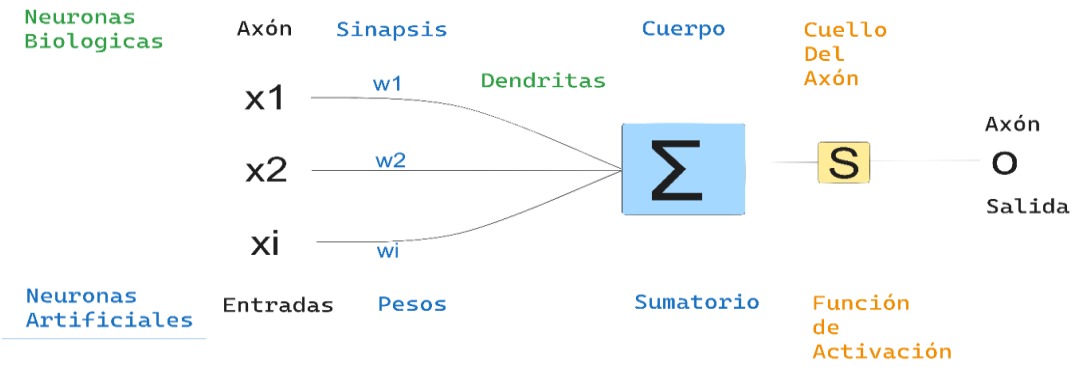
Neurona del Sistema Nervioso Humano



Nota: Elaborado por Universidad Nacional Autónoma de México.

Figura 2

Red Neuronal Artificial comparación con Red Neuronal Biológica.



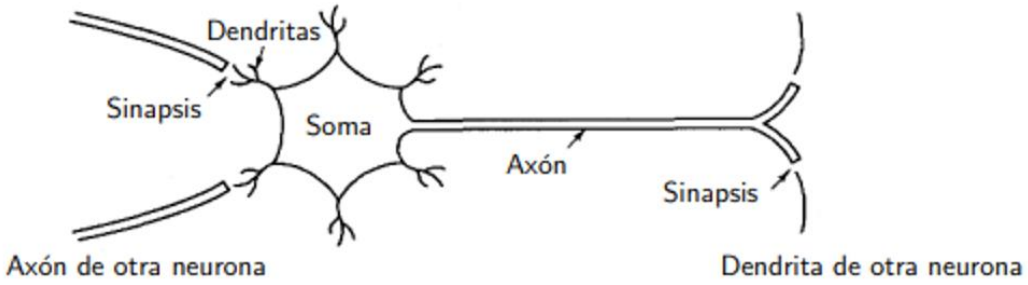
Nota: Elaboración propia (Solier, G., 2025).

2.2.3 Dendritas

En neurobiología, las dendritas son extensiones ramificadas del cuerpo celular de una neurona, diseñadas para recibir señales de otras neuronas. Estas estructuras son los responsables de transmitir información en el sistema nervioso, debido a que hacen posible la entrada de estímulos mediante conexiones sinápticas. De esta manera la comunicación sináptica en las dendritas ocurre por medio de transmisión eléctrica o en su defecto química, los cuales dependen de las características ocurridas en la sinapsis específica. En la Figura 3 se muestra la dendrita

Figura 3

Dendrita en una Red Neuronal Artificial comparación con Red Neuronal Biológica.



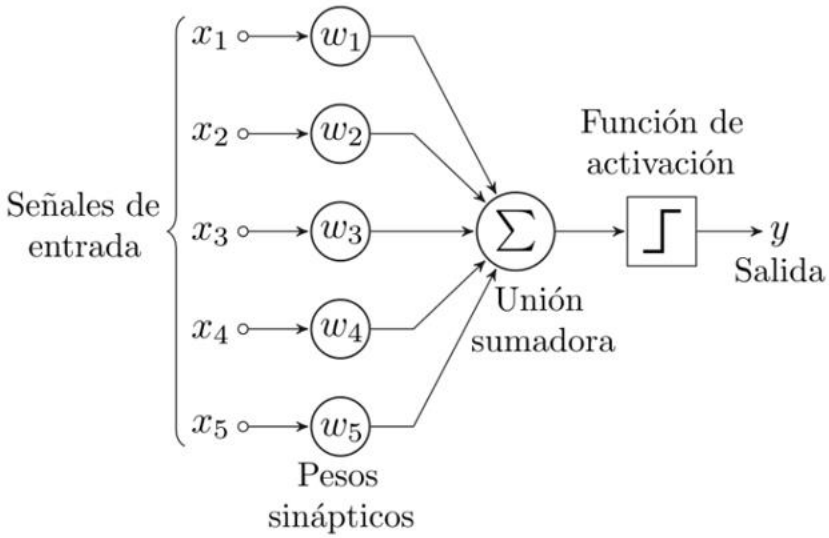
Nota: "Redes Neuronales Artificiales" Claudio Javier Tablada – Germán Ariel Torres.

2.2.4 Pesos

En el marco de las redes neuronales, los pesos representan valores numéricos asignados a las conexiones entre las neuronas. Estos parámetros determinan la fuerza y la dirección de la influencia de una neurona sobre otra. Durante etapa de entrenamiento, los pesos de la red se ajustan de forma iterativa y progresiva. De esta manera la red aprende patrones y realiza tareas específicas, de tal forma que permite adaptarse a los datos de entrada y mejorar su capacidad de generalización. En la Figura 4 se muestra los pesos asignados entre neuronas en una red neuronal artificial.

Figura 4

Pesos asignados a las conexiones entre neuronas.



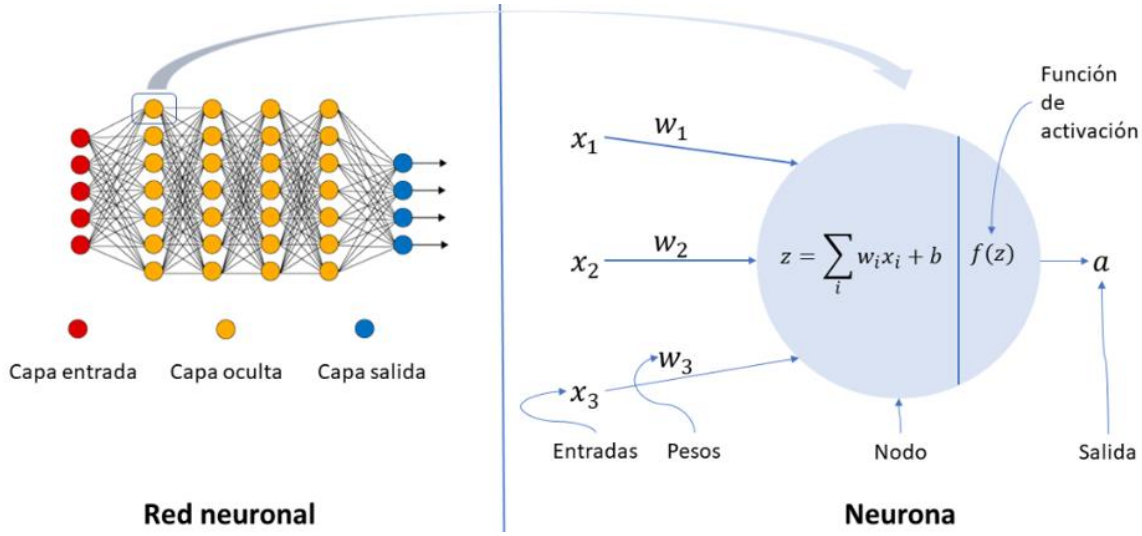
2.2.5 Función de Transferencia

En el contexto especifico de las redes neuronales, la función de transferencia es una expresión matemática que establece la relación entre la entrada total por una neurona y la respuesta que esta genera como salida. Esta función modela cómo la neurona responde a las señales de entrada y cómo transforma esta información para influir en la salida de la neurona.

La elección adecuada de la función de transferencia resulta un aspecto fundamental al momento de diseñar una red neuronal, debido a que influye en la capacidad de dicha red para aprender, adaptarse y generar patrones a partir de los datos de entrada proporcionados. Dicha función de activación se muestra en la Figura 5.

Figura 5

Función de Activación en una Red Neuronal.



Nota: "FUNCIONES DE ACTIVACIÓN". Jahaziel Ponce.

2.2.6 Arquitectura

La combinación de neuronas dentro de una red neuronal se realiza mediante capas, estos se muestran en la Tabla 3, que dependen de dichas capas y como estas se interconectan, así se tiene diferentes clasificaciones.

Tabla 3

Capas de la Arquitectura neuronal

Depende de	Clasificación
Numero de capas	Monocapa (1 capa) Multicapa (Mas de una capa)
Tipos de Conexiones	Recurrente (por realimentación) No recurrente
Número de conexiones	Totalmente conectada Parcialmente conectada

Nota: Elaboración propia (Solier, G., 2025).

2.2.7 Perceptrón Multicapa

La red neuronal conocida como perceptrón multicapa es una clase especifica de red neuronal artificial estructurada en múltiples capas. Este modelo es ampliamente usado para resolver problemas de clasificación, modelamiento matemático, predicción en series temporales y otros problemas complejos. A continuación, se presenta el proceso de aprendizaje en una red neuronal de perceptrón multicapa:

Capas:

1. Capa de Entrada: Esta primera capa está compuesta por nodos que representan las variables o características del sistema que serán ingresados a la red. Así cada uno de estos nodos en esta capa corresponden a una característica concreta del conjunto de datos usado en el entrenamiento.
2. Capas Ocultas: Son las capas intermedias entre la capa de entrada y la capa de salida y puede ser una o varias. Estas capas están conformadas por nodos, también llamados neuronas, dichas salidas se calculan a partir de la entrada de la capa anterior. La inclusión de capas ocultas permite que la red identifique patrones más complejos y ejecute tareas más sofisticadas y avanzadas.
3. Capa de Salida: Esta capa es la responsable de generar los resultados o respuestas que ofrece la red neuronal. Dependiendo del problema que se busque resolver, la mencionada capa de salida puede estar formada de uno a múltiples nodos o neuronas.

Tabla 4

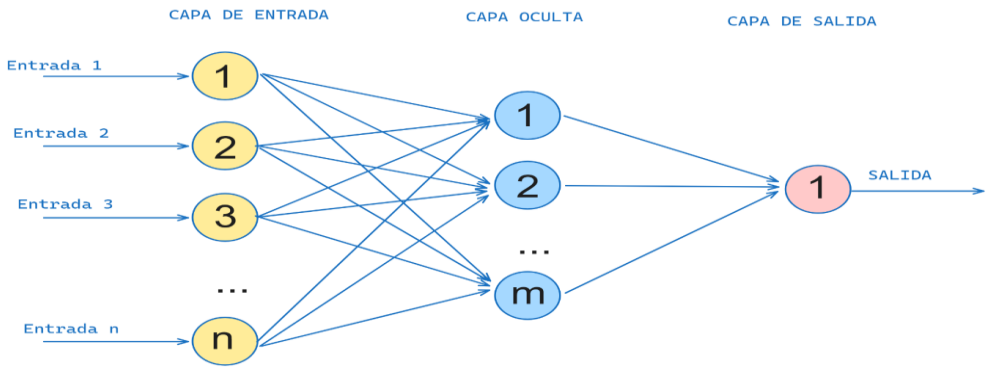
Capas de una red neuronal

Nivel de capa	Nombre
Primera Capa	Entrada
Capa Intermedia	Oculta
Ultima Capa	Salida

Nota: Elaboración propia (Solier, G., 2025).

Figura 6

Red Perceptrón Multicapa



Nota: Elaboración propia (Solier, G., 2025).

2.2.8 Predicción del Tráfico De Datos de Internet

Los llamados modelos de tráfico de datos de Internet consisten en herramientas teóricas y matemáticas diseñadas para describir y pronosticar el comportamiento del flujo de información dentro de las redes de Internet. A través de estos modelos se hace posible comprender cómo transitan los datos, que factores influyen y de qué manera se pueden gestionar los recursos de red de manera eficiente. La predicción del tráfico se vuelve especialmente relevante en entornos de uso donde la red varia significativamente tal como sucede en las instituciones educativas debido a que permite optimizar recursos, prever picos de demanda y mejorar la calidad de servicio adecuada para el acceso a la red.

2.2.9 Series Temporales

Las series temporales se componen de un conjunto de mediciones u observaciones obtenidas en intervalos regulares de tiempo. Esta secuencia cronológica de datos permite identificar patrones repetitivos, fenómenos estacionales y tendencias en general, estos aspectos nos ayudan en la predicción del ancho de banda en redes de datos. La modelización basada en series temporales se utiliza en el análisis de datos en los cuales los puntos en observación dependen de valores previos, esta es una característica importante en el flujo de tráfico de datos en redes universitarias. Según Hyndman y Athanasopoulos, las series temporales permiten “extraer patrones útiles para la predicción y comprensión de cambios estructurales en el tiempo” (p.4) . En las redes universitarias el tráfico de datos varia sustancialmente de acuerdo al horario del día, es decir de acuerdo a horarios de clase, periodos de exámenes y vacaciones. De esta manera las series temporales nos brindan una herramienta eficaz en la gestión de red.

De esta manera para hacer un adecuado y óptimo uso de las series temporales aplicadas en la predicción del ancho de banda, es necesario realizar de manera correcta la preparación de los datos, manejo de valores atípicos, tratar los datos y normalizarlos. Es así como estas técnicas permiten mejorar la precisión y confiabilidad de los modelos planteados. Además de ello se pueden incluir métodos tales como el suavizado exponencial o también el modelado estacional con la finalidad de capturar patrones recurrentes que son recurrentes en los entornos académicos universitarios.

2.2.10 Redes Neuronales Recurrentes (RNN)

Las redes neuronales recurrentes (RNN) constituyen una clase de modelos de Deep Learning creados para conocer y aprovechar secuencias temporales, un caso de aplicación son las redes temporales gracias a la retroalimentación interna que caracteriza su estructura. La habilidad de “recordar” información previa dota a las RNN de poder representar a lo largo del tiempo, el cual es muy importante cuando se realiza la predicción del tráfico de datos en las redes. Liu et al., destaca “las RNN ofrecen una alternativa eficiente para la predicción de series temporales en redes complejas, adaptándose a la

naturaleza dinámica de las redes de datos". Debido a ello las Redes Neuronales Recurrentes aprenden patrones de corto y mediano plazo en el flujo de datos de redes universitarias y de esta manera se adapta a las variaciones del uso del ancho de banda en la universidad.

Aunque las Redes Neuronales Recurrentes ofrecen beneficios, se presentan dificultades para conservar información a largo plazo debido al problema del desvanecimiento de gradientes. Para superar esta limitación es conveniente recurrir a arquitecturas más sofisticadas, una de ellas son las redes LSTM descritas posteriormente. Por otro lado, las Redes Neuronales Recurrentes convencionales resultan útiles en el análisis de series temporales en situaciones de periodicidad a corto plazo, sin embargo, esto influye en el deterioro de su desempeño, donde RNN estándar pueden ser adecuadas para el análisis de series temporales en situaciones donde la periodicidad es de corto plazo, sin embargo, su rendimiento se ve deteriorado en contextos donde el tráfico sufre cambios prolongados y variados en el tráfico de red.

2.2.11 Modelos ARIMA

El modelo ARIMA (Promedio Móvil Integrado Autorregresivo) se consolida como una de las herramientas esenciales para el análisis de series temporales, debido a que permite modelar datos tanto estacionarios como no estacionarios mediante la combinación de componentes autorregresivos y de media móvil. Gracias a esta estructura el presente modelo es capaz de captar tanto tendencias lineales como patrones estacionales, lo que hace especialmente útil para la predicción del tráfico en redes con un uso estable. Tal como señala Zhang, "ARIMA sigue siendo un método confiable para la predicción de series temporales debido a su simplicidad y eficacia en escenarios de dependencia lineal" (p. 18).

Pese a que ARIMA posee fortalezas este modelo enfrenta limitaciones para capturar patrones que son no lineales y estacionalidades complejas, de esta manera dicha situación es frecuente en entornos universitarios en los cuales los picos de tráfico no siempre siguen patrones regulares además pueden verse influenciados por eventos no tan frecuentes como conferencias o también actividades especiales. En consecuencia, el

modelo ARIMA puede mezclarse con otros enfoques, con la finalidad de mejorar su precisión en la predicción de tráfico de datos de corto plazo.

2.2.12 Prophet

Prophet, desarrollado por Facebook, es una herramienta de predicción de series temporales el cual consiste en descomponer de manera aditiva los componentes de tendencia, estacionalidad y efectos residuales. En comparación con otros métodos Prophet tiene la capacidad de gestionar irregularidades en la data, tales como los valores atípicos o también datos faltantes, lo que resulta sumamente útil en las redes universitarias donde a menudo existe la posibilidad de presentar picos de tráfico inesperados. Según los autores Taylor y Letham indican que Prophet “proporciona un marco flexible para la predicción de series temporales de uso complejo, ofreciendo predicciones robustas en escenarios de datos difíciles” (p. 42).

Prophet puede detectar múltiples estacionalidades, de esta manera incluye patrones diarios y semanales típicos en las redes académicas lo que ayuda considerablemente en la planificación del ancho de banda. Además, es capaz de incorporar eventos especiales los cuales alteran el tráfico, de esta amanera se convierte en una buena opción para escenarios donde se desarrolla alta variabilidad o ruido en los datos

2.2.13 Long Short-Term Memory (LSTM)

Las redes de memoria a largo y corto plazo LSTM, son una evolución de las Redes Neuronales Recurrentes diseñada para gestionar específicamente dependencias a largo plazo en series temporales. Debido a sus celdas de memoria, LSTM puede retener información durante períodos prolongados, lo cual resulta crucial trabajar con datos que presenta patrones complejos o prolongados. De esta manera Hochreiter y Schmidhuber con esta arquitectura solucionaron el problema de desvanecimiento de gradientes, limitante de las RNN convencionales para captar dependencias extendidas. De esta manera las Memorias a corto-largo plazo (LSTM) son eficaces para la predicción de tráfico de red en entornos universitarios, donde los patrones de uso pueden incluir componentes tanto a corto como a largo plazo.

Además, Greff et al. afirmo que las Memorias a corto-largo plazo (LSTM) funciona de manera óptima en escenarios donde es alta la variabilidad, en nuestro caso de redes universitarias en donde el tráfico cambia de patrón considerablemente durante el día, se requiere ajustes en tiempo real para gestionar eficientemente el ancho de banda.

2.2.14 Minería de Datos

La minería de datos usa técnicas de análisis con la finalidad de extraer patrones de grandes volúmenes de datos, de esta forma se puede detectar patrones y segmentar distintos tipos de usuarios en las redes de datos. De esta manera se agiliza la personalización de modelos predictivos ajustados a comportamientos del grupo de usuarios (como estudiantes, docentes y personal administrativo). Han, Pei y Kamber afirman sobre la minería de datos “permite optimizar el proceso de preparación de datos, lo cual incrementa la precisión y eficiencia de los modelos predictivos en la gestión de redes”.

2.2.15 Integración de Flask y Dash para Aplicaciones Web

En este proyecto Flask y Dash construyen una interfaz web robusta en la que Flask se ocupa de la autenticación y la lógica del backend, a su vez Dash construye dashboards interactivos. Esta integración permite centralizar en una misma plataforma la visualización de los datos y la autenticación, permitiendo ofrecer así seguridad e interactividad la gestión y análisis de datos. Con Flask se controla las rutas y autenticación de usuarios haciendo uso de Fl2ask-Login, mientras que por su parte Dash que está basado en Plotly, proporciona un entorno visual dinámico para monitorear en tiempo real los datos de tráfico.

Con el uso de Flask-Login se implementa la solución de autenticación solo para usuarios registrados puedan acceder a la aplicación, y de esta manera se protege la información sensible, así como también los datos y su análisis. Por lo cual resulta muy importante integrarlo a las aplicaciones de análisis de tráfico de red, en el cual es primordial mantener bajo control la integridad de los datos de red.

2.2.16 Manejo de Series Temporales en Redes Neuronales Recurrentes (RNN)

En este proyecto para hacer el modelado y predecir patrones de series temporales de datos de trafico de red se hace uso de Redes Neuronales Recurrentes (RNN) y su variante LSTM (Long Short-Term Memory). Las redes LSTM retienen la información a largo plazo, y de este modo captura patrones en datos de tráfico de red donde las dependencias temporales tienen complejidad y variabilidad. Así los modelos LSTM aprenden de los patrones históricos y predice los comportamientos futuros, los cuales se adaptan a los cambios estacionales y anomalías.

La arquitectura de LSTM es óptima para analizar el tráfico de red, debido a que captura dependencias de corto y largo plazo, importante en redes donde el uso de ancho de banda varia de manera significativa debido a eventos como exámenes o conferencias.

2.2.17 Modelos ARIMA y Prophet para la Predicción de Series Temporales

Con los modelos ARIMA y Prophet en el presente proyecto se predecirá las series temporales de tráfico. ARIMA resulta eficaz para el modelado estadístico de series temporales estacionarias, identificando los patrones lineales y también estacionales, por su parte Prophet un modelo diseñado por Facebook, fue creado con el propósito de analizar series temporales con tendencias no lineales y también estacionalidades complejas. Prophet resulta especialmente útil en contextos con ausencia de datos, así como también variaciones repentinas tal como ocurre en redes universitarias donde el tráfico puede fluctuar considerablemente en períodos de exámenes.

Prophet hace gestión de forma eficaz datos faltantes y variaciones estacionales específicas, lo que lo transforma en una herramienta ideal para la predicción de tráfico de red y también de la captura de irregularidades comunes en redes de instituciones educativas.

2.2.18 Preprocesamiento de Datos y Escalamiento

El código implementa MinMaxScaler con el fin de normalizar los datos de tráfico de red, llevándolo a un rango entre 0 y 1, lo que resulta clave para optimizar el rendimiento de redes neuronales. El proceso de normalización facilita el trabajo de los algoritmos de

aprendizaje profundo a converger más rápidamente y aumentar la precisión del modelo, especialmente en series temporales cuando la escala de los valores de tráfico puede cambiar significativamente.

El preprocesamiento también incluye la conversión de unidades (Mbps, Kbps) a valores numéricos homogéneos, lo que posibilita una representación homogénea de los datos y también simplifica su interpretación y aplicación en modelos predictivos.

2.2.19 Validación Cruzada para Series Temporales

La validación cruzada divide la información en distintas "ventanas de tiempo", evalúa la robustez y consistencia de los modelos predictivos. A diferencia de la validación cruzada convencional, este método funciona conservando la dependencia temporal entre los datos esta función es muy importante en series temporales.

En el caso de modelos como ARIMA y LSTM la validación cruzada hace que las predicciones no tengan dependencia de datos futuros en la etapa de validación, de esta manera evita un sesgo artificial en las métricas de evaluación y estima con más precisión el rendimiento del modelo en la etapa de producción.

2.2.20 Métricas de Evaluación de Modelos

EL código usa RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), MAPE (Mean Absolute Percentage Error) y Huber Loss para evaluar el desempeño de los modelos predictivos. Cada métrica da otro enfoque distinto con respecto a la precisión de las predicciones, así RMSE y MAE son usados más comúnmente para errores absolutos mientras que la función Huber Loss resulta útil cuando existen valores atípicos.

En series temporales se usan métricas como MASE (Mean Absolute Scaled Error) y RAE (Relative Absolute Error) para medir su eficacia del modelo con respecto a otros modelos de referencia. Así se lleva a cabo un análisis detallado respecto a la precisión del modelo a través de distintos niveles de variabilidad en el tráfico.

2.2.21 Uso de Gráficas y Visualización en Dash

Dash tiene uso en el código ya que permite visualizar de manera grafica e interactiva las predicciones de tráfico, comparando valores reales y predichos por

diferentes modelos. La interfaz mide desempeños de ARIMA, LSTM y Prophet en tiempo real, con una mejor comprensión y análisis de los datos y así tomar decisiones más eficaces en la gestión del tráfico de red.

La interfaz mediante gráficos de barras detecta más rápidamente que modelo tiene mejor eficiencia, al ser más visible el desempeño de cada técnica predictiva.

2.2.22 Predicción Iterativa con LSTM

El enfoque de predicción iterativa en LSTM hace proyecciones a largo plazo de esta manera el resultado de una predicción se use como entrada para el siguiente paso de predicción. Esto resulta de mucha importancia para extender las predicciones más allá del conjunto de prueba original.

La predicción iterativa resulta esencial en series temporales ya que facilita la proyección de tendencias futuras. No obstante, esto puede generar errores acumulativos por eso es muy importante un ajuste meticuloso con el objetivo de preservar la precisión en predicciones de largo alcance.

2.2.23 Cross-Validation en Prophet

Prophet implementa una validación cruzada de alto nivel haciendo uso de las técnicas de cross_validation y tambien performance_metrics, que estiman métricas promedio en distintas ventanas de tiempo para analizar la robustez de dicho modelo. De esta manera es posible identificar si el modelo conserva su exactitud en períodos de alta variabilidad.

La validación cruzada es importante para el análisis de tráfico de red ya que facilita identificar cuando Prophet puede manejar cambios abruptos en patrones estacionales y además asegurar que las predicciones tengan la confiabilidad en diferentes períodos.

2.2.24 Logística y Control de Acceso con Flask-Login

Flask-Login gestiona en el proyecto la autenticación de usuarios al momento de ingresar al panel interactivo de esta forma solo usuarios registrados y autenticados accedan a los dashboards de predicción de tráfico. Así se tiene control de acceso adecuado para proteger la información sensible presente en la aplicación.

En aplicaciones de análisis de datos, la autenticación es fundamental ya que restringe el acceso a la visualización y manejo de datos sensibles, protege la integridad de los datos y asegura que las métricas y predicciones sean accesibles y visibles solo para los usuarios autorizados.

2.2.25 Logging para Depuración y Auditoría

El módulo logging en Python registra eventos como el estado de carga de archivos, los errores en el procesamiento y el estado de entrenamiento de los modelos. Es así como el proceso permite la identificación rápida y resolución de problemas, de esta manera registra información sobre el rendimiento del sistema y el flujo de datos en tiempo real.

En la etapa de producción el logging audita el rendimiento y también mantiene un historial de la actividad del sistema. Tiene mucha importancia para la depuración en aplicaciones complejas de análisis predictivo, mejorando así la detección y corrección de errores de una forma rápida.

CAPÍTULO III. Desarrollo del trabajo de investigación

Este capítulo desarrolla la metodología que se utiliza para la predicción del tráfico de red en la Universidad Nacional de Ingeniería (UNI) mediante modelos de series temporales. Se analizan tres enfoques clave: ARIMA (AutoRegressive Integrated Moving Average), LSTM (Long Short-Term Memory) y Prophet, elegidos por su eficacia en el modelado de datos temporales y la predicción de tendencias futuras basada en patrones históricos.

La investigación se enmarca en un enfoque cuantitativo y de diseño explicativo con enfoque experimental orientado al desarrollo y validación de modelos predictivos aplicables en la gestión del tráfico de red en la UNI. De acuerdo con el método científico que fue propuesto por Sampieri et al. (2022), la metodología presenta fases: recolección de datos, preprocesamiento, modelado, validación y análisis de resultados.

El objetivo principal es optimizar la gestión el uso de los recursos de la red, de tal manera que sea posible anticipar congestiones y también mejorar la distribución del tráfico con lo cual se responde eficientemente la demanda de los usuarios.

3.1 Enfoque

La investigación se desarrolla bajo un enfoque cuantitativo, al centrarse en el análisis de los datos numéricos del tráfico de red del Campus de la Universidad Nacional de Ingeniería, que se obtienen del software de monitoreo OpManager y la herramienta Panorama de Palo Alto. Se aplican métodos estadísticos y de aprendizaje profundo que permiten evaluar y predecir el comportamiento del tráfico.

3.2 Población y Muestra

3.2.1 Población

La población está constituida por los registros de tráfico de red de la Universidad Nacional de Ingeniería, recopilados en los siguientes puntos de monitoreo:

1. Routers Huawei (Activo-CETEL y Pasivo-CITIC).
2. Firewalls Palo Alto PA 3260 (Activo/Pasivo).

3. Switches Huawei (Core de la red interna).

Estos dispositivos generan logs y métricas sobre el tráfico de red en la institución.

3.2.2 Muestra

Se utiliza un muestreo no probabilístico por conveniencia, seleccionando datos históricos del sistema Panorama de Palo Alto, con las siguientes características:

1. Período de análisis: Últimos 12 meses (01 de Octubre del 2023 hasta 31 de Octubre 2024).
2. Frecuencia de captura: Datos recolectados en intervalos de 1 minuto diariamente.

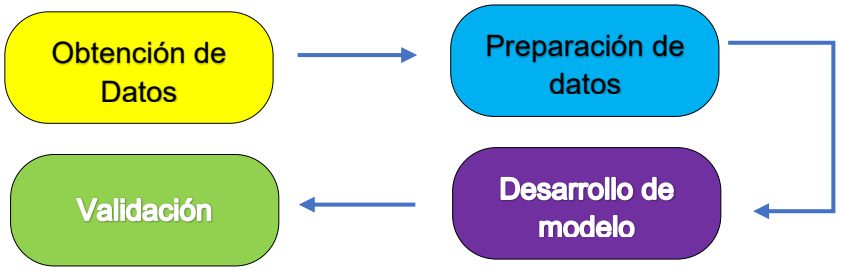
3.3 Metodología

La metodología empleada para la predicción del tráfico de red en la UNI sigue un enfoque estructurado y se divide en cuatro fases principales: Obtención de datos, Preparación de datos, Desarrollo del modelo y Validación.

A continuación, en la Figura 7, se describe cada una de estas fases basadas en estudios previos y las mejores prácticas documentadas en la literatura.

Figura 7

Diagrama de Flujo para la predicción del tráfico en la Universidad Nacional de Ingeniería.



Nota: Elaboración propia (Solier, G., 2025).

El diseño es cuasi-experimental, ya que no se manipulan variables, sino que se analizan datos históricos obtenidos en condiciones operativas reales de la infraestructura de red de la UNI.

3.3.1 Obtención de datos

La obtención de datos es un paso crítico para el desarrollo de cualquier modelo predictivo. Según Hyndman y Athanasopoulos (2018), la calidad y la precisión de los datos

son fundamentales para el éxito de los modelos de pronóstico en series temporales. Para este trabajo, los datos de tráfico de la red fueron recolectados entre el 1 de octubre de 2023 y el 30 de Octubre de 2024, con una frecuencia de un minuto, lo cual permite capturar las variaciones en el tráfico en intervalos suficientemente pequeños para predecir picos y valles de uso en la red de la Universidad Nacional de Ingeniería.

Los datos utilizados en esta investigación incluyen:

1. Time (Tiempo): Marca temporal de cada medición.
2. Bits per Second (Avg): Tráfico promedio en bits por segundo.

Estos datos fueron recolectados de los sistemas de monitoreo de tráfico de la UNI y almacenados en archivos CSV para su posterior procesamiento. Agrupados en tres categorías:

1. InTraffic (Tráfico Entrante): El tráfico que ingresa a la red.
2. OutTraffic (Tráfico Saliente): El tráfico que sale de la red.
3. TotalTraffic (Tráfico Total): La suma del tráfico entrante y saliente.

La ubicación de los datos está almacenada en la ruta D:\DATAPY\DATA TESIS, asegurando la integridad y accesibilidad de la información.

3.3.2 Carga de Datos

En este estudio, la recolección de datos de tráfico de red se realiza a través del firewall Palo Alto PA-3260, un dispositivo clave para la seguridad perimetral y el monitoreo del tráfico. Para capturar métricas de tráfico se usa herramientas como NetFlow/IPFIX y SNMP, estas son configuradas para extraer datos en intervalos de 1 minuto (Cisco, 2022).

Debido a que el firewall gestiona altos volúmenes de tráfico en tiempo es importante hallar un equilibrio entre la precisión del monitoreo y eficiencia computacional. Según Palo Alto Networks (2023), los dispositivos de seguridad de red requieren una frecuencia de muestreo óptima con el propósito de identificar eventos relevantes sin generar una sobrecarga innecesaria.

a) Balance entre granularidad y eficiencia

- Capturas con una frecuencia inferior a 1 minuto (ej. 10-30 segundos) generan un volumen excesivo de datos, lo que incrementa la carga en los sistemas de almacenamiento y análisis (Palo Alto Networks, 2023).
- Intervalos superiores a 5 minutos son inadecuados para detectar fluctuaciones súbitas en el tráfico, retrasando la identificación de eventos críticos como ataques DDoS o picos de consumo (ManageEngine, 2023).
- Investigaciones como las de Elsayed (2020) y Romo Caicedo (2021) demuestran que un muestreo de 1 minuto permite capturar tanto patrones generales como anomalías locales, optimizando la precisión de los modelos predictivos sin sobrecargar el sistema.
- En soluciones de monitoreo como OpManager, SolarWinds y Cisco NetFlow, los valores de muestreo recomendados oscilan entre 1 y 5 minutos, validando la selección de 1 minuto como un estándar de la industria (SolarWinds, 2021).

b) Comparación con Estándares en la Industria

El intervalo de 1 minuto se ajusta a las prácticas de monitoreo reconocidas en la industria, como se detalla en la siguiente Tabla 5:

Tabla 5

Comparativo de métricas de evaluación de los modelos

Tecnología	Frecuencia Recomendada	Uso
NetFlow (Cisco, Palo Alto, Fortinet)	**1 min (ajustable de 30s - 5 min)**	Captura de tráfico en firewalls (Cisco, 2022).
SNMP (ManageEngine OpManager, PRTG)	**1 - 5 min**	Monitoreo de tráfico y recursos (ManageEngine, 2023).
Telemetría basada en modelos (Cisco YANG, Palo Alto)	**30s - 1 min**	Análisis en tiempo real (Palo Alto Networks, 2023).

Nota: Elaboración propia (Solier, G., 2025).

El intervalo de 1 minuto permite registrar eventos críticos sin que genere información redundante, lo que garantiza un monitoreo preciso y eficiente (Elsayed, 2020).

La elección de este intervalo de 1 minuto también está respaldada por diversas investigaciones y estándares. Romo Caicedo (2021) señala que un muestreo de 1 minuto equilibra entre granularidad y carga computacional, mientras que intervalos de 15 minutos o más dificultan la detección oportuna de anomalías. Cisco (2022) recomienda 1 minuto en NetFlow para lograr eficiencia en la supervisión del tráfico. SolarWinds (2021) ofrece configurar su SNPM a 1 minuto para mejorar la detección de anomalías.

3.3.3 Carga de Datos

Los datos fueron cargados al entorno de Python utilizando la biblioteca Pandas (McKinney, 2010). Se especificó explícitamente la ruta de los archivos para garantizar el acceso adecuado. En la Figura 8 se muestra el código Python usado.

Figura 8

Lectura de archivos CSV para el análisis de tráfico de datos.

```
import pandas as pd
# Cargar los archivos desde la ruta proporcionada
in_traffic = pd.read_csv(r'D:\DATAPY\NO TOCAR ESTA DATA\InTraffic.csv')
out_traffic = pd.read_csv(r'D:\DATAPY\NO TOCAR ESTA DATA\OutTraffic.csv')
total_traffic = pd.read_csv(r'D:\DATAPY\NO TOCAR ESTA DATA\TotalTraffic.csv')
```

Nota: Este código utiliza pandas para cargar datos de tráfico desde archivos CSV almacenados localmente. Elaboración propia (Solier, G., 2025).

3.3.4 Preparación de Datos

El objetivo de esta fase fue limpiar y transformar los datos para su uso en los modelos de predicción. Esto incluyó:

1. Se eliminó la zona horaria "PET" de la columna Time.
2. La columna Time se convirtió al formato datetime.
3. Los valores de la columna Bits per Second (Avg), inicialmente en texto con unidades "M" (millones), se transformaron en valores numéricos y se multiplicaron por 10^6.

El proceso de limpieza de datos consistió en preparar las series temporales de tráfico de red para garantizar que fueran consistentes, completas y representativas. Este procedimiento se dividió en cuatro pasos principales, descritos en la Tabla 6, y el código Python usado se muestra en la Figura 9. :

Tabla 6

Comparación de las métricas de evaluación de los modelos

Paso 1	Completar Datos Vacíos
Paso 2	Agregar Datos por Horas
Paso 3	Completar Días Faltantes
Paso 4	Corregir Lecturas Erróneas

Nota: Elaboración propia (Solier, G., 2025).

Figura 9

Preprocesamiento de datos de tráfico.

```
def preprocess_data(df):
    # Limpieza de datos
    df['Time'] = df['Time'].str.replace(' PET', '', regex=False)
    df['Time'] = pd.to_datetime(df['Time'], format='%d %b %Y %I:%M:%S %p', errors='coerce')
    df['Bits per Second(Avg)'] = pd.to_numeric(df['Bits per Second(Avg)'].str.replace(' M', '')
    .str.replace(',', ''), errors='coerce') * 1e6
    # Interpolación de datos faltantes
    df['Bits per Second(Avg)'] = df['Bits per Second(Avg)'].interpolate()
    return df

# Aplicar preprocesamiento
in_traffic = preprocess_data(in_traffic)
out_traffic = preprocess_data(out_traffic)
total_traffic = preprocess_data(total_traffic)
```

Nota: Este código limpia y convierte datos de tráfico, ajustando fechas y valores numéricos, e interpola datos faltantes para análisis continuo. Elaboración propia (Solier, G., 2025).

3.3.5 Visualización de Datos

3.3.5.1 Graficado General

Se generaron gráficos para analizar la evolución del tráfico de red a lo largo del tiempo. El gráfico combinado permite observar las tendencias del tráfico de entrada, salida y total, el código usado esta descrito en la Figura 10.

Figura 10

Visualización del tráfico de red en el tiempo.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(14, 8))
plt.plot(in_traffic['Time'], in_traffic['Bits per Second(Avg)'], label='In Traffic', alpha=0.7)
plt.plot(out_traffic['Time'], out_traffic['Bits per Second(Avg)'], label='Out Traffic', alpha=0.7)
plt.plot(total_traffic['Time'], total_traffic['Bits per Second(Avg)'], label='Total Traffic', alpha=0.7)
plt.title('Network Traffic Over Time', fontsize=16)
plt.xlabel('Time', fontsize=12)
plt.ylabel('Bits per Second (Avg)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Nota: Este código utiliza Matplotlib para graficar el tráfico de red entrante, saliente y total a lo largo del tiempo, proporcionando una visualización clara de las tendencias. Elaboración propia (Solier, G., 2025).

3.3.5.2 Identificación de Picos

Se analizaron los picos de tráfico para identificar los valores máximos en cada conjunto de datos, destacándolos en los gráficos individuales y documentándolos en tablas.

El tráfico desde el 1 de octubre de 2023 y el 30 de octubre de 2024 registrado se muestra en la Figura 12, Figura 13, Figura 14 y Figura 15 para Intraffic, Outraffic y TotalTraffic respectivamente.

Figura 11

Identificación de picos en el tráfico de red.

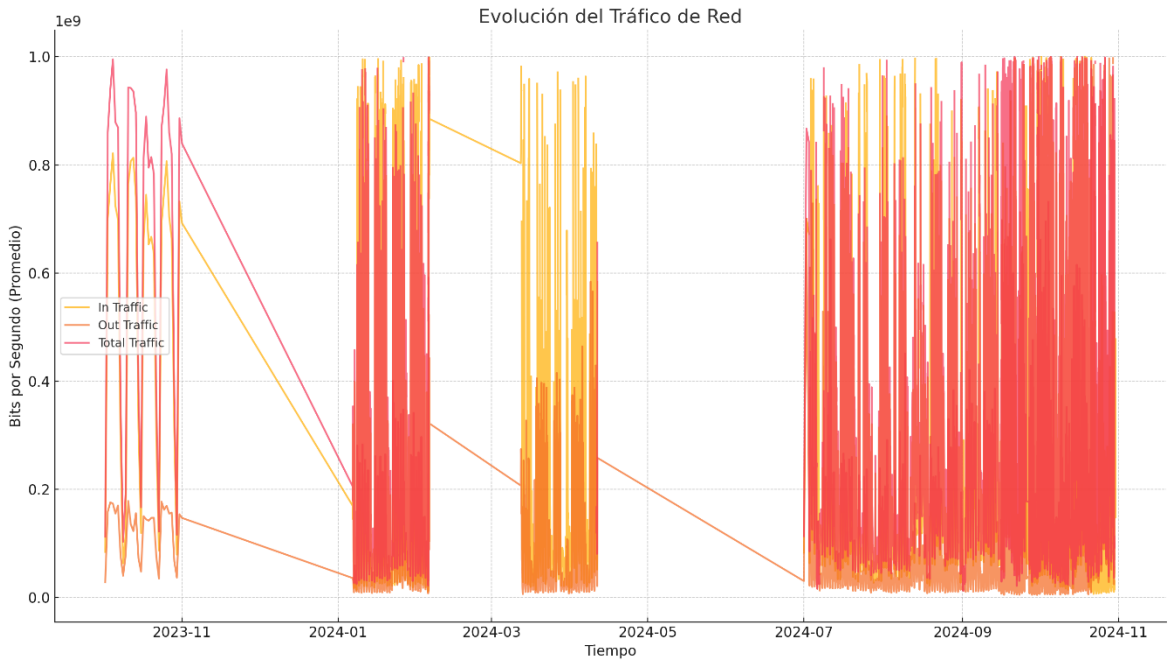
```
# Encontrar picos
in_peak = in_traffic.loc[in_traffic['Bits per Second(Avg)'].idxmax()]
out_peak = out_traffic.loc[out_traffic['Bits per Second(Avg)'].idxmax()]
total_peak = total_traffic.loc[total_traffic['Bits per Second(Avg)'].idxmax()]

print(f"In Traffic Peak: {in_peak['Bits per Second(Avg)']} on {in_peak['Time']}")
print(f"Out Traffic Peak: {out_peak['Bits per Second(Avg)']} on {out_peak['Time']}")
print(f"Total Traffic Peak: {total_peak['Bits per Second(Avg)']} on {total_peak['Time']}")
```

Nota: Este código identifica los valores pico de tráfico de red entrante, saliente y total, indicando el momento exacto en que ocurrieron. Elaboración propia (Solier, G., 2025).

Figura 12

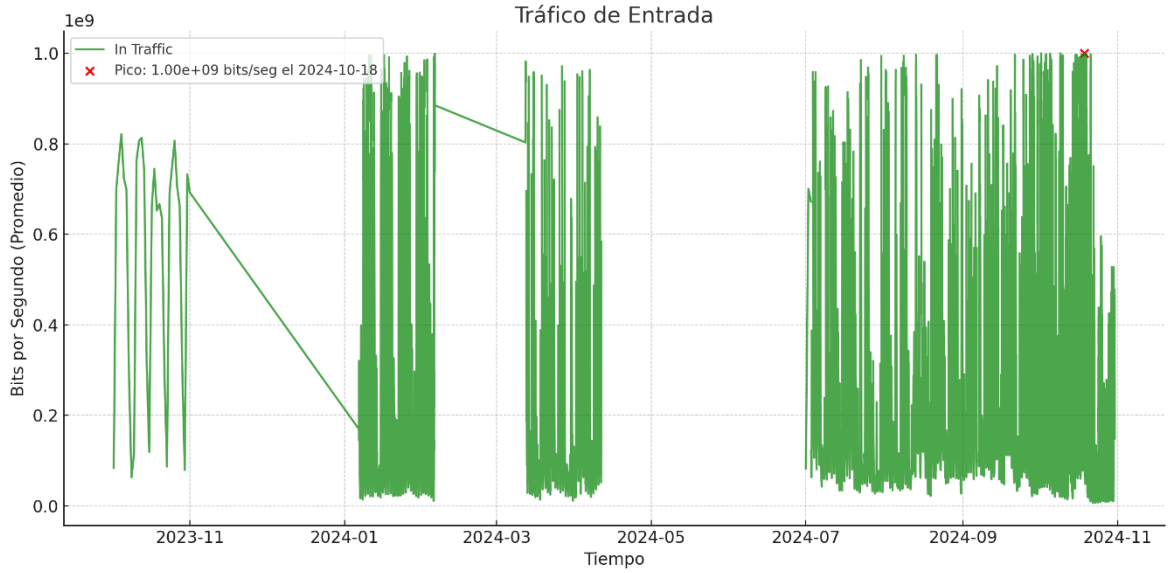
Tráfico de entrada, salida y total en la Universidad Nacional de Ingeniería entre el 1 de octubre de 2023 y el 30 de octubre de 2024.



Nota: Elaboración propia (Solier, G., 2025).

Figura 13

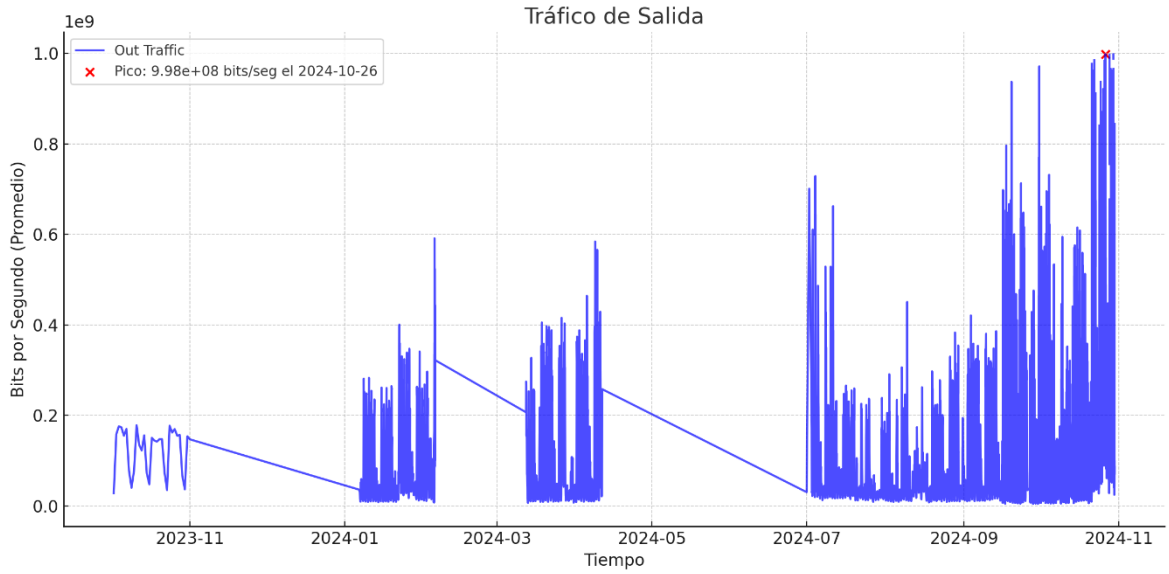
Tráfico de entrada registrado en la Universidad Nacional de Ingeniería entre el 1 de octubre de 2023 y el 30 de octubre de 2024.



Nota: Elaboración propia (Solier, G., 2025).

Figura 14

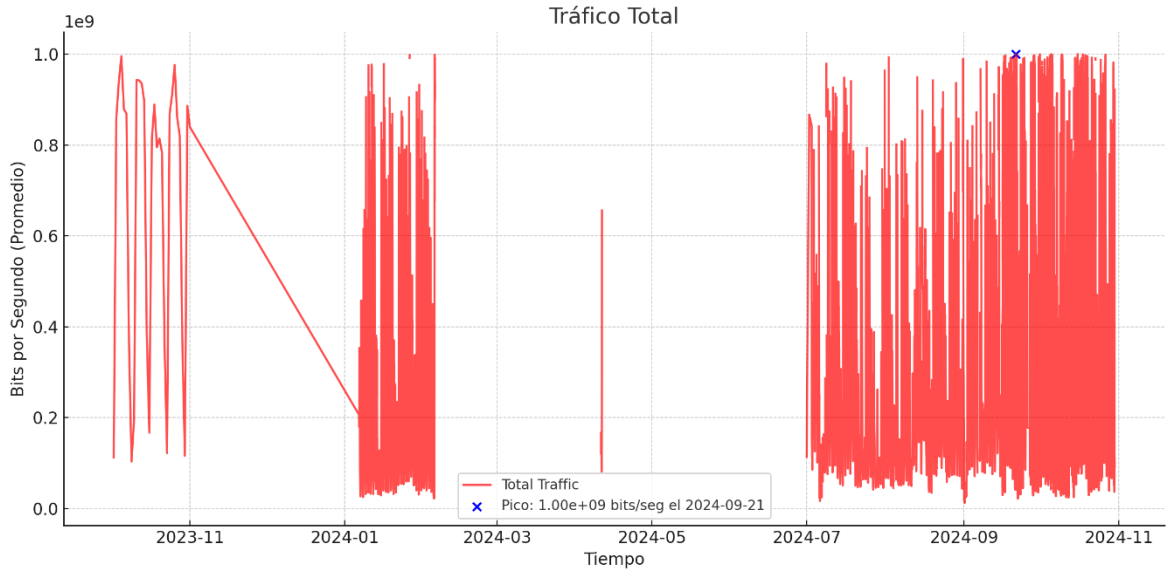
Tráfico de salida registrado en la Universidad Nacional de Ingeniería entre el 1 de octubre de 2023 y el 30 de octubre de 2024.



Nota: Elaboración propia (Solier, G., 2025).

Figura 15

Tráfico Total registrado en la Universidad Nacional de Ingeniería entre el 1 de octubre de 2023 y el 30 de octubre de 2024.



Nota: Elaboración propia (Solier, G., 2025).

Figura 16

Visualización detallada del tráfico de red.

```
CODIGOS PYTHON
# Tráfico de entrada, salida y total
plt.figure(figsize=(14, 8))
plt.plot(in_traffic['Time'], in_traffic['Bits per Second(Avg)'], label='In Traffic', alpha=0.7)
plt.plot(out_traffic['Time'], out_traffic['Bits per Second(Avg)'], label='Out Traffic', alpha=0.7)
plt.plot(total_traffic['Time'], total_traffic['Bits per Second(Avg)'], label='Total Traffic', alpha=0.7)
plt.title('Evolución del Tráfico de Red', fontsize=16)
plt.xlabel('Tiempo', fontsize=12)
plt.ylabel('Bits por Segundo (Promedio)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("Figura1_Tráfico_Combinado.png")
plt.show()
```

```
# In Traffic
plt.figure(figsize=(12, 6))
plt.plot(in_traffic['Time'], in_traffic['Bits per Second(Avg)'], label='In Traffic', color='green', alpha=0.7)
plt.scatter(in_peak['Time'], in_peak['Bits per Second(Avg)'], color='red',
            label=f"Pico: {in_peak['Bits per Second(Avg)']:.2e} bits/seg el {in_peak['Time'].date()}")
plt.title('Tráfico de Entrada', fontsize=16)
plt.xlabel('Tiempo', fontsize=12)
plt.ylabel('Bits por Segundo (Promedio)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("Figura2_Tráfico_Entrada.png")
plt.show()
# Out Traffic
plt.figure(figsize=(12, 6))
plt.plot(out_traffic['Time'], out_traffic['Bits per Second(Avg)'], label='Out Traffic', color='blue', alpha=0.7)
plt.scatter(out_peak['Time'], out_peak['Bits per Second(Avg)'], color='red',
            label=f"Pico: {out_peak['Bits per Second(Avg)']:.2e} bits/seg el {out_peak['Time'].date()}")
plt.title('Tráfico de Salida', fontsize=16)
plt.xlabel('Tiempo', fontsize=12)
plt.ylabel('Bits por Segundo (Promedio)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("Figura3_Tráfico_Salida.png")
plt.show()
# Total Traffic
plt.figure(figsize=(12, 6))
plt.plot(total_traffic['Time'], total_traffic['Bits per Second(Avg)'],
         label='Total Traffic', color='red', alpha=0.7)
plt.scatter(total_peak['Time'], total_peak['Bits per Second(Avg)'], color='blue',
            label=f"Pico: {total_peak['Bits per Second(Avg)']:.2e} bits/seg el {total_peak['Time'].date()}")
plt.title('Tráfico Total', fontsize=16)
plt.xlabel('Tiempo', fontsize=12)
plt.ylabel('Bits por Segundo (Promedio)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("Figura4_Tráfico_Total.png")
plt.show()
```

Nota: Este código genera gráficos para analizar la evolución del tráfico de red (entrada, salida y total) a lo largo del tiempo, identificando y destacando los picos. Las figuras se guardan como imágenes para su inclusión en el documento. Elaboración propia (Solier, G., 2025)

3.3.6 Paso 1: Completar Datos Vacíos

Según Cristian Villarroya Sánchez en su trabajo "Modelo basado en redes neuronales para la predicción de tráfico en la ciudad de Valencia" (2020-2021), el primer paso en el proceso de limpieza de datos consiste en solucionar la falta de lecturas incompletas para días en los que existen datos parciales. Este procedimiento se realiza en tres etapas:

- Interpolación Lineal: Los valores faltantes se interpolan utilizando los valores inmediatamente anterior y posterior.
- Relleno Hacia Adelante: En secuencias donde falten múltiples lecturas consecutivas, los valores se rellenan con el último valor disponible.
- Relleno Hacia Atrás: Para cualquier dato restante faltante, se utiliza el valor posterior más cercano

3.3.6.1 Identificación de Valores Faltantes

Figura 17

Verificación de valores faltantes en los datos de tráfico.

```
# Revisar valores faltantes en cada DataFrame
print("Valores faltantes antes de la limpieza:")
print(f"InTraffic: {in_traffic['Bits per Second(Avg)'].isnull().sum()} valores faltantes")
print(f"OutTraffic: {out_traffic['Bits per Second(Avg)'].isnull().sum()} valores faltantes")
print(f"TotalTraffic: {total_traffic['Bits per Second(Avg)'].isnull().sum()} valores faltantes")
```

Nota: Este código identifica la cantidad de valores faltantes en los datos de tráfico antes de la limpieza.

Elaboración propia (Solier, G., 2025).

Tabla 7

Valores faltantes antes de la limpieza

InTraffic	7969 valores faltantes
OutTraffic	988 valores faltantes
TotalTraffic	988 valores faltantes

Nota: La tabla muestra los valores faltantes por cada categoría de datos.
Elaboración propia (Solier, G., 2025).

3.3.6.2 Aplicación de Interpolación y Relleno

Figura 18

Interpolación y relleno de datos faltantes.

```
# Interpolación lineal y relleno en los tres DataFrames
for traffic in [in_traffic, out_traffic, total_traffic]:
# Interpolación lineal
traffic['Bits per Second(Avg)'] = traffic['Bits per Second(Avg)'].interpolate()
# Relleno hacia adelante y hacia atrás
traffic['Bits per Second(Avg)'] = traffic['Bits per Second(Avg)'].fillna(method='ffill')
traffic['Bits per Second(Avg)'] = traffic['Bits per Second(Avg)'].fillna(method='bfill')
```

Nota: Este código aplica interpolación lineal y relleno hacia adelante y atrás para manejar valores faltantes en los datos de tráfico. Elaboración propia (Solier, G., 2025).

3.3.6.3 Revisar valores faltantes después de la limpieza

Figura 19

Verificación de valores faltantes después de la limpieza.

```
print("Valores faltantes después de la limpieza:")
print(f"InTraffic: {in_traffic['Bits per Second(Avg)'].isnull().sum()} valores faltantes")
print(f"OutTraffic: {out_traffic['Bits per Second(Avg)'].isnull().sum()} valores faltantes")
print(f"TotalTraffic: {total_traffic['Bits per Second(Avg)'].isnull().sum()} valores faltantes")
```

Nota: Este código verifica y reporta los valores faltantes en los datos de tráfico después del proceso de limpieza. Elaboración propia (Solier, G., 2025).

Tabla 8

Valores faltantes después de la limpieza

InTraffic	0 valores faltantes
OutTraffic	0 valores faltantes
TotalTraffic	0 valores faltantes

Nota: La tabla muestra los valores faltantes por cada categoría de datos después de haber hecho la limpieza. Elaboración propia (Solier, G., 2025).

3.3.6.4 Validación Gráfica

Tras aplicar este paso, se genera una gráfica para verificar que los valores faltantes han sido correctamente interpolados y rellenados. Tal grafica se muestra en la Figura 21, Figura 22 y Figura 23 para In traffic, Out traffic y Total Traffic respectivamente.

Figura 20

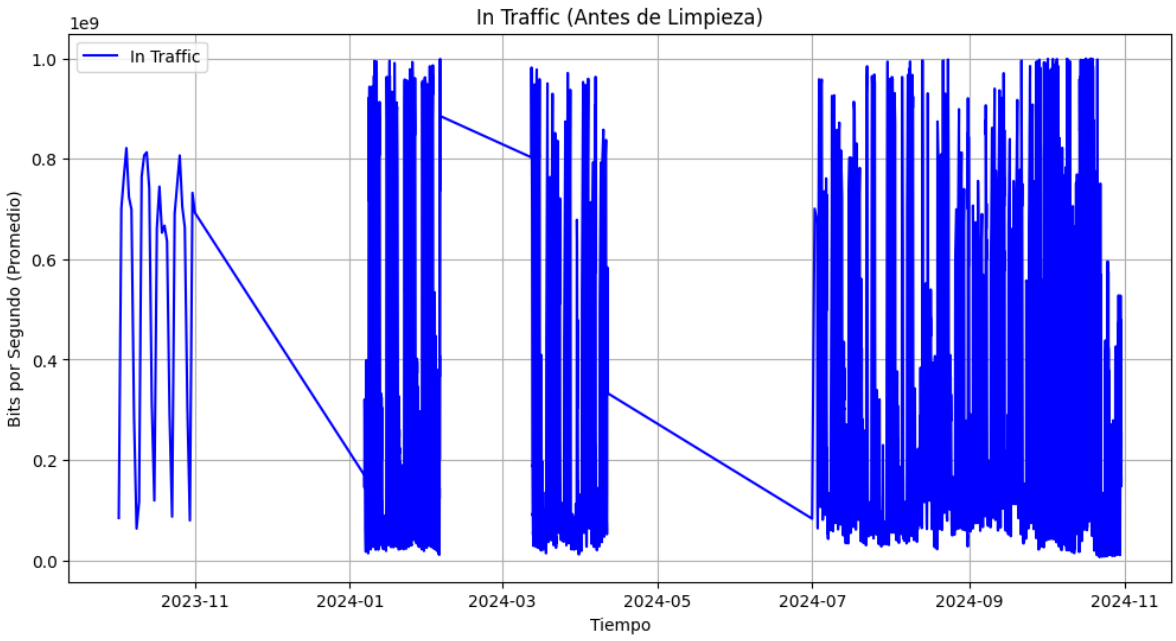
Gráficas individuales del tráfico antes de la limpieza.

```python
import matplotlib.pyplot as plt
# Función para graficar
def plot_traffic(data, title, label, color):
plt.figure(figsize=(12, 6))
plt.plot(data['Time'], data['Bits per Second(Avg)'], label=label, color=color)
plt.title(title)
plt.xlabel('Tiempo')
plt.ylabel('Bits por Segundo (Promedio)')
plt.legend()
plt.grid(True)
plt.show()
# Gráficas individuales
plot_traffic(in_traffic, 'In Traffic (Antes de Limpieza)', 'In Traffic', 'blue')
plot_traffic(out_traffic, 'Out Traffic (Antes de Limpieza)', 'Out Traffic', 'orange')
plot_traffic(total_traffic, 'Total Traffic (Antes de Limpieza)', 'Total Traffic', 'green')
```

Nota: Este código utiliza una función personalizada para graficar el tráfico de red (entrada, salida y total) antes de aplicar el proceso de limpieza. Elaboración propia (Solier, G., 2025).

Figura 21

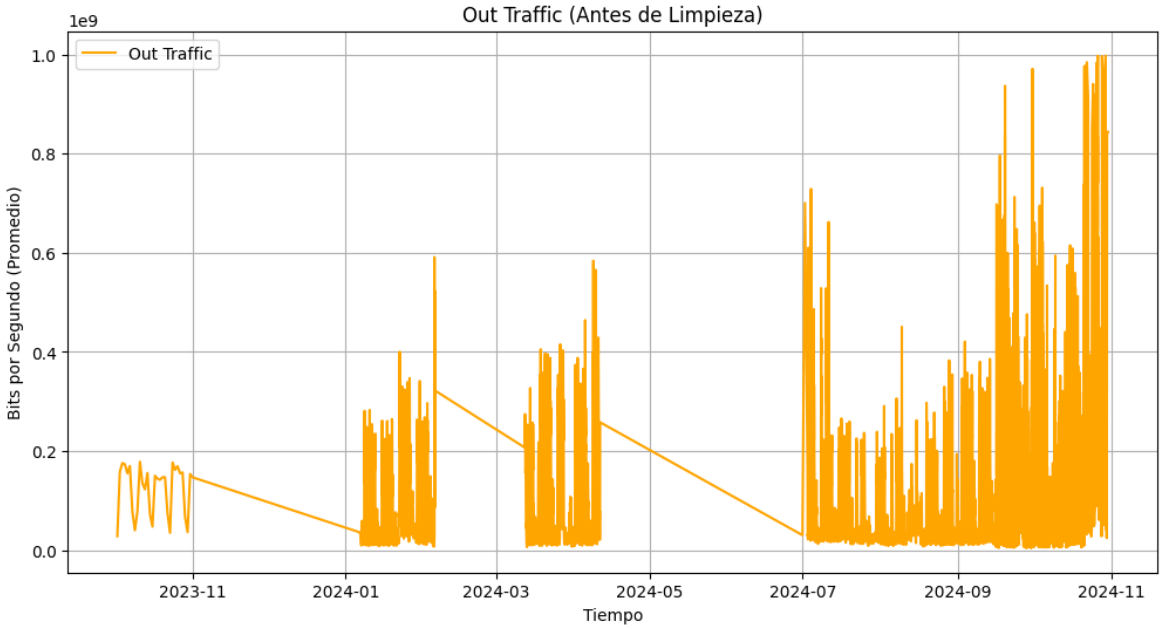
In Traffic (Antes de Limpieza).



Nota: Elaboración propia (Solier, G., 2025).

Figura 22

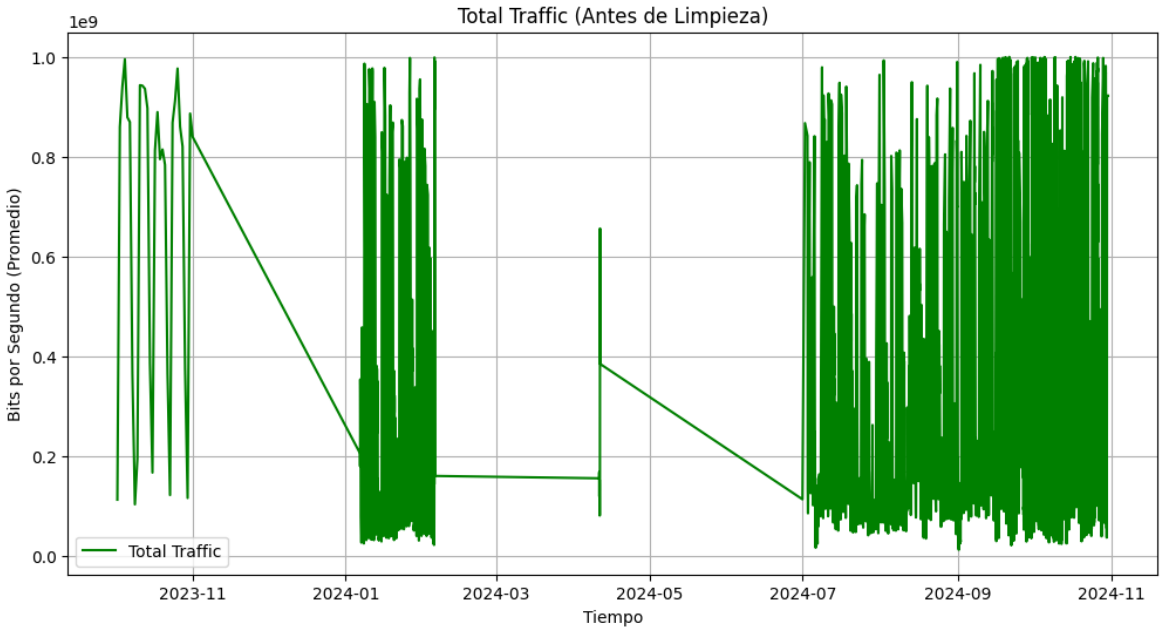
Out Traffic (Antes de Limpieza).



Nota: Elaboración propia (Solier, G., 2025).

Figura 23

Total Traffic (Antes de Limpieza)



Nota: Elaboración propia (Solier, G., 2025).

3.3.7 Paso 2: Agregar Datos por Hora

Según Cristian Villarroya Sánchez en su trabajo "Modelo basado en redes neuronales para la predicción de tráfico en la ciudad de Valencia" (2021), la agregación de datos de tráfico por intervalos horarios es fundamental para reducir la variabilidad no representativa y homogenizar la serie temporal. Este paso utiliza técnicas de resampleo y cálculo de promedios para transformar lecturas originales cada 15 minutos en valores horarios.

Antes de realizar el resampleo, es crucial garantizar que los datos estén en un formato numérico adecuado. Los valores de tráfico originalmente incluyen el sufijo M (millones) y pueden contener comas que dificultan las operaciones numéricas. Este procedimiento incluye:

3.3.7.1 Conversión de Valores de Texto a Números:

1. Se eliminan los sufijos (M) y las comas.
2. Los valores resultantes se multiplican por 10^6 para convertirlos a bits por segundo.

3.3.7.2 Relleno de Valores Faltantes:

1. Los valores nulos se interpolan o rellenan hacia adelante y hacia atrás para garantizar consistencia.

Figura 24

Conversión y limpieza de datos en el tráfico total.

```
# Convertir valores a texto y limpiar formato
total_traffic['Bits per Second(Avg)'] = total_traffic['Bits per Second(Avg)'].astype(str)
# Eliminar "M" y comas, luego convertir a números
total_traffic['Bits per Second(Avg)'] = pd.to_numeric(
total_traffic['Bits per Second(Avg)'].str.replace(' M', '', regex=False).str.replace(',', '', regex=False),
errors='coerce') * 1e6
# Manejar valores nulos
total_traffic['Bits per Second(Avg)'] = total_traffic['Bits per Second(Avg)'].fillna(method='ffill').fillna(method='bfill')
```

Nota: Este código convierte los valores de tráfico total a formato numérico, elimina caracteres no deseados y maneja valores nulos con relleno. Elaboración propia (Solier, G., 2025).

3.3.7.3 Resampleo por Hora

1. Conversión del Tiempo: Se asegura que la columna Time esté en formato datetime.
2. Configuración del Índice: Se establece la columna Time como índice del DataFrame.
3. Agrupación por Intervalos Horarios: Se utiliza resample('H') para calcular el promedio de lecturas dentro de cada hora.

El código Python se muestra en la Figura 25 y el Resampleo en la Figura 26.

Figura 25

Resampleo y promediado del tráfico total por hora.

```
# Convertir tiempo a datetime y establecer como índice
total_traffic['Time'] = pd.to_datetime(total_traffic['Time'])
total_traffic.set_index('Time', inplace=True)
# Resampleo por hora con promedio
hourly_traffic = total_traffic.resample('H').mean().reset_index()
# Revisar resultados
print(hourly_traffic.head())
```

Nota: Este código ajusta el índice temporal y calcula promedios horarios del tráfico total. Elaboración propia (Solier, G., 2025).

Figura 26

Luego de Resampleo por hora.

	Time	Bits per Second(Avg)
0	2023-10-01 23:00:00	112523000.0
1	2023-10-02 00:00:00	NaN
2	2023-10-02 01:00:00	NaN
3	2023-10-02 02:00:00	NaN
4	2023-10-02 03:00:00	NaN

Nota: Elaboración propia (Solier, G., 2025).

3.3.7.4 Visualización Comparativa

Se genera una gráfica para comparar los datos originales (cada 15 minutos) con los datos agregados (por hora). Esto permite visualizar la estabilidad y reducción del ruido en la serie temporal. El código esta detallado en la Figura 27, y la gráfica en la Figura 28.

Figura 27

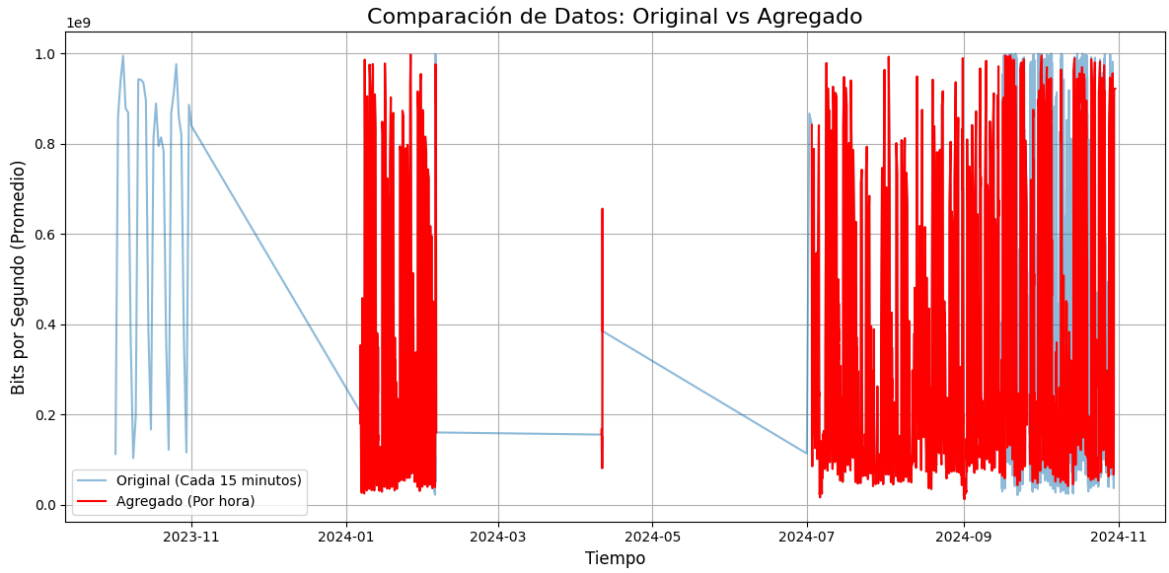
Comparación de datos originales y agregados.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(total_traffic.index, total_traffic['Bits per Second(Avg)'], label='Original (Cada 15 minutos)', alpha=0.5)
plt.plot(hourly_traffic['Time'], hourly_traffic['Bits per Second(Avg)'], label='Agregado (Por hora)', color='red')
plt.title('Comparación de Datos: Original vs Agregado', fontsize=16)
plt.xlabel('Tiempo', fontsize=12)
plt.ylabel('Bits por Segundo (Promedio)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Nota: Este código compara gráficamente el tráfico total original (cada 15 minutos) con los datos agregados por hora. Elaboración propia (Solier, G., 2025).

Figura 28

Comparación de Datos: Original vs Agregado.



Nota: Elaboración propia (Solier, G., 2025).

3.3.7.5 Explicación del Proceso

El procedimiento sigue el enfoque descrito por Villarroya Sánchez (2021), quien explica que:

1. Este proceso reduce la variabilidad no representativa en los datos.
2. Garantiza consistencia al transformar lecturas irregulares (con intervalos inconsistentes) en una serie temporal uniforme con un intervalo fijo.

3.3.7.6 Resultados esperados:

1. Una serie temporal más estable.
2. Reducción del ruido debido a diferencias significativas entre lecturas consecutivas.

3.3.8 Paso 3: Completar Días Faltantes

En el trabajo de Cristian Villarroya Sánchez "Modelo basado en redes neuronales para la predicción de tráfico en la ciudad de Valencia" (2021), se detalla que, tras completar los valores horarios, el siguiente paso es manejar los días completos faltantes. Estos pueden surgir debido a errores en la recolección de datos o fallos del sistema de captura. En lugar de eliminar dichos días, que afectaría la continuidad de la serie temporal, se decidió rellenarlos utilizando lecturas correspondientes a un día similar (por ejemplo, un lunes faltante se completa con datos de otro lunes).

A continuación, se describe cómo aplicar este paso basado en la metodología del Cristian Villarroya Sánchez:

3.3.8.1 Identificar los Días Faltantes

El objetivo es detectar los días que están completamente ausentes dentro del rango de fechas del conjunto de datos. Esto se realiza generando una lista de todos los días en el rango de fechas y comparándola con los días únicos disponibles en los datos.

Para identificar los días faltantes:

1. Crear un rango de fechas completo desde el inicio hasta el fin de los datos.
2. Comparar este rango con los días únicos disponibles en los datos.

Figura 29

Identificación de días faltantes en los datos

```
# Identificar días faltantes
missing_days = pd.date_range(
    start=hourly_traffic['Time'].min(),
    end=hourly_traffic['Time'].max(),
    freq='D'
).difference(hourly_traffic['Time'].dt.date.unique())

print("Días faltantes:", missing_days)
```

Nota: La figura muestra el código usado para identificar días faltantes. Elaboración propia (Solier, G., 2025).

Figura 30

Identificar días faltantes

```
Días faltantes: DatetimeIndex(['2023-10-01 23:00:00', '2023-10-02 23:00:00',
               '2023-10-03 23:00:00', '2023-10-04 23:00:00',
               '2023-10-05 23:00:00', '2023-10-06 23:00:00',
               '2023-10-07 23:00:00', '2023-10-08 23:00:00',
               '2023-10-09 23:00:00', '2023-10-10 23:00:00',
               ...
               '2024-10-20 23:00:00', '2024-10-21 23:00:00',
               '2024-10-22 23:00:00', '2024-10-23 23:00:00',
               '2024-10-24 23:00:00', '2024-10-25 23:00:00',
               '2024-10-26 23:00:00', '2024-10-27 23:00:00',
               '2024-10-28 23:00:00', '2024-10-29 23:00:00'],
              dtype='datetime64[ns]', length=395, freq='D')
```

Nota: La figura muestra los días faltantes que se identificaron en los datos
Elaboración propia (Solier, G., 2025).

3.3.8.2 Rellenar Días Faltantes

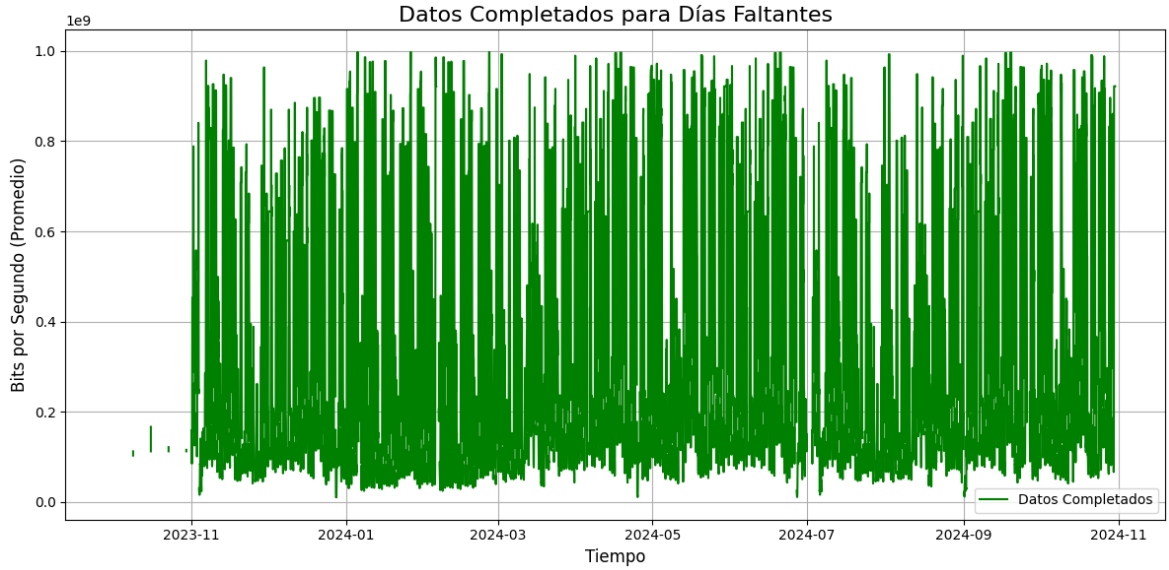
Por cada día faltante identificado:

1. Se busca un día con el mismo día de la semana dentro de los datos disponibles (p. ej., un lunes).
2. Se utiliza este día como reemplazo para el día faltante.

En la Figura 31, Figura 32 y Figura 33 se muestra la gráfica de los datos completos desde 01 octubre del 2023 hasta 31 de Octubre del 2024.

Figura 31

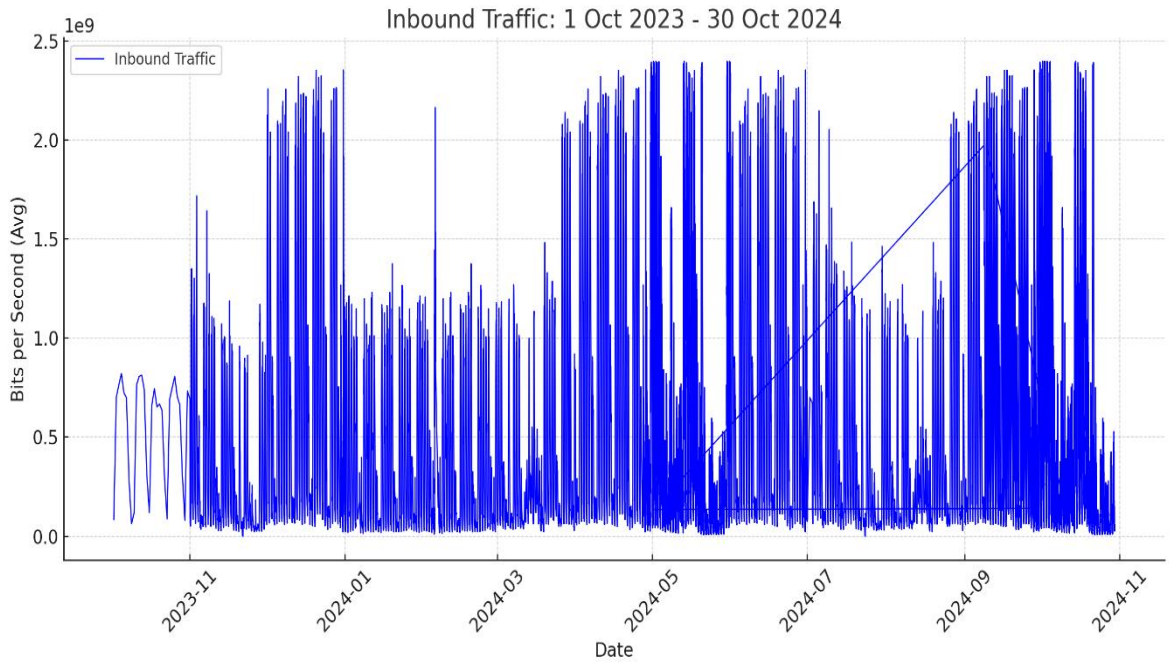
Completar Días Faltantes: Datos Completados con días faltantes TotalTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 32

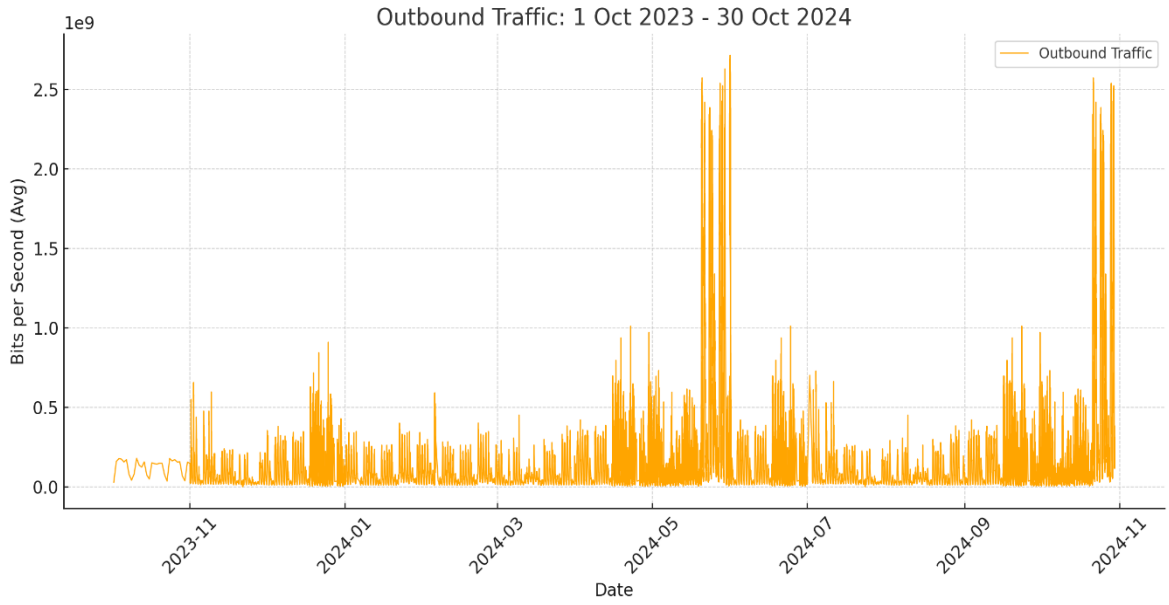
Completar Días Faltantes: Datos Completados con días faltantes InTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 33

Completar Días Faltantes: Datos Completados con días faltantes OutTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Justificación del Método

Según Villarroya Sánchez (2021), este enfoque garantiza:

1. Continuidad Temporal: Mantener una serie temporal uniforme es crucial para entrenar modelos predictivos como redes neuronales.
2. Representatividad de Datos: Al completar días faltantes con días similares, se preserva la coherencia de los patrones estacionales o semanales en los datos.
3. Evitar Eliminaciones: Eliminar días faltantes puede resultar en una pérdida de información valiosa y dificultar la generalización del modelo.

3.3.9 Paso 4: Corregir Lecturas Erróneas

En el trabajo de Cristian Villarroya Sánchez "Modelo basado en redes neuronales para la predicción de tráfico en la ciudad de Valencia" (2021), el último paso en la preparación de los datos consiste en corregir lecturas erróneas o repetitivas. Estas lecturas suelen ser valores constantes durante todo un día, lo cual indica un fallo en el sistema de captura. Para solucionar este problema, dichas lecturas se reemplazan con datos correspondientes a un día similar de la misma semana (por ejemplo, un lunes repetitivo se reemplaza con lecturas de otro lunes).

3.3.9.1 Identificación de Lecturas Repetitivas

Se detectan valores duplicados dentro del mismo día. Se considera una lectura repetitiva si aparece más de 6 veces iguales en el mismo día, visible en la Figura 34.

Figura 34

Identificación de Lecturas Repetitivas.

```
# Detectar lecturas repetitivas
hourly_traffic['Duplicate'] = hourly_traffic.duplicated(subset=['Bits per Second(Avg)'], keep=False)

# Verificar lecturas repetitivas
print("Lecturas repetitivas detectadas:")
print(hourly_traffic[hourly_traffic['Duplicate']].head())
```

```
Lecturas repetitivas detectadas:
                  Time  Bits per Second(Avg)  Duplicate
0    2023-10-01 23:00:00          112523000.0       True
9476 2023-10-01 23:00:00          112523000.0       True
1    2023-10-02 00:00:00                  NaN       True
2    2023-10-02 01:00:00                  NaN       True
3    2023-10-02 02:00:00                  NaN       True
```

Nota: Elaboración propia (Solier, G., 2025).

3.3.9.2 Reemplazo de Lecturas Erróneas

Las lecturas erróneas se reemplazan por valores correspondientes a un día similar (mismo día de la semana) que no tenga errores. El código esta desarrollado a continuación en la Figura 35.

Figura 35

Reemplazo de lecturas repetitivas en los datos

```
# Reemplazar lecturas repetitivas
for index, row in hourly_traffic[hourly_traffic['Duplicate']].iterrows():
# Buscar un día similar sin duplicados
similar_day = hourly_traffic[(hourly_traffic['Time'].dt.dayofweek == row['Time'].dayofweek) & (~hourly_traffic['Duplicate'])].iloc[0]
# Reemplazar el valor erróneo
 hourly_traffic.loc[index, 'Bits per Second(Avg)'] = similar_day['Bits per Second(Avg)']
# Eliminar la columna de duplicados
hourly_traffic.drop(columns=['Duplicate'], inplace=True)
```

Nota: Este código identifica y reemplaza valores duplicados en los datos horarios utilizando valores de días similares. Elaboración propia (Solier, G., 2025).

3.3.9.3 Gráfica Final

Se genera una gráfica para mostrar los datos corregidos y verificar visualmente que las lecturas repetitivas han sido eliminadas. Como en el código Python de la Figura 36. Y la gráfica de los datos limpios y corregidos se muestran en la Figura 37, Figura 38 y Figura 40 para InTraffic, Out Traffic y Total Traffic.

Figura 36

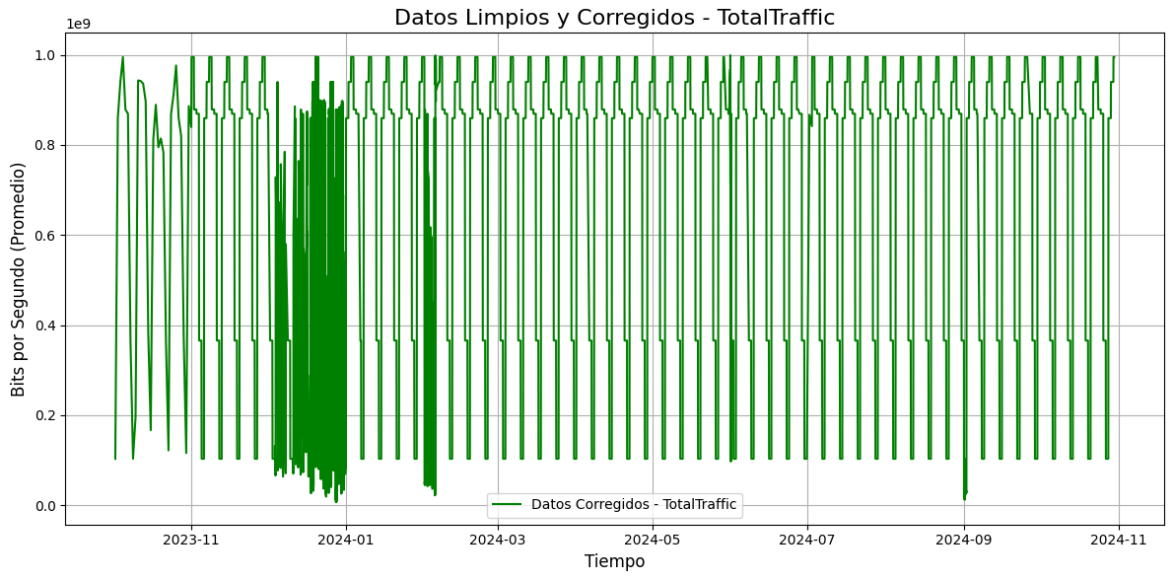
Visualización de datos limpios y corregidos

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(hourly_traffic['Time'], hourly_traffic['Bits per Second(Avg)'], label='Datos Corregidos', color='red')
plt.title('Datos Limpios y Corregidos', fontsize=16)
plt.xlabel('Tiempo', fontsize=12)
plt.ylabel('Bits por Segundo (Promedio)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Nota: Este código grafica los datos de tráfico después del proceso de limpieza y corrección. Elaboración propia (Solier, G., 2025).

Figura 37

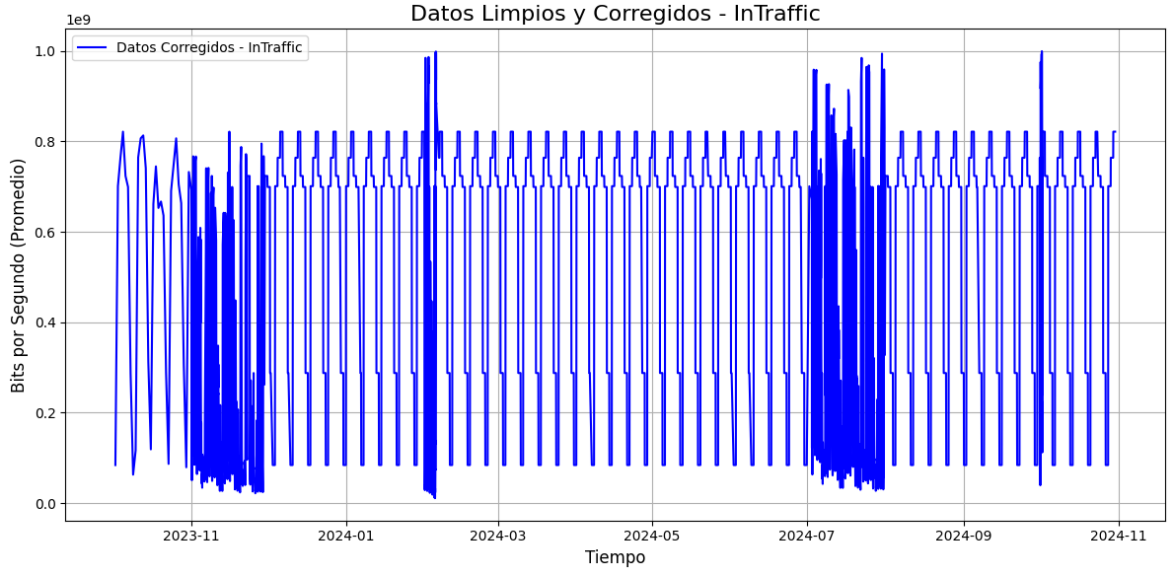
Datos Limpios y Corregidos TotalTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 38

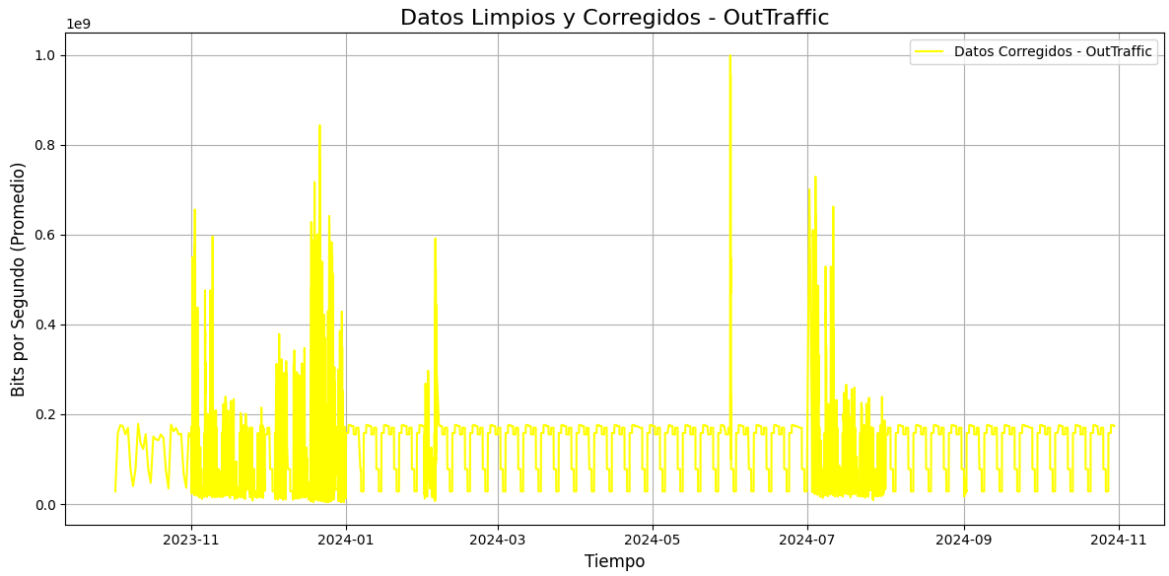
Datos Limpios y Corregidos InTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 39

Datos Limpios y Corregidos OutTraffic.



Nota: Elaboración propia (Solier, G., 2025).

3.3.9.4 Justificación del Método

Este procedimiento sigue los lineamientos descritos por Villarroya Sánchez (2021), que destacan:

1. Detección de Fallos: La repetición de lecturas constantes durante todo un día es indicativa de errores en el sistema de captura o la API.
2. Corrección Basada en Patrones Similares: Reemplazar los datos erróneos con valores de días similares asegura la continuidad y coherencia de los patrones temporales.
3. Evitar la Eliminación de Días Completos: En lugar de descartar los días con errores, se preserva su estructura temporal mediante reemplazos apropiados.

3.3.9.5 Resultados Esperados

1. Eliminación de Lecturas Erróneas: Los valores repetitivos se sustituyen con datos representativos.
2. Consistencia Temporal: Los datos corregidos mantienen la estructura estacional y semanal.

3. Base Limpia para el Modelado: Los datos están preparados para ser utilizados en la fase de modelado predictivo.

3.3.10 Análisis de Estacionalidad, Estacionariedad y Autocorrelación

3.3.10.1 Descripción General

El análisis de series temporales tiene como objetivo principal comprender las características de los datos para su posterior modelado. En este caso, se evaluaron las series temporales InTraffic, OutTraffic y TotalTraffic, correspondientes al tráfico de datos en la red de la Universidad Nacional de Ingeniería. El análisis se centró en tres aspectos clave:

1. Estacionalidad: Identifica patrones recurrentes en los datos.
2. Estacionariedad: Determina si las propiedades estadísticas (media, varianza) son constantes en el tiempo.
3. Autocorrelación: Mide la relación entre valores actuales y pasados para identificar rezagos relevantes.

Los análisis se realizaron para las tres series temporales disponibles: InTraffic, OutTraffic, y TotalTraffic, considerando frecuencias diaria y semanal.

3.3.10.2 Estacionalidad

La estacionalidad representa fluctuaciones recurrentes sobre una tendencia subyacente en las series temporales. Este análisis descompone la serie en tres componentes:

1. Tendencia: Cambios a largo plazo.
2. Estacionalidad: Fluctuaciones periódicas.
3. Residuo: Variabilidad no explicada por la tendencia ni la estacionalidad.

3.3.10.2.1 Método:

Se utilizó el método seasonal_decompose de la biblioteca statsmodels. Los resultados se analizaron en frecuencias diaria y semanal.

3.3.10.2.2 Resultados InTaffic, OutTraffic y TotalTraffic

3.3.10.2.2.1 InTraffic

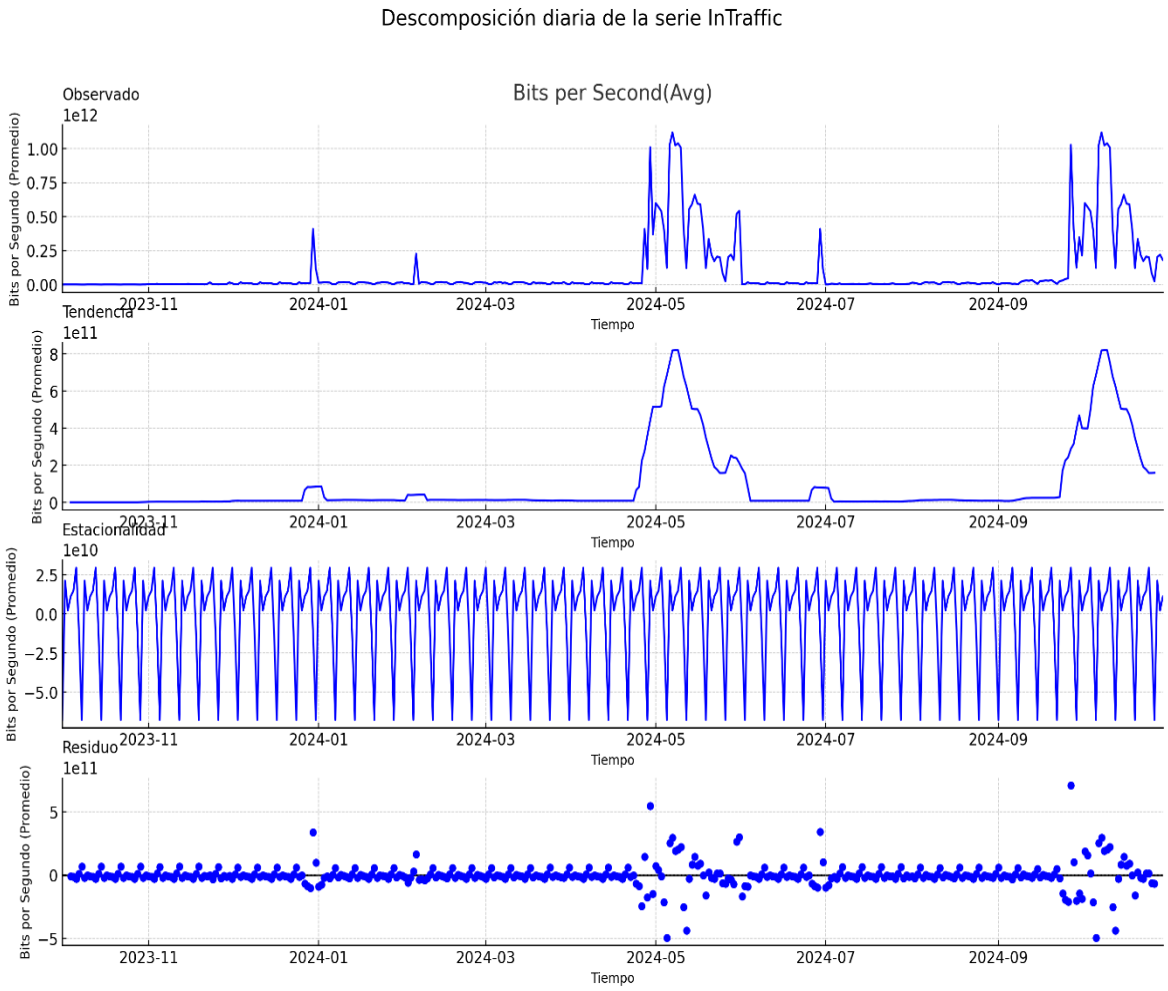
Frecuencia diaria:

1. Tendencia: Incremento moderado a lo largo del tiempo.
2. Estacionalidad: Fluctuaciones cíclicas, con picos durante días laborales.
3. Residuo: Variabilidad uniforme, indicando un buen ajuste del modelo.

En la Figura 40 y Figura 41 se muestra la gráfica de la descomposición diaria y semanal para In Traffic respectivamente:

Figura 40

Descomposición diaria de la serie.



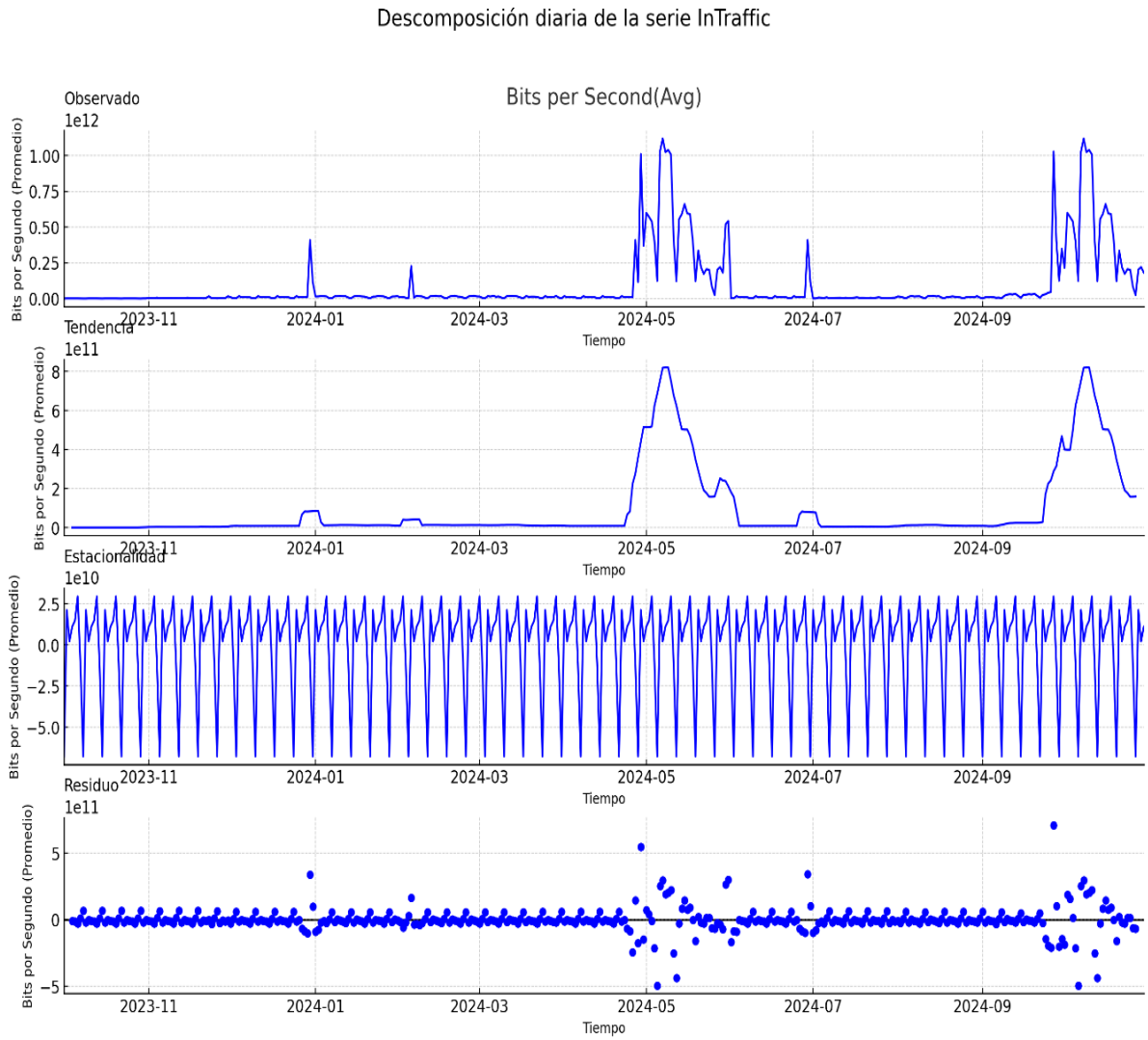
Nota: Elaboración propia (Solier, G., 2025).

Frecuencia semanal:

1. Tendencia: Similar a la frecuencia diaria.
2. Estacionalidad: Ciclos claros correspondientes a días laborales y fines de semana.
3. Residuo: Variaciones menores en comparación con la frecuencia diaria.

Figura 41

Descomposición semanal de la serie InTraffic.



Nota: Elaboración propia (Solier, G., 2025).

3.3.10.2.2.2 OutTraffic

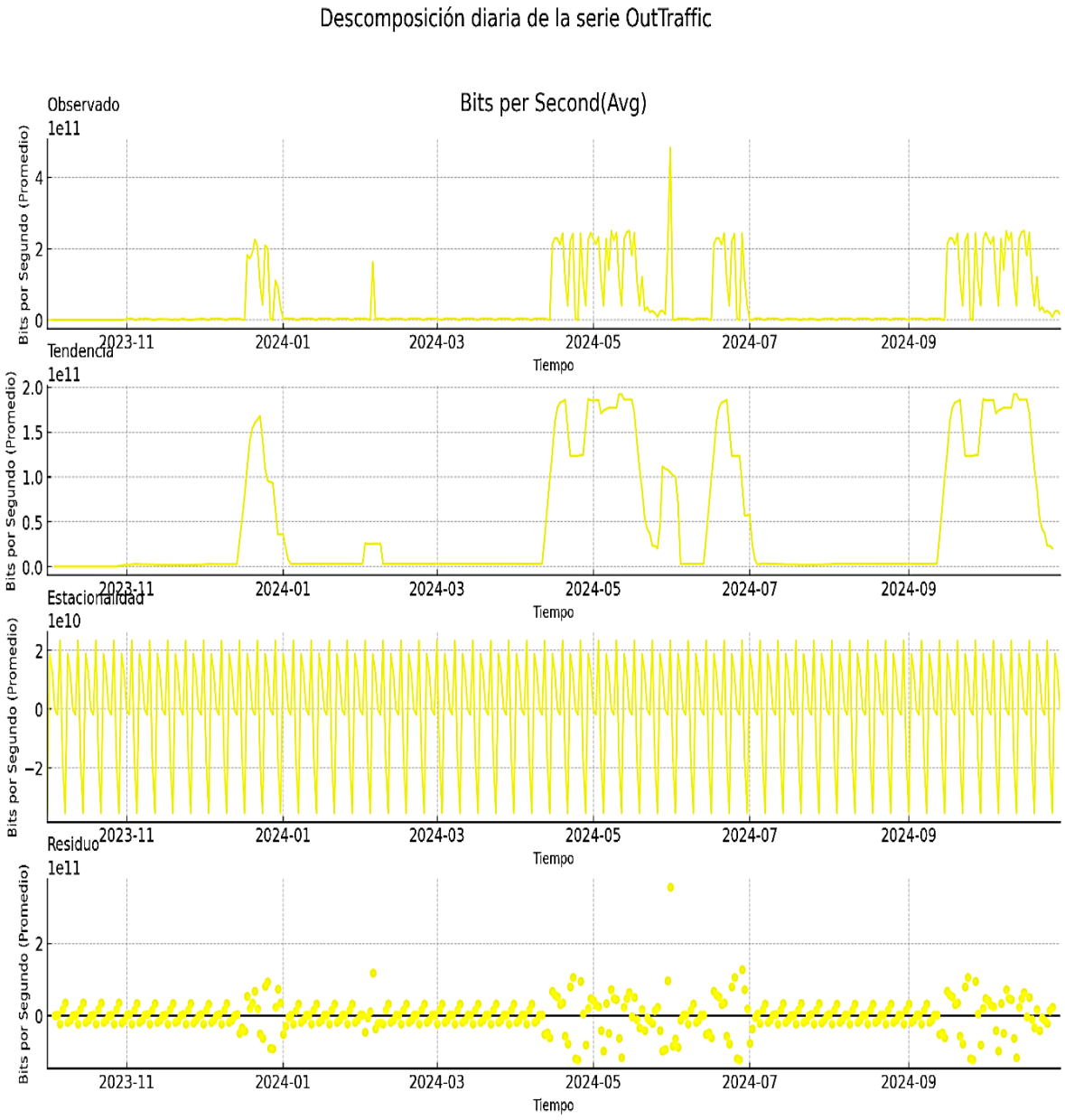
Frecuencia diaria y semanal:

Resultados similares a los de InTraffic, pero con menor intensidad en las fluctuaciones estacionales.

En la Figura 42 y Figura 43 se muestra la gráfica de la descomposición diaria y semanal para Out Traffic respectivamente

Figura 42

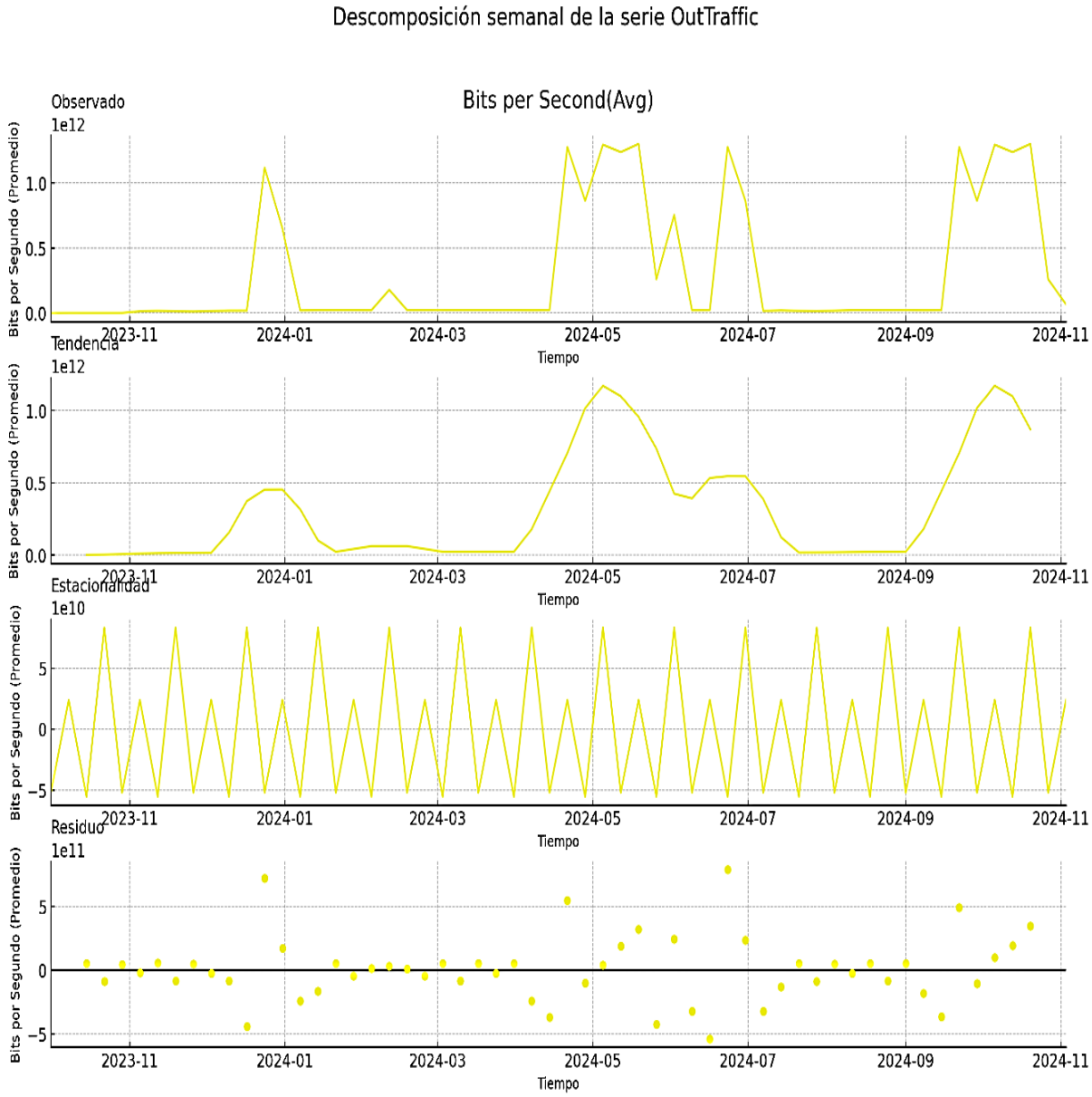
Descomposición diaria de la serie OutTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 43

Descomposición semanal de la serie OutTraffic.



Nota: Elaboración propia (Solier, G., 2025).

3.3.10.2.2.3 TotalTraffic

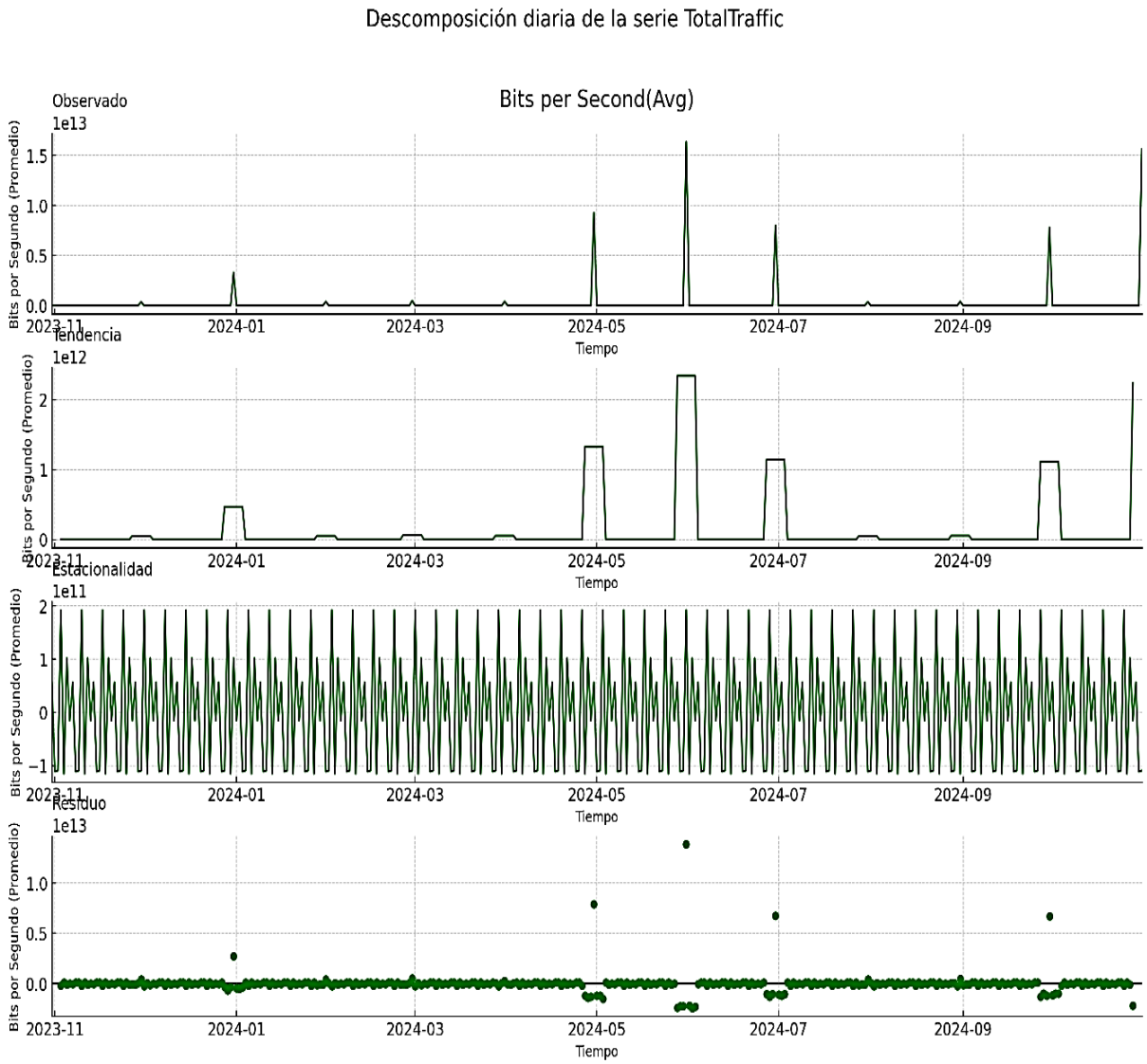
Frecuencia diaria:

1. Tendencia: Incremento general más pronunciado que en las otras series.
2. Estacionalidad: Fluctuaciones regulares, más marcadas durante días laborales.
3. Residuo: Mayor variabilidad en comparación con InTraffic y OutTraffic.

En la Figura 44 y Figura 45 se muestra la gráfica de la descomposición diaria y semanal para Total Traffic respectivamente

Figura 44

Descomposición diaria de la serie TotalTraffic.



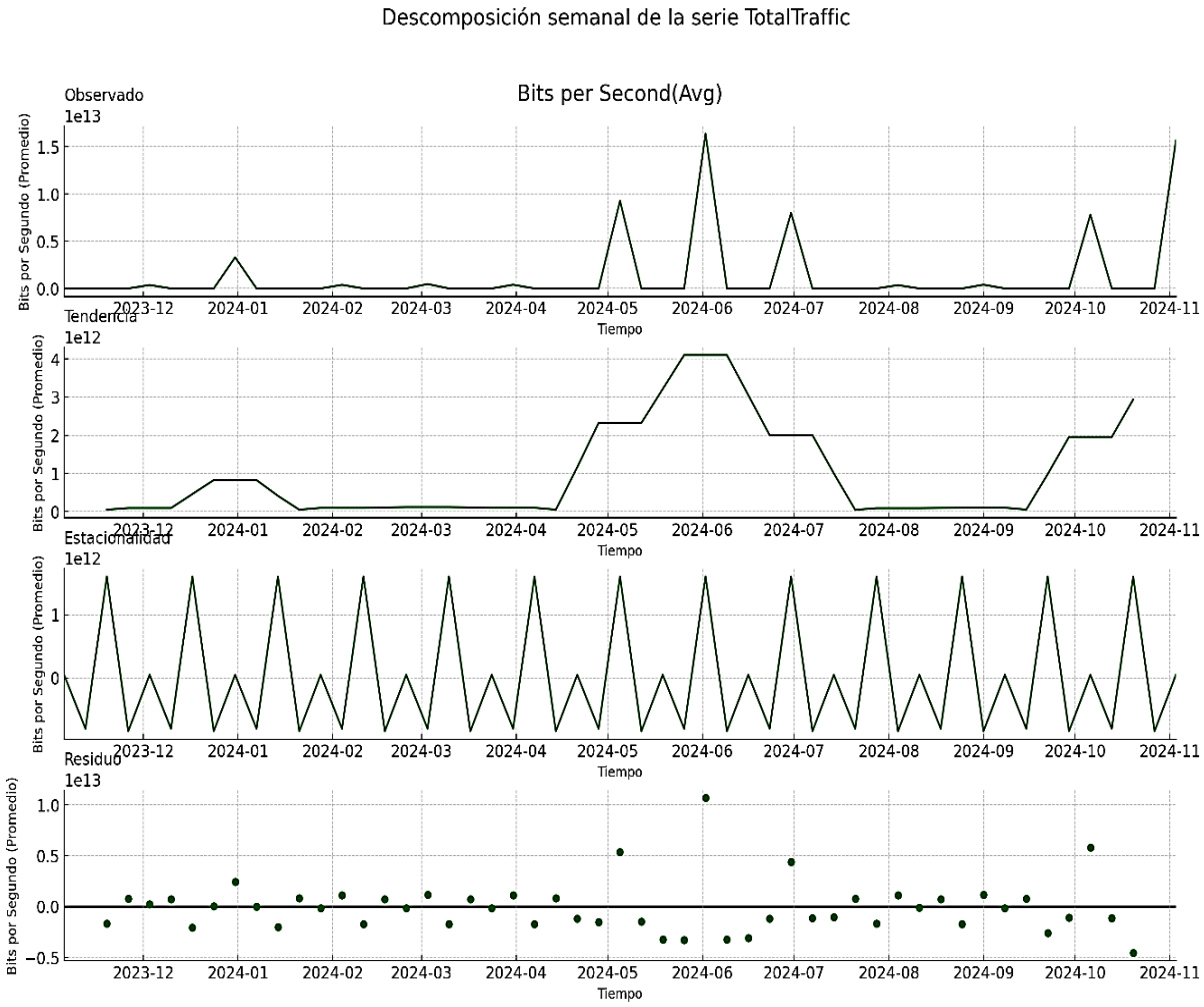
Nota: Elaboración propia (Solier, G., 2025).

Frecuencia semanal:

1. Similar a la frecuencia diaria, con un residuo más uniforme.

Figura 45

Descomposición semanal de la serie TotalTraffic.



Nota: Elaboración propia (Solier, G., 2025).

3.3.10.3 Estacionariedad

La estacionariedad fue evaluada con el Test de Dickey-Fuller Aumentado (ADF) para determinar la presencia de raíces unitarias. Este test es crucial para identificar si las series tienen propiedades estadísticas constantes en el tiempo.

Interpretación:

- p-value < 0.05: La serie es estacionaria.
- ADF Statistic < Valores críticos: La serie es estacionaria.

3.3.10.3.1 Resultados InTaffic, OutTraffic y TotalTraffic

3.3.10.3.1.1 InTraffic

1. Frecuencia diaria:

- ADF Statistic: -4.30
- p-value: 0.0004

Conclusión: La serie es estacionaria.

2. Frecuencia semanal:

- ADF Statistic: -3.98
- p-value: 0.0015

Conclusión: La serie es estacionaria.

3.3.10.3.1.2 OutTraffic

1. Frecuencia diaria:

- ADF Statistic: -3.83
- p-value: 0.0026

Conclusión: La serie es estacionaria.

2. Frecuencia semanal:

- ADF Statistic: -3.78
- p-value: 0.0031

Conclusión: La serie es estacionaria.

3.3.10.3.1.3 TotalTraffic

1. Frecuencia diaria:

- ADF Statistic: -3.46
- p-value: 0.0089

Conclusión: La serie es estacionaria.

2. Frecuencia semanal:

- ADF Statistic: -2.59
- p-value: 0.096

Conclusión: La serie no es estacionaria.

3.3.10.4 Autocorrelación

La autocorrelación mide la relación entre valores actuales y pasados en una serie temporal. Se generaron gráficos de autocorrelación (ACF) y autocorrelación parcial (PACF) para analizar los rezagos significativos y preparar los datos para modelos autoregresivos como se muestra en la Figura 46, Figura 47 y Figura 48.

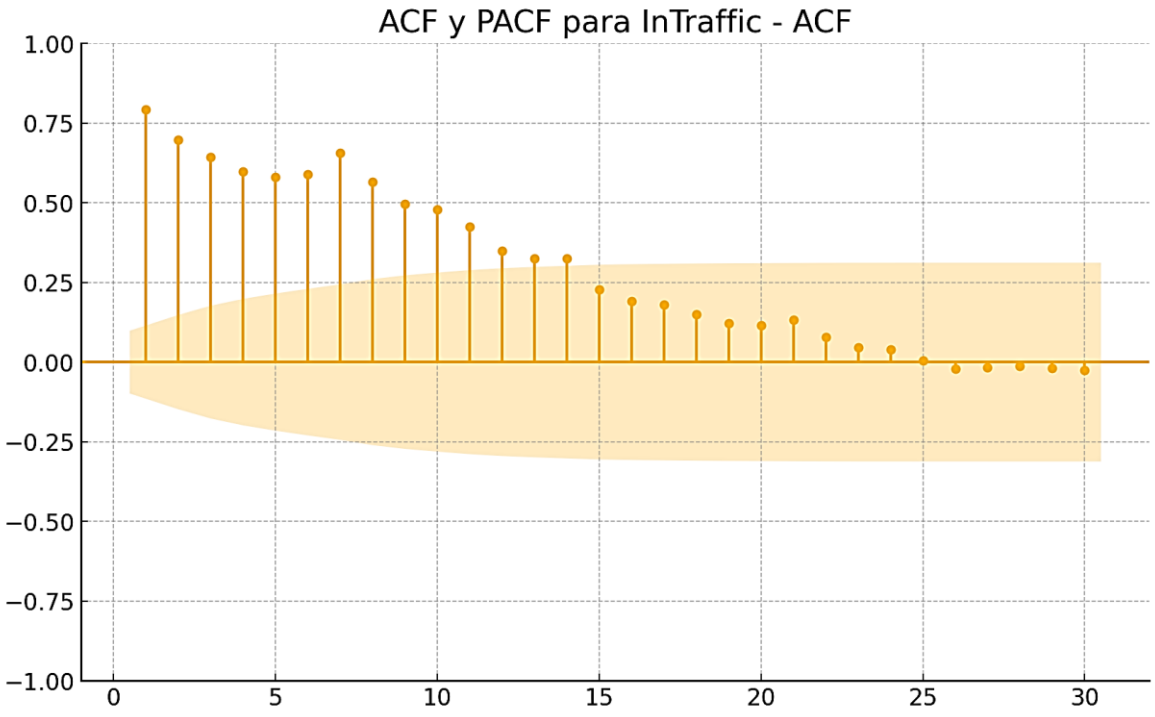
3.3.10.4.1 Resultados lnTaffic, OutTraffic y TotalTraffic

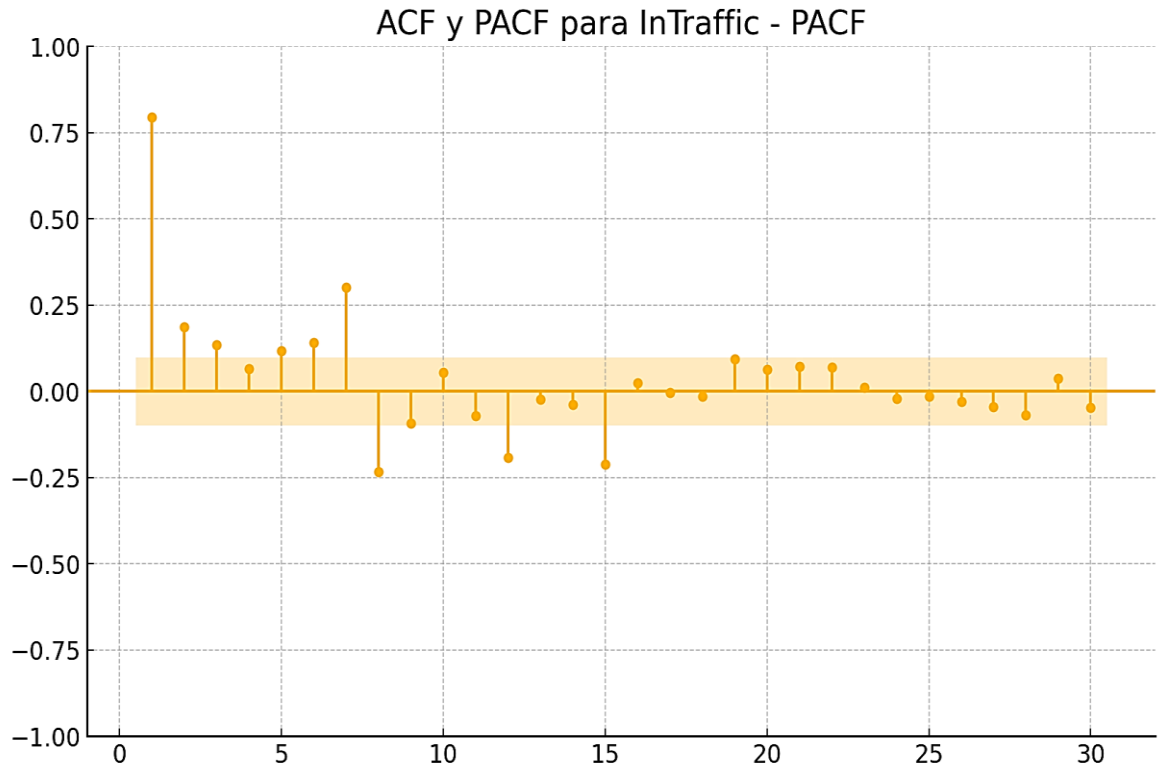
3.3.10.4.1.1 lnTraffic

1. Frecuencia diaria: Correlaciones significativas en los primeros lags (1-7), con patrones decrecientes.
2. Frecuencia semanal: Correlaciones moderadas, disminuyendo más rápido que en frecuencia diaria.

Figura 46

ACF y PACF para InTraffic.





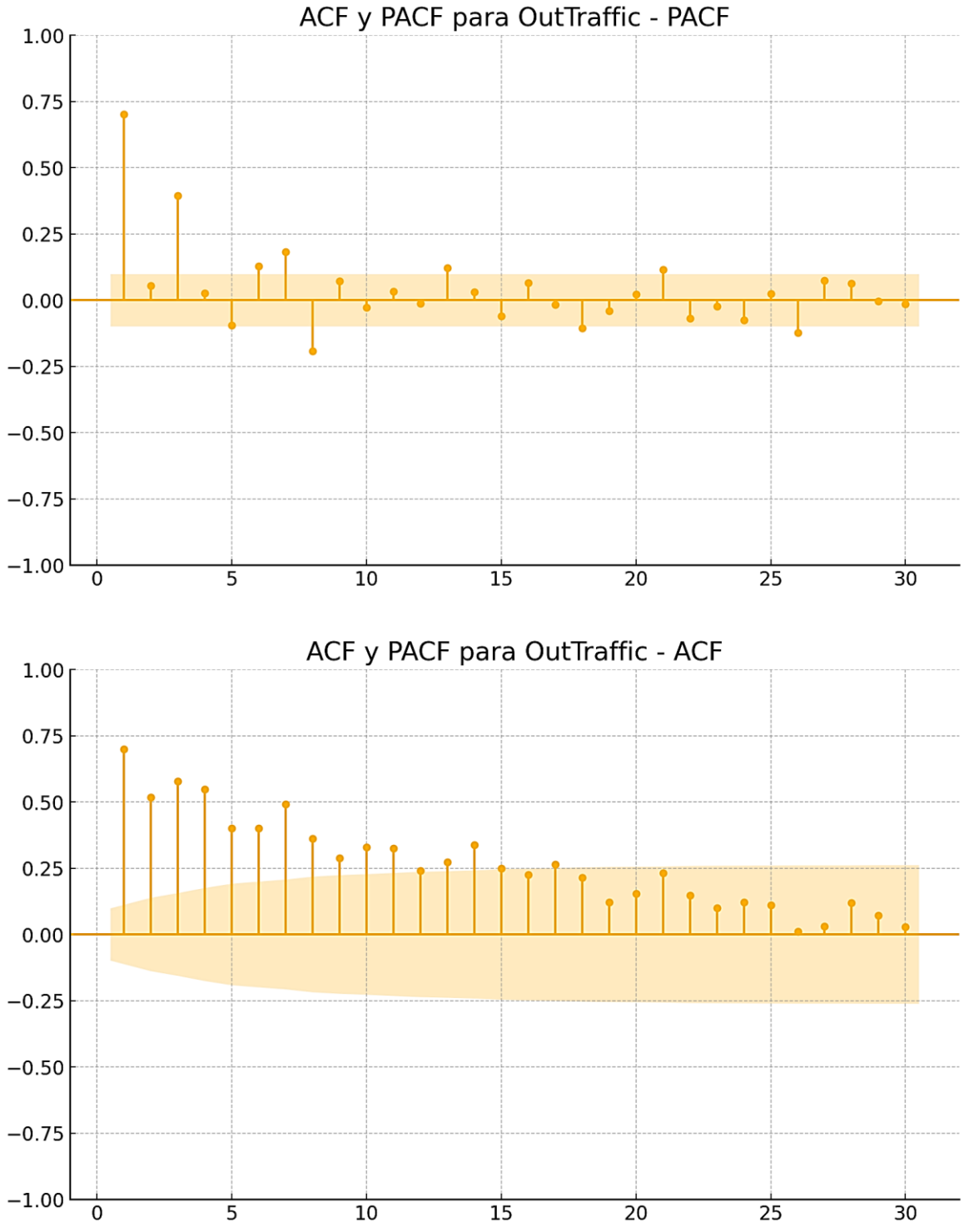
Nota: Elaboración propia (Solier, G., 2025).

3.3.10.4.1.2 OutTraffic

1. Frecuencia diaria: Similar a InTraffic, pero con menor intensidad en los lags posteriores.
2. Frecuencia semanal: Correlaciones significativas en los primeros lags, mostrando menor variabilidad.

Figura 47

ACF y PACF para OutTraffic.



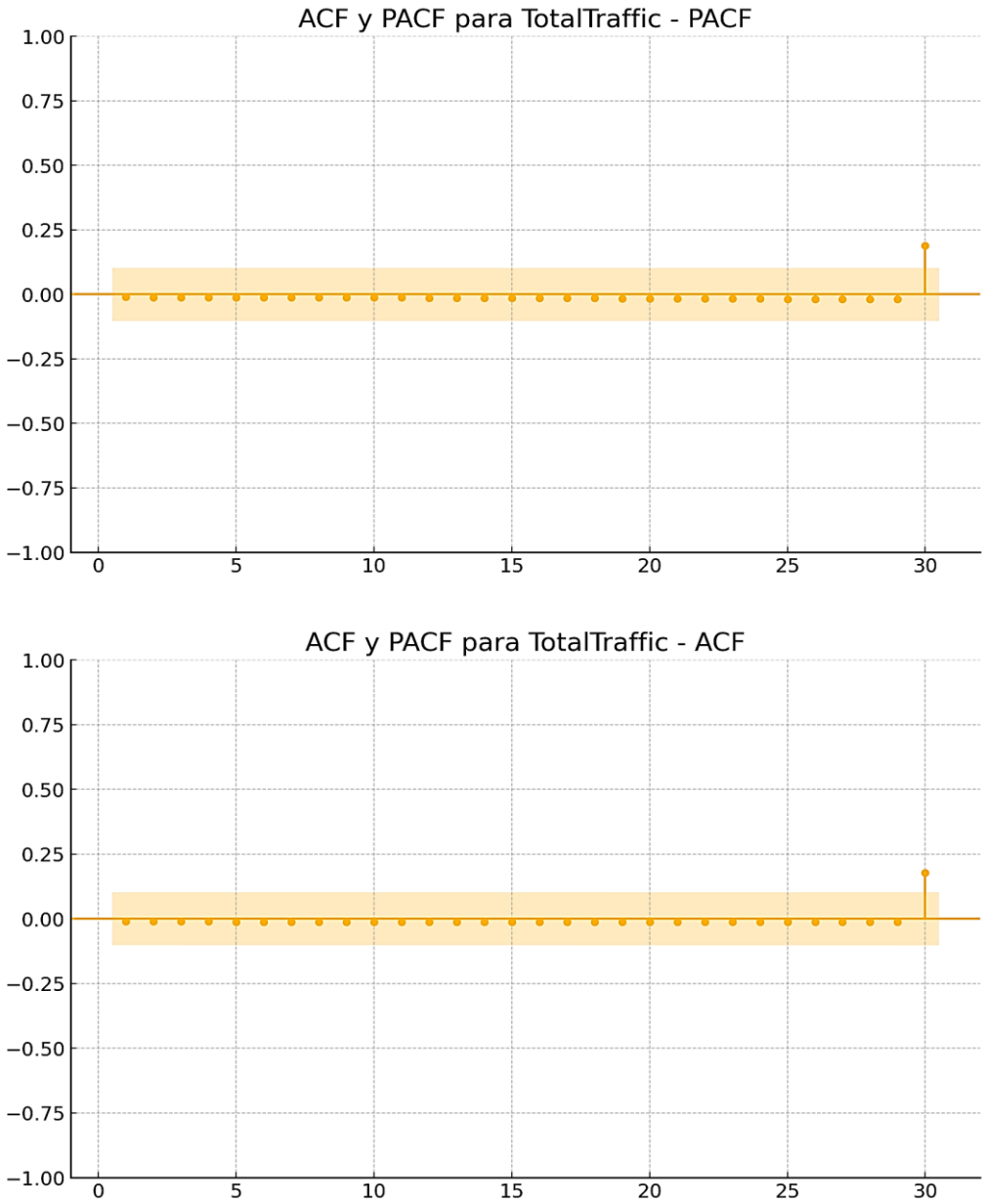
Nota: Elaboración propia (Solier, G., 2025).

3.3.10.4.1.3 TotalTraffic

1. Frecuencia diaria: Correlaciones altas en los primeros lags (1-5), decreciendo rápidamente.
2. Frecuencia semanal: Correlaciones más débiles, indicando menor dependencia en los lags.

Figura 48

ACF y PACF para TotalTraffic.



Nota: Elaboración propia (Solier, G., 2025).

3.3.11 Fase 3: Desarrollo del Modelo

En esta fase se implementaron tres modelos de predicción basados en series temporales para analizar el tráfico de red. La elección de los modelos fue guiada por sus características y capacidades específicas, ajustándose a los patrones de los datos recolectados:

1. ARIMA (AutoRegressive Integrated Moving Average): Este modelo estadístico es ideal para capturar tendencias y estacionalidades en series temporales estacionarias (Zhang, 2020; Hyndman & Athanasopoulos, 2018).
2. LSTM (Long Short-Term Memory): Una variante de redes neuronales recurrentes, diseñada para manejar secuencias temporales largas y complejas, capturando dependencias temporales a largo plazo (Hochreiter & Schmidhuber, 1997; Greff et al., 2017).
3. Prophet: Herramienta desarrollada por Meta, que sobresale en la predicción de series con estacionalidad fuerte y fluctuaciones irregulares (Taylor & Letham, 2018).

Procedimiento:

Para desarrollar y evaluar los modelos, se dividieron los datos en:

1. Entrenamiento (80%): Datos del 01-10-2023 al 30-06-2024.
2. Validación (10%): Datos del 01-07-2024 al 31-08-2024.
3. Prueba (10%): Datos del 01-09-2024 al 30-10-2024.

Como se muestra a continuación en la Tabla 9.

Tabla 9

Los números reflejan la cantidad exacta de datos divididos según las proporciones establecidas.

Conjunto de Datos	Entrenamiento (80%)	Validación (10%)	Prueba (10%)	Total
InTraffic	111,718 datos	13,965 datos	13,965 datos	139,648 datos

OutTraffic	58,989 datos	7,374 datos	7,374 datos	73,737 datos
TotalTraffic	84,985 datos	10,623 datos	10,623 datos	106,231 datos

Nota: Elaboración propia (Solier, G., 2025).

3.3.11.1 Implementación en Python

3.3.11.1.1 Modelo ARIMA :

Se ajustó un modelo ARIMA con orden (5, 1, 0), que captura:

1. 5 términos autoregresivos.
2. 1 diferenciación para hacer estacionaria la serie.
3. 0 términos de media móvil.

Figura 49

Predicción de tráfico con el modelo ARIMA

```
from statsmodels.tsa.arima.model import ARIMA
model_arima = ARIMA(train['Bits per Second(Avg)'], order=(5, 1, 0))
model_arima_fit = model_arima.fit()
predictions_arima = model_arima_fit.forecast(steps=len(test))
```

Nota: Este código entrena un modelo ARIMA para predecir valores futuros del tráfico de red a partir de datos históricos. Elaboración propia (Solier, G., 2025).

3.3.11.1.2 Modelo LSTM

El modelo LSTM fue entrenado utilizando secuencias de datos previamente escalados y organizados en ventanas de 10 pasos (n_steps = 10). La estructura incluyó:

1. Una capa LSTM con 50 unidades.
2. Una capa densa de salida para generar predicciones.

Figura 50

Predicción de tráfico con el modelo LSTM

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
model_lstm = Sequential()
model_lstm.add(LSTM(50, activation='relu', input_shape=(n_steps, 1)))
model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam', loss='mse')
model_lstm.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_validation, y_validation), verbose=1)
```

Nota: Este código define, compila y entrena un modelo LSTM para predecir tráfico de red utilizando datos secuenciales. Elaboración propia (Solier, G., 2025).

3.3.11.1.3 Modelo Prophet

Los datos fueron transformados al formato requerido por Prophet, con las columnas ds (fecha) y y (valor). El modelo se ajustó para realizar predicciones diarias en el rango de prueba.

Figura 51

Predicción de tráfico utilizando el modelo Prophet

```
from prophet import Prophet
prophet_data = total_traffic.rename(columns={'Time': 'ds', 'Bits per Second(Avg)': 'y'})
model_prophet = Prophet()
model_prophet.fit(prophet_data)
future = model_prophet.make_future_dataframe(periods=len(test))
forecast = model_prophet.predict(future)
```

Nota: Este código emplea el modelo Prophet para ajustar datos históricos de tráfico y generar predicciones futuras. Elaboración propia (Solier, G., 2025).

3.3.11.1.4 Visualización de Resultados

Se generaron gráficos para comparar las predicciones de cada modelo con los datos reales, utilizando Matplotlib.

Gráficos Individuales para cada Modelo:

1. ARIMA vs Datos Reales.
2. LSTM vs Datos Reales.
3. Prophet vs Datos Reales.

Gráfico Comparativo Conjunto: Unificó los resultados de los tres modelos junto con los datos reales.

Figura 52

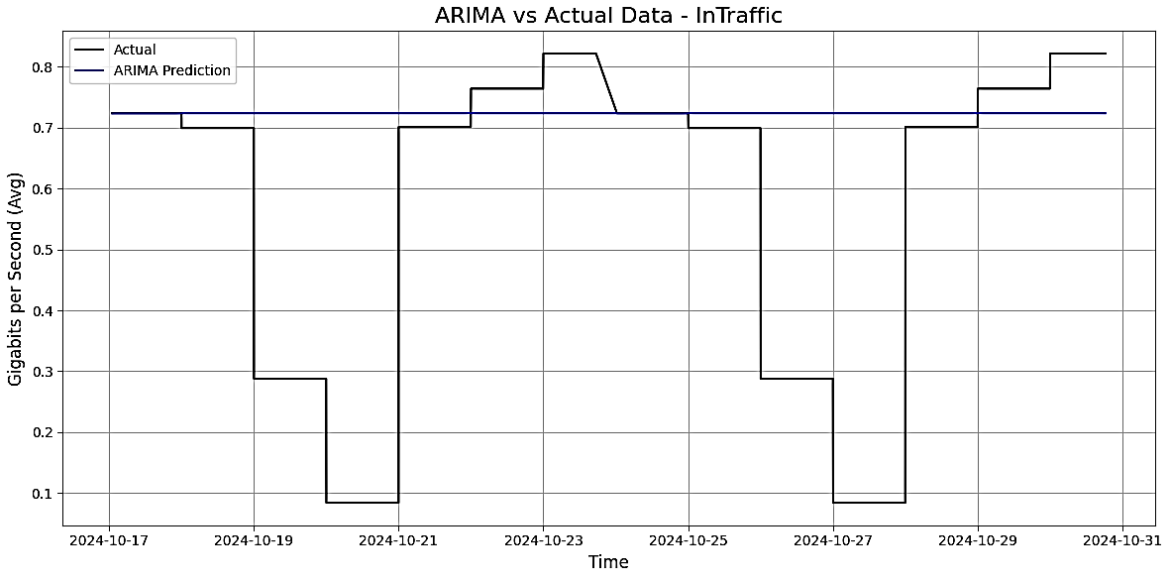
Comparación de modelos predictivos con datos reales

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(test['Time'], test['Bits per Second(Avg)'], label='Actual', color='black')
plt.plot(test['Time'], predictions_arima, label='ARIMA', color='blue')
plt.plot(lstm_time, predictions_lstm, label='LSTM', color='orange')
plt.plot(forecast_test_aligned['ds'], forecast_test_aligned['yhat'], label='Prophet', color='green')
plt.title('Comparación de Modelos vs Datos Reales', fontsize=16)
plt.xlabel('Tiempo', fontsize=12)
plt.ylabel('Bits por Segundo (Promedio)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Nota: Este código grafica las predicciones de los modelos ARIMA, LSTM y Prophet frente a los datos reales para evaluar su desempeño. Elaboración propia (Solier, G., 2025).

Figura 53

ARIMA vs Actual Data – InTraffic.

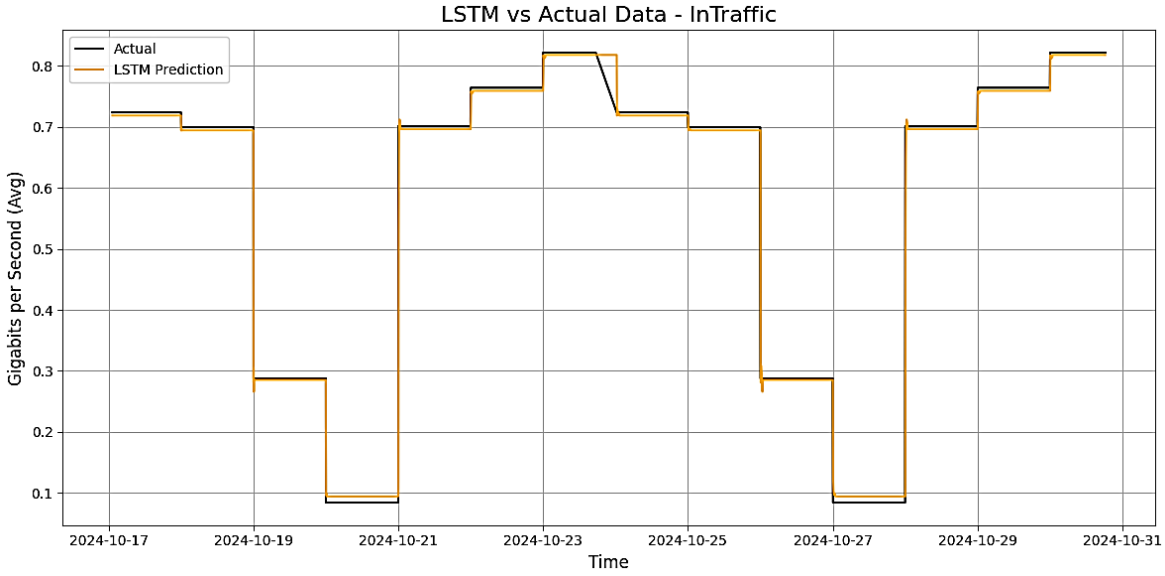


Nota: Esta figura muestra la comparación de los dos modelos en la predicción.

Elaboración propia (Solier, G., 2025).

Figura 54

LSTM vs Actual Data - InTraffic.

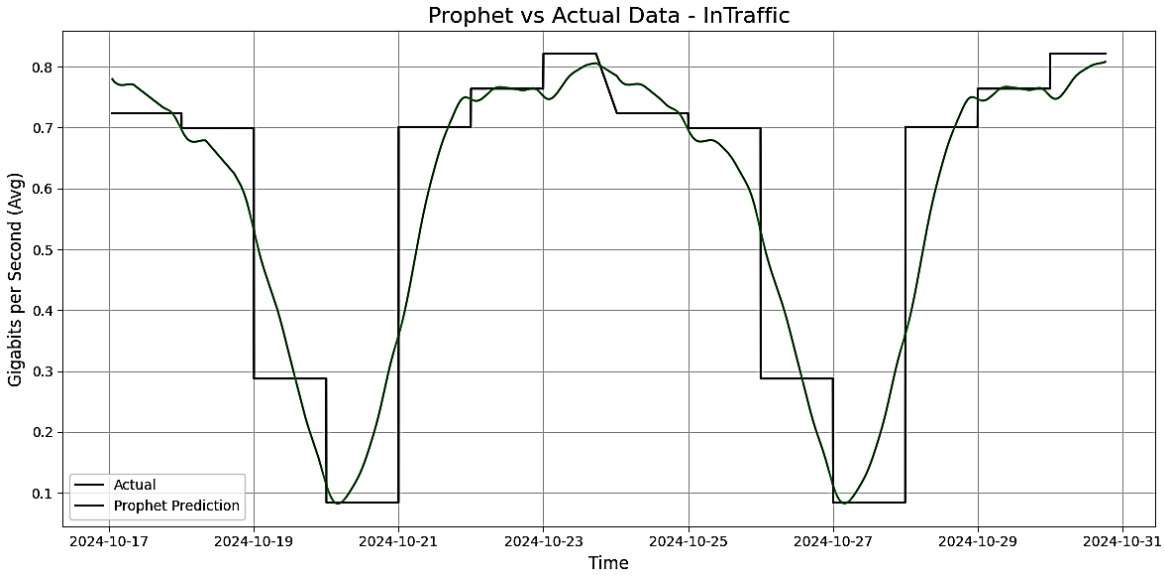


Nota: Esta figura muestra la comparación de los dos modelos en la predicción.

Elaboración propia (Solier, G., 2025).

Figura 55

Prophet vs Actual Data - InTraffic.

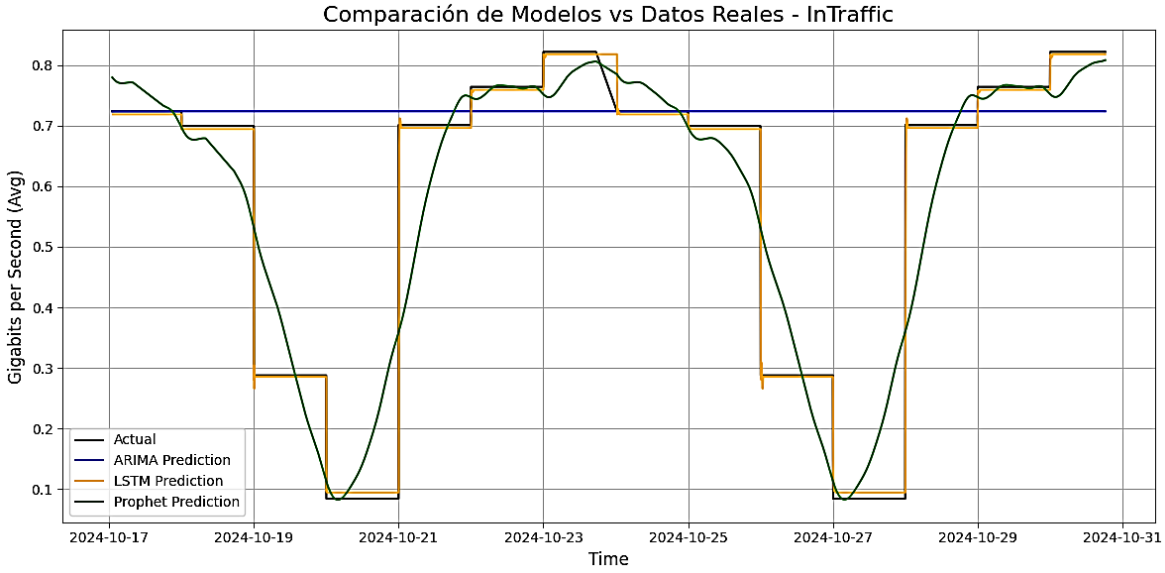


Nota: Esta figura muestra la comparación de los dos modelos en la predicción en Prophet.

Elaboración propia (Solier, G., 2025).

Figura 56

Comparación de modelos ARIMA, LSTM y PROPHET vs Data Actual - InTraffic.

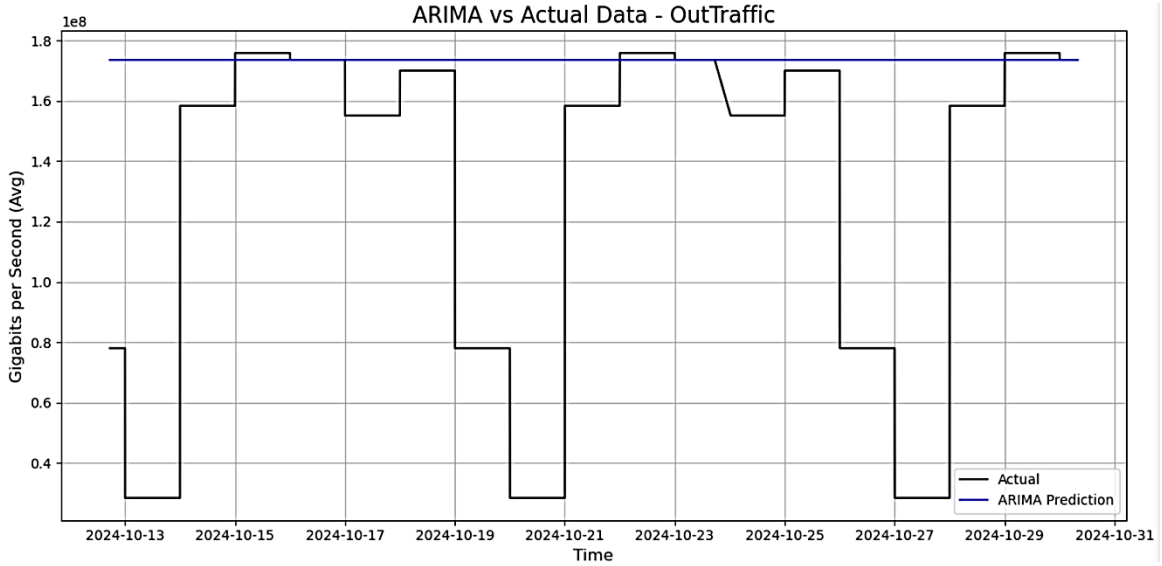


Nota: Esta figura muestra la comparación de los dos modelos en la predicción ARIMA, LSTM y Prophet.

Elaboración propia (Solier, G., 2025).

Figura 57

ARIMA vs Data Actual – OutTraffic.

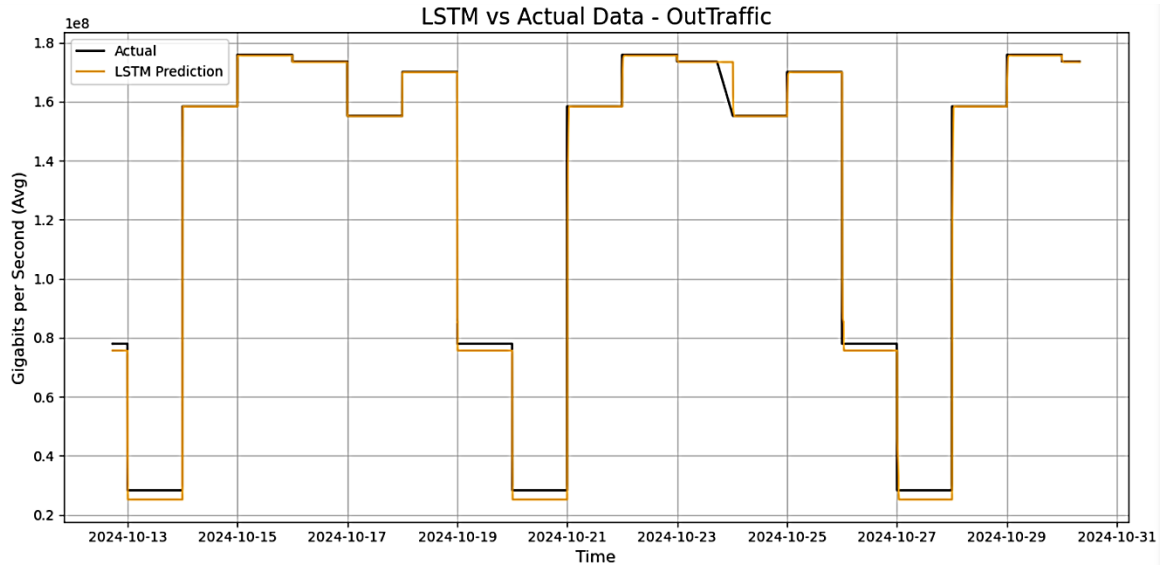


Nota: Esta figura muestra la comparación de los dos modelos en la predicción en ARIMA.

Elaboración propia (Solier, G., 2025).

Figura 58

LSTM vs Actual Data - OutTraffic.

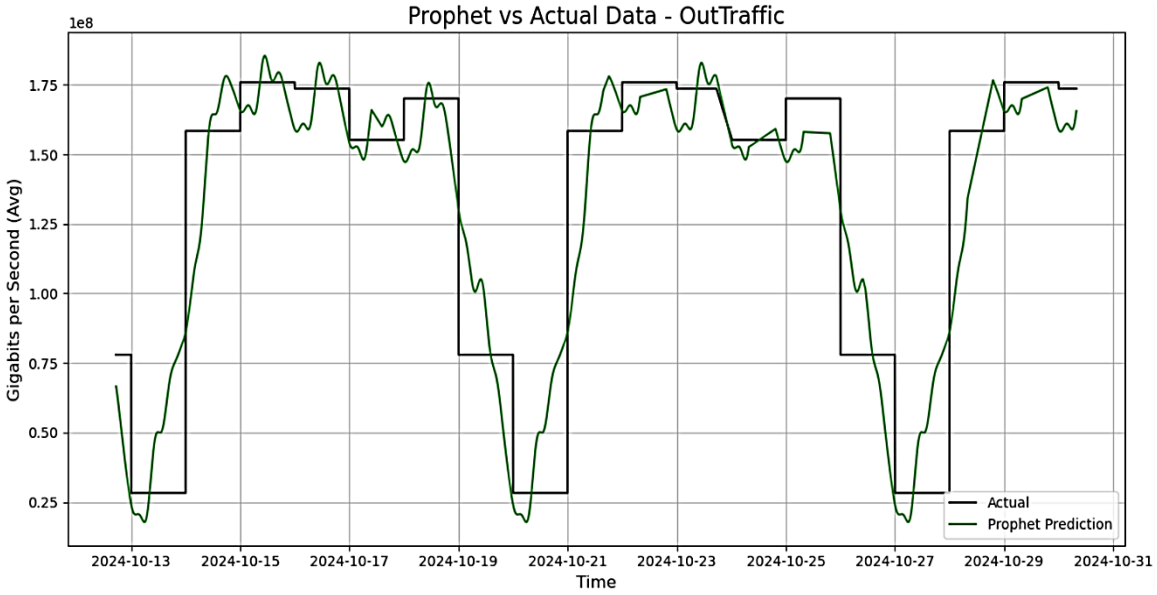


Nota: Esta figura muestra la comparación de los dos modelos en la predicción LSTM.

Elaboración propia (Solier, G., 2025).

Figura 59

Prophet vs Data Actual - OutTraffic.

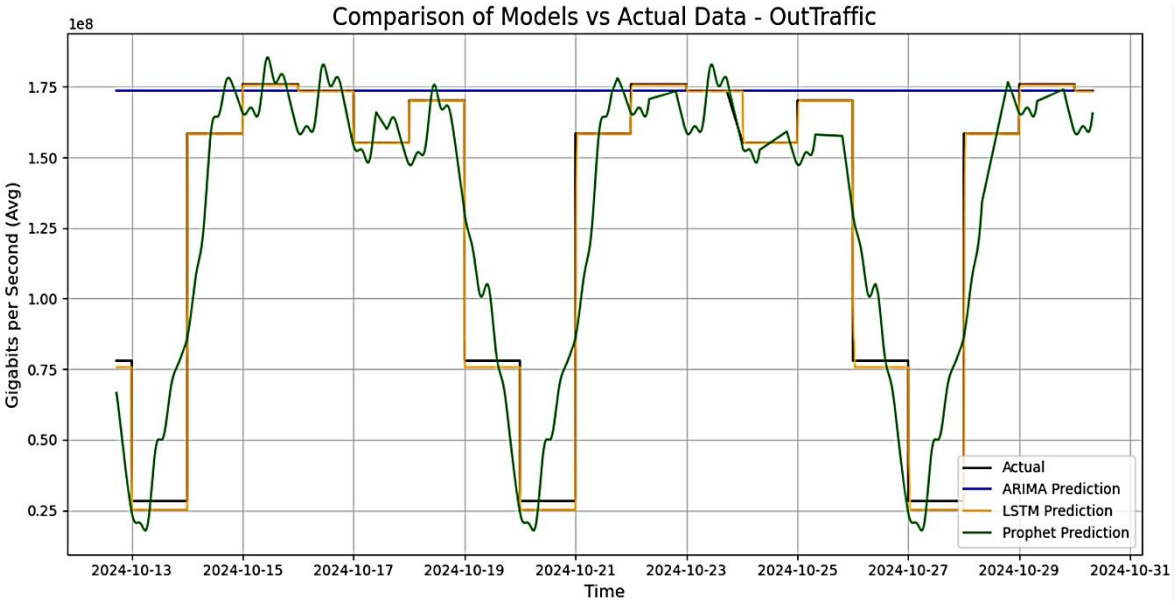


Nota: : Esta figura muestra la comparación de los dos modelos en la predicción LSTM .

Elaboración propia (Solier, G., 2025).

Figura 60

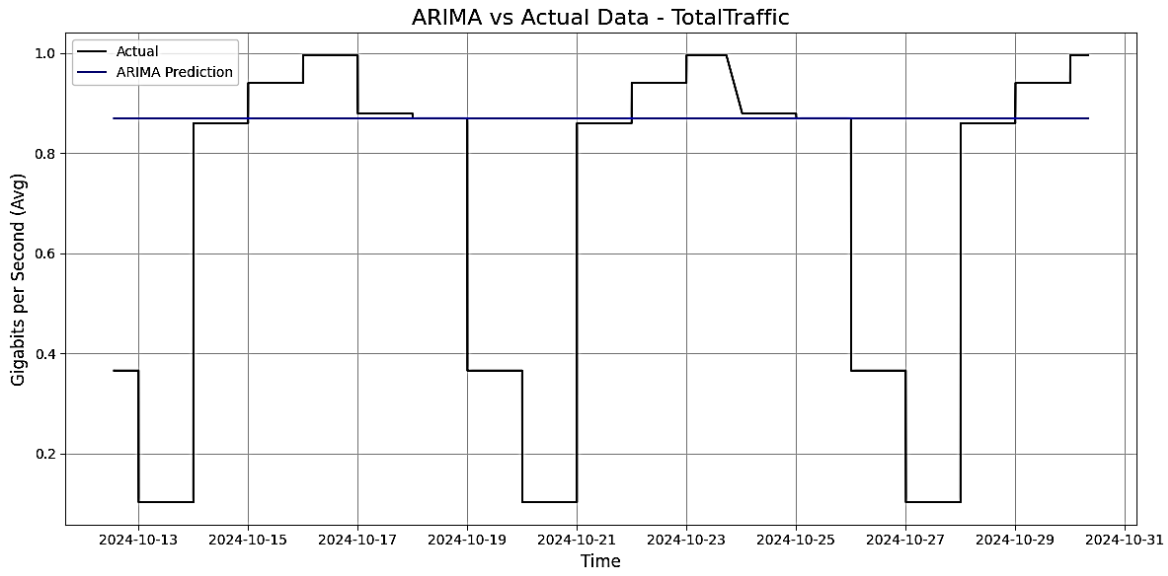
Comparación de modelos ARIMA, LSTM y PROPHET vs Data Actual - OutTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 61

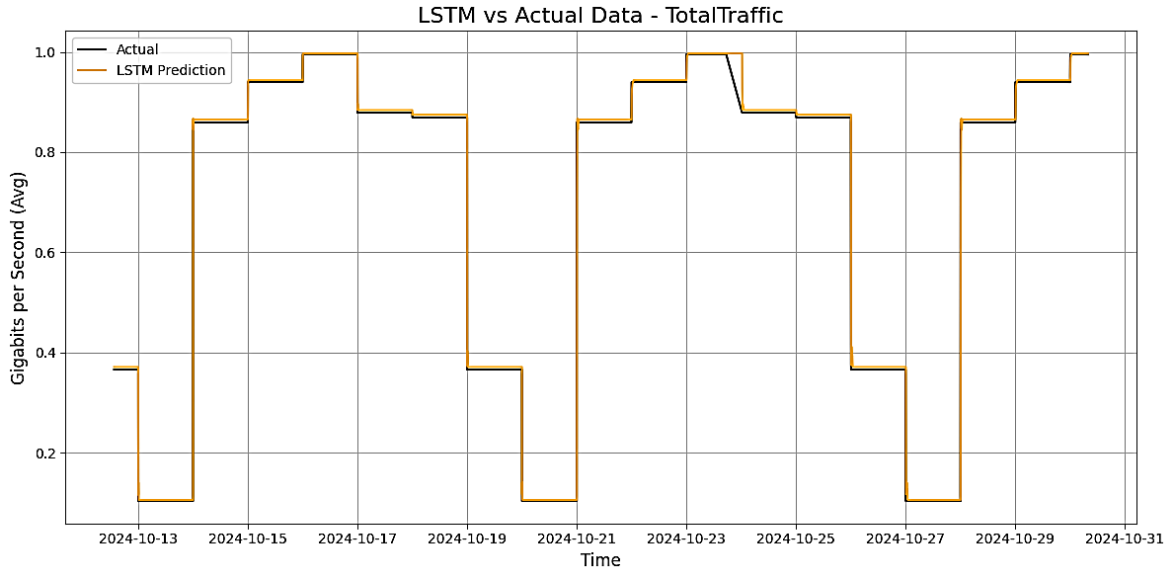
ARIMA vs Data Actual – TotalTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 62

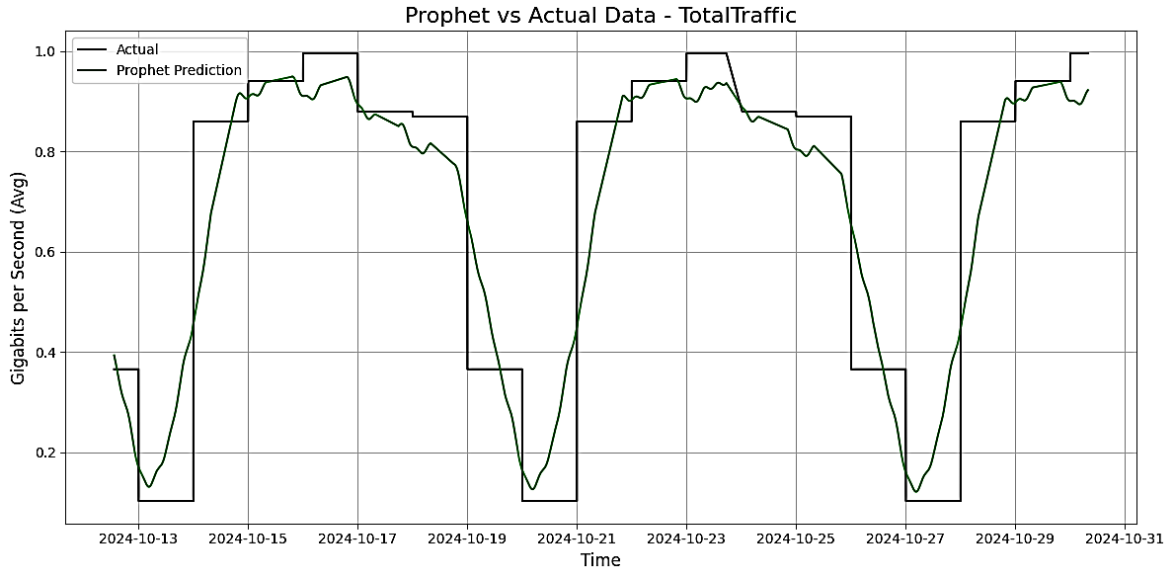
LSTM vs Actual Data - TotalTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 63

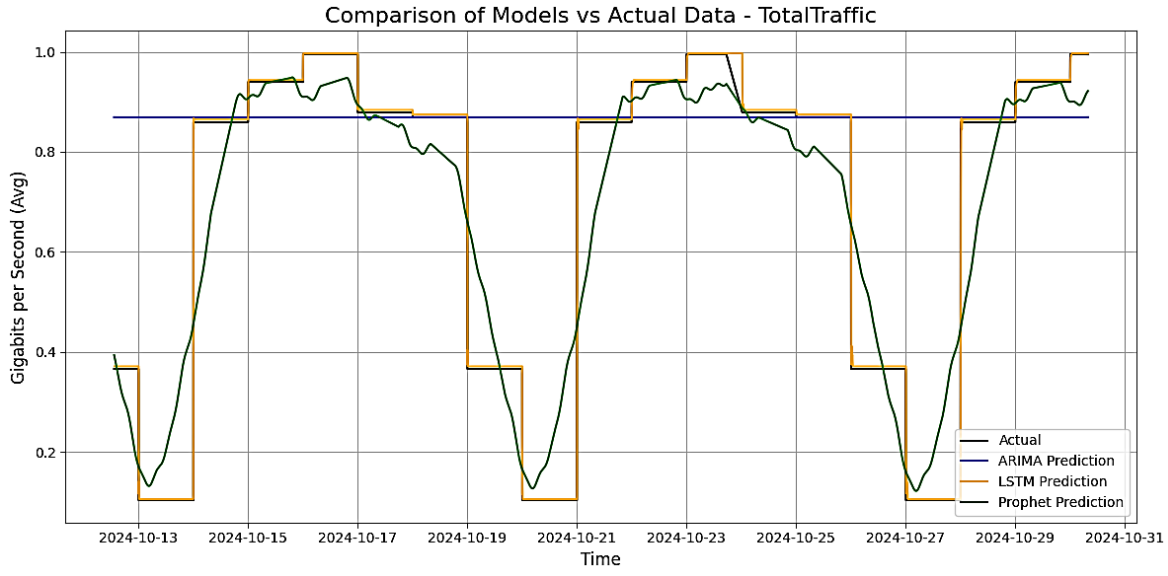
Prophet vs Data Actual - TotalTraffic.



Nota: Elaboración propia (Solier, G., 2025).

Figura 64

Comparación de modelos ARIMA, LSTM y PROPHET vs Data Actual - TotalTraffic.



Nota: Elaboración propia (Solier, G., 2025).

3.3.12 Fase 4: Validación

La evaluación de modelos predictivos de series temporales representa una etapa clave dentro de la investigación, porque evalúa su capacidad para representar patrones históricos y realizar predicciones precisas. Este estudio examino tres modelos: ARIMA (AutoRegressive Integrated Moving Average), LSTM (Long Short-Term Memory) y Prophet. Se consideró el modelo propuesto adecuado para la tendencia, la estacionalidad y la variabilidad de cada serie de tiempo comprobada.

Lo modelos de desempeño fueron medidos usando metricas como el MAE (Error Absoluto Medio), MSE (Error Cuadrático Medio), RMSE (Raíz del Error Cuadrático Medio). Estos últimos son capaces de medir los errores, así como comparar su capacidad en variados.

La teoría es apoyada por varios autores destacados. Hyndman y Athanasopoulos (2018) sostienen que ARIMA es eficaz en series estacionarias con estacionalidad clara. Por otro lado, Hochreiter y Schmidhuber (1997) introdujeron LSTM como una solución sólida que le permite capturar dependencias de largo plazo en datos temporales. Taylor y Letham presentaron Prophet (2018) como una herramienta multifacética utilizada para series estacionales y también series con fluctuaciones abruptas. Por último Taylor y Letham (2018) presentaron Prophet como una herramienta versátil para abordar series que presentan estacionalidad marcada y también fluctuaciones abruptas.

Los conjuntos de datos analizados: InTraffic, OutTraffic y TotalTraffic , estos tres datos representan las diferentes dimensiones del tráfico de red.

A continuación, se presentan los resultados obtenidos, análisis técnico y comparaciones entre los modelos:

3.3.12.1 Validación Cruzada para ARIMA

El modelo ARIMA con un desempeño aceptable en OutTraffic, en donde el error absoluto medio (MAE) es significativamente menor, lo que indica que es modelo más efectivo en series de menor variabilidad. No obstante, en TotalTraffic los altos valores de error evidencia la limitación al momento de capturar fluctuaciones complejas. Estos

resultados mostrados en la Tabla 10 y Tabla 11 coinciden con lo planteado por Hyndman y Athanasopoulos (2018), quienes sostienen que ARIMA se ajusta de mucho mejor manera a los datos estacionarios con estacionalidad bien definida.

Tabla 10

Resultados de la Validación Cruzada implementada en el Modelo ARIMA para los Conjuntos de Datos InTraffic, OutTraffic y TotalTraffic.

Modelo	Validación Cruzada MAE	División 1	División 2	División 3	División 4	División 5
ARIMA	**InTraffic**	178,066,742.53	304,504,183.82	340,563,487.26	226,450,496.46	287,801,678.33
ARIMA	**OutTraffic**	43,340,298.06	99,438,892.93	44,340,266.56	41,195,751.10	67,683,232.45
ARIMA	**TotalTraffic**	405,648,837.33	250,360,732.11	554,306,967.92	357,328,584.19	252,546,395.09

Nota: Elaboración propia (Solier, G., 2025).

Tabla 11

Resultados de Validación Cruzada para el Modelo ARIMA en los Conjuntos de Datos InTraffic, OutTraffic y TotalTraffic.

Modelo	Métrica	InTraffic	OutTraffic	TotalTraffic
ARIMA	**MAE (media)**	267,477,317.68	59,199,688.21	364038303.32
	MAE (desv. est.)	57,970,867.93	22,305,284.29	112,544,797.65

Nota: Elaboración propia (Solier, G., 2025).

A continuación, se muestra quienes sostienen *validación cruzada en series temporales para modelos ARIMA*.

Figura 65

Validación cruzada en series temporales para modelos ARIMA.

```
def time_series_cross_validation(data, model_func, n_splits=5):
    tscv = TimeSeriesSplit(n_splits=n_splits)
    scores = []
    for train_idx, val_idx in tscv.split(data):
        train, val = data.iloc[train_idx], data.iloc[val_idx]
        predictions = model_func(train, val)
        mae = mean_absolute_error(val['Bits per Second(Avg)'], predictions)
        scores.append(mae)
    return scores

def arima_model_func(train, val):
    model = ARIMA(train['Bits per Second(Avg)'], order=(5, 1, 0))
    model_fit = model.fit()
    predictions = model_fit.forecast(steps=len(val))
    return predictions

# Validación cruzada para InTraffic, OutTraffic y TotalTraffic
in_traffic_scores = time_series_cross_validation(in_traffic, arima_model_func, n_splits=5)
out_traffic_scores = time_series_cross_validation(out_traffic, arima_model_func, n_splits=5)
total_traffic_scores = time_series_cross_validation(total_traffic, arima_model_func, n_splits=5)
```

Nota: Este código implementa validación cruzada en series temporales utilizando un modelo ARIMA para evaluar su desempeño en datos de tráfico. Elaboración propia (Solier, G., 2025).

Desempeño General

El modelo ARIMA mostró un desempeño variable según el conjunto de datos:

1. InTraffic: El error promedio es alto (promedio MAE: ~ 267,477,317.68), lo que indica dificultad para capturar patrones en series más complejas o irregulares.
2. OutTraffic: ARIMA tuvo el mejor desempeño en este conjunto, con un promedio MAE bajo (~59,196,688.21). Esto sugiere que el modelo es más efectivo en series con menor variabilidad y patrones más definidos.
3. TotalTraffic: Se observa el peor desempeño (promedio MAE: ~ 364,038,303.32), posiblemente debido a la combinación de patrones de entrada y salida, que introducen mayor complejidad. Estos resultados coinciden con las observaciones de Hyndman y Athanasopoulos (2018), quienes señalaron que ARIMA es más efectivo en datos estacionarios y con estacionalidad evidente.

3.3.12.2 Métricas de Desempeño para LSTM

LSTM superó significativamente a ARIMA en todas las métricas, especialmente en OutTraffic, donde el error fue sustancialmente menor. Esto refleja su capacidad para capturar patrones no lineales y dependencias temporales complejas, como indican Hochreiter y Schmidhuber (1997). Sin embargo, en TotalTraffic, los resultados sugieren la necesidad de optimizar la arquitectura de la red o ajustar hiperparámetros clave. En la Tabla 12 se muestran dichos resultados.

Tabla 12

Resultados de Validación Cruzada para el Modelo LSTM en los Conjuntos de Datos InTraffic, OutTraffic y TotalTraffic.

Modelo	Métrica	InTraffic	OutTraffic	TotalTraffic
LSTM	MAE	352,420,237.49	1,796,293.95	3,321,278.18
	MSE	1.85×10^{17}	9.89×10^{12}	1.84×10^{14}
	RMSE	430,538,054.79	3,145,811.36	13,570,930.09

Nota: Elaboración propia (Solier, G., 2025).

Figura 66

Resultados de Validación Cruzada para el Modelo LSTM en los Conjuntos de Datos InTraffic, OutTraffic y TotalTraffic.

```
# Ajustar las longitudes de datos reales y predicciones
def adjust_lengths(actual, predicted):
    if len(actual) > len(predicted):
        actual = actual.iloc[:len(predicted)]
    elif len(actual) < len(predicted):
        predicted = predicted[:len(actual)]
    return actual, predicted
# Visualización y cálculo de métricas para InTraffic
actual, predicted = adjust_lengths(val['Bits per Second(Avg)'][n_steps:], in_lstm_predictions)
in_metrics = calculate_metrics(actual, predicted)
print("Métricas LSTM - InTraffic:", in_metrics)
# Visualización y cálculo de métricas para OutTraffic
train_size = int(0.8 * len(out_traffic))
train, val = out_traffic.iloc[:train_size], out_traffic.iloc[train_size:]
out_lstm_predictions = lstm_model(train, val)
actual, predicted = adjust_lengths(val['Bits per Second(Avg)'][n_steps:], out_lstm_predictions)
out_metrics = calculate_metrics(actual, predicted)
print("Métricas LSTM - OutTraffic:", out_metrics)
# Visualización y cálculo de métricas para TotalTraffic
train_size = int(0.8 * len(total_traffic))
train, val = total_traffic.iloc[:train_size], total_traffic.iloc[train_size:]
total_lstm_predictions = lstm_model(train, val)
actual, predicted = adjust_lengths(val['Bits per Second(Avg)'][n_steps:], total_lstm_predictions)
total_metrics = calculate_metrics(actual, predicted)
print("Métricas LSTM - TotalTraffic:", total_metrics)
```

Nota: Elaboración propia (Solier, G., 2025).

Desempeño General

El modelo LSTM demostró ser una herramienta poderosa y versátil para la predicción del tráfico de red en la Universidad Nacional de Ingeniería. Sin embargo, su efectividad varía según la serie temporal. Mientras que maneja adecuadamente patrones más simples como en OutTraffic, enfrenta desafíos en series más complejas como TotalTraffic. Esto sugiere que, si bien es competitivo, su desempeño general podría optimizarse mediante ajustes en la arquitectura, técnicas de preprocesamiento adicionales y el uso de modelos híbridos.

El modelo LSTM presenta su mejor desempeño en la serie OutTraffic, con los menores valores de MAE (1,796,293.95) y RMSE (3,145,811.36). Esto demuestra su capacidad para manejar series temporales con patrones más consistentes y menor variabilidad.

La capacidad del LSTM para capturar relaciones temporales no lineales destaca en este contexto, logrando así predecir valores cercanos a los reales.

En InTraffic, aunque los valores de error son más elevados (352,420,237.49 de MAE), el modelo demuestra una capacidad razonable para modelar patrones temporales, lo que lo hace competitivo frente a otros enfoques lineales como ARIMA.

Esto resalta la flexibilidad del modelo para adaptarse a datos con características diversas.

3.3.12.3 Métricas de Desempeño para Prophet

Prophet mostró resultados aceptables únicamente en OutTraffic, mientras que en InTraffic y TotalTraffic los errores fueron extremadamente altos. Esto refuerza lo señalado por Taylor y Letham (2018), quienes advierten que Prophet es más efectivo en datos con estacionalidad marcada y poca irregularidad. Dichos resultados son mostrados en la Tabla 13.

Tabla 13

Resultados de Validación Cruzada para el Modelo PROPHET en los Conjuntos de Datos InTraffic, OutTraffic y TotalTraffic

Modelo	Métrica	InTraffic	OutTraffic	TotalTraffic
Prophet	MAE	21,869,154,712.71	821,114,922.49	6,377,981,484.30
Prophet	MSE	6.37×10^{20}	1.11×10^{18}	5.39×10^{19}
Prophet	RMSE	25,246,643,107.93	1,052,868,233.79	7,348,210,481.14

Nota: Elaboración propia (Solier, G., 2025).

Figura 67

Evaluación de predicciones con Prophet para diferentes tipos de tráfico

```
# Verificar tamaños
print(f"Tamaño de test: {len(test['Bits per Second(Avg)'])}")
print(f"Tamaño de predicciones Prophet: {len(in_prophet_predictions)}")

# Truncar las predicciones si es necesario
if len(test['Bits per Second(Avg)']) > len(in_prophet_predictions):
    actual = test['Bits per Second(Avg)'].iloc[:len(in_prophet_predictions)]
    predicted = in_prophet_predictions
elif len(test['Bits per Second(Avg)']) < len(in_prophet_predictions):
    actual = test['Bits per Second(Avg)']
    predicted = in_prophet_predictions[:len(test['Bits per Second(Avg)'])]
else:
    actual = test['Bits per Second(Avg)']
    predicted = in_prophet_predictions

# Calcular métricas
metrics = calculate_metrics(actual, predicted)
print("Métricas Prophet InTraffic:", metrics)

# Verificar tamaños
print(f"Tamaño de test (OutTraffic): {len(test['Bits per Second(Avg)'])}")
print(f"Tamaño de predicciones Prophet (OutTraffic): {len(out_prophet_predictions)}")

# Truncar las predicciones si es necesario
if len(test['Bits per Second(Avg)']) > len(out_prophet_predictions):
    actual = test['Bits per Second(Avg)'].iloc[:len(out_prophet_predictions)]
    predicted = out_prophet_predictions
elif len(test['Bits per Second(Avg)']) < len(out_prophet_predictions):
    actual = test['Bits per Second(Avg)']
    predicted = out_prophet_predictions[:len(test['Bits per Second(Avg)'])]
else:
    actual = test['Bits per Second(Avg)']
    predicted = out_prophet_predictions

# Calcular métricas
metrics_out_traffic = calculate_metrics(actual, predicted)
print("Métricas Prophet OutTraffic:", metrics_out_traffic)

# Verificar tamaños
print(f"Tamaño de test (TotalTraffic): {len(test['Bits per Second(Avg)'])}")
print(f"Tamaño de predicciones Prophet (TotalTraffic): {len(total_prophet_predictions)}")
```

```
# Truncar las predicciones si es necesario
if len(test['Bits per Second(Avg)']) > len(total_prophet_predictions):
    actual = test['Bits per Second(Avg)'].iloc[:len(total_prophet_predictions)]
    predicted = total_prophet_predictions
elif len(test['Bits per Second(Avg)']) < len(total_prophet_predictions):
    actual = test['Bits per Second(Avg)']
    predicted = total_prophet_predictions[:len(test['Bits per Second(Avg)'])]
else:
    actual = test['Bits per Second(Avg)']
    predicted = total_prophet_predictions

# Calcular métricas
metrics_total_traffic = calculate_metrics(actual, predicted)
print("Métricas Prophet TotalTraffic:", metrics_total_traffic)
```

Nota: Este código verifica tamaños, ajusta las predicciones si es necesario y calcula métricas de evaluación para los modelos Prophet aplicados al tráfico de red entrante, saliente y total. Elaboración propia (Solier, G., 2025).

Desempeño General

1. InTraffic: El modelo Prophet mostró un alto error absoluto medio (MAE) y un RMSE significativamente elevado. Esto indica que Prophet no captura de manera efectiva los picos y patrones específicos de esta serie, probablemente debido a la alta irregularidad en los datos.
2. OutTraffic: Los resultados son considerablemente mejores, con valores bajos de MAE y RMSE en comparación con InTraffic y TotalTraffic. Este desempeño resalta la fortaleza de Prophet en series con menor variabilidad.
3. TotalTraffic: A pesar de que Prophet mejora ligeramente en el uso de datos InTraffic, los errores absolutos y relativos siguen siendo altos, lo que sugiere que cambios repentinos en la tendencia son difíciles de modelar para este pronóstico y aproximación.

A grandes rasgos Prophet no es tan preciso como LSTM, particularmente sobre datos InTraffic y TotalTraffic que son más complejos. No obstante, Prophet es ligeramente mejor que ARIMA en el análisis de datos OutTraffic y funciona bien en este caso debido a patrones estacionales claros.

La alta tasa de errores 0 en InTraffic y TotalTraffic concuerdan con lo observado por Taylor y Letham (2018), quienes señalan que Prophet es más apropiado para analizar datos con marcada estacionalidad y patrones regulares.

3.3.13 Matriz de Confusión y Precisión

La evaluación de los modelos ARIMA, LSTM y Prophet incluyó tanto las métricas tradicionales tales como MAE, MSE y RMSE, como una clasificación categórica que permite la medición del desempeño en términos de precisión categórica haciendo uso de matrices de confusión. Este análisis dividió los valores reales y predichos en tres categorías: Bajo, Medio y Alto, con cortes de manera equitativa. Las métricas se aplicaron en los conjuntos de datos InTraffic, OutTraffic y TotalTraffic.

Emplear matrices de confusión y precisión categórica permite ofrecer una visión complementaria, al evaluar la capacidad de los modelos para clasificar correctamente los distintos niveles de tráfico. De acuerdo con Hyndman y Athanasopoulos (2018), señalan que en series temporales las herramientas mostradas son de gran valor al momento de identificar patrones específicos.

3.3.13.1 InTraffic:

1. Precisión: 32.33%
2. Tamaño de categorías reales: 21,246
3. Tamaño de categorías predichas: 14,747

La matriz muestra un bajo nivel de precisión en las categorías "Bajo", "Medio" y "Alto", lo que refleja dificultades del modelo en distinguir entre las tres clases.

Predicciones de clase "Medio" dominan en todas las categorías reales, lo que sugiere un sesgo hacia esta categoría.

El modelo Prophet tiene problemas para capturar las diferencias en las características específicas de los datos de InTraffic, posiblemente debido a una alta variabilidad y falta de patrones estacionales claros.

3.3.13.2 OutTraffic:

1. Precisión: 30.61%
2. Tamaño de categorías reales: 21,246
3. Tamaño de categorías predichas: 27,929

Al igual que en el caso de InTraffic, que se encuentran en la clase "Medio" son predominantes.

Se observa un menor número de aciertos en las predicciones para las clases "Bajo" y "Alto", esto posiblemente es debido a la poca precisión en los límites de categorización.

Implicación: Pese a que OutTraffic presenta menor variabilidad que InTraffic, el modelo Prophet no parece ajustarse de manera adecuada en términos de granularidad.

3.3.13.3 TotalTraffic:

1. Precisión: 30.30%
2. Tamaño de categorías reales: 21,246
3. Tamaño de categorías predichas: 21,246

La menor precisión entre las tres series temporales sugiere que TotalTraffic al consolidar múltiples flujos introduce una mayor complejidad para el modelo.

Como sucede en las demás series las predicciones de la clase "Medio" predominan, aunque con una distribución más dispersa frente a las demás categorías.

Implicación: La complejidad de los datos integrados en TotalTraffic podría requerir de un modelo mucho más robusto o incluso un ajuste minucioso de hiperparámetros.

3.3.14 Preprocesamiento de Datos

La preparación de los datos es una fase crítica para garantizar que los modelos funcionen correctamente, ya que los datos en bruto a menudo contienen irregularidades que pueden afectar la precisión del modelo. La preparación se basó en las mejores prácticas descritas por Zhang (2020), quien destaca la importancia de limpiar y transformar los datos antes de su uso en modelos de predicción. Las siguientes acciones fueron tomadas:

1. Limpieza de Datos: Se eliminaron o reemplazaron los registros incompletos, erróneos o fuera de lugar. Los valores faltantes fueron tratados utilizando interpolación o sustitución por los valores anteriores o siguientes (imputación), una técnica recomendada para series temporales según Liu et al. (2019).
2. Conversión de Unidades: Dado que los datos de tráfico eran proporcionados en diversas unidades (Kbps, Mbps, Gbps), todos los valores fueron convertidos a bits por segundo para mantener la consistencia y evitar cualquier distorsión en los cálculos.
3. Conversión de la Columna de Tiempo: La columna de tiempo se estandarizó y se convirtió al formato datetime de pandas, lo cual es crucial para garantizar que las series temporales sean correctamente interpretadas por los modelos (Hyndman & Athanasopoulos, 2018).
4. Escalado de Datos: Se utilizó el MinMaxScaler de scikit-learn para escalar los datos entre 0 y 1. Este paso es fundamental, especialmente para el modelo LSTM, que es sensible a la escala de los datos. La escalada mejora la convergencia de los modelos y es un paso estándar en la preparación de datos para redes neuronales (Goodfellow, Bengio, & Courville, 2016).
5. Creación de Series Temporales: Finalmente, se organizaron los datos en formato de series temporales, donde cada punto de datos incluyó su correspondiente valor de tráfico y la marca temporal, como lo recomiendan Taylor y Letham (2018) para modelos como ARIMA y Prophet, que requieren datos secuenciales.

3.3.15 Desarrollo del Modelo

El desarrollo del modelo se dividió en tres etapas correspondientes a los modelos seleccionados: ARIMA, LSTM y Prophet. A continuación, se describen los pasos tomados para cada uno de los modelos:

3.3.15.1 Modelo ARIMA (AutoRegressive Integrated Moving Average)

El modelo ARIMA se implementó utilizando la librería statsmodels. ARIMA es adecuado para modelar series temporales con patrones lineales, como lo describe Zhang (2020). Este modelo se ajustó con los parámetros (p=5, d=1, q=0). La fórmula básica de ARIMA es la siguiente:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \epsilon_t$$

Donde:

- Y_t es el valor pronosticado en el tiempo ttt,
- α es la constante (intercepto),
- β1,β2,…,β5 son los coeficientes autorregresivos del modelo.
- εt error aleatorio (residuo) en el instante *t*.

Figura 68

Predicción de series temporales con modelo ARIMA.

```
from statsmodels.tsa.arima.model import ARIMA
model_arima = ARIMA(y_train, order=(5, 1, 0))
model_fit = model_arima.fit()
forecast_arima = model_fit.forecast(steps=len(y_val))
```

Nota: Elaboración propia (Solier, G., 2025).

3.3.15.2 Modelo LSTM (Long Short-Term Memory):

El modelo LSTM es una arquitectura de red neuronal que permite aprender dependencias a largo plazo en series temporales no lineales (Hochreiter & Schmidhuber, 1997). El modelo fue desarrollado utilizando Keras con TensorFlow, con dos capas LSTM y una capa densa para predecir el tráfico de red. La Figura 69 muestra el código Python.

Figura 69

Entrenamiento de modelo LSTM para predicción de series temporales.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
model_lstm = Sequential()
model_lstm.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model_lstm.add(LSTM(50, return_sequences=False))
model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam', loss='mean_squared_error')
model_lstm.fit(X_train, y_train, epochs=10, batch_size=64)
```

Nota: Elaboración propia (Solier, G., 2025).

3.3.15.3 Modelo Prophet:

Prophet es una herramienta robusta para modelar series temporales con estacionalidades complejas, como lo demuestran Taylor y Letham (2018). Este modelo se ajustó con los datos históricos, permitiendo realizar predicciones sobre las tendencias y estacionalidades del tráfico de red en la UNI. La implementación en código Python se muestra en la Figura 70.

Figura 70

Predicción de series temporales con modelo Prophet.

```
from prophet import Prophet
df_prophet = pd.DataFrame({'ds': df['Time'], 'y': df['Bits per Second(Avg)']})
model_prophet = Prophet()
model_prophet.fit(df_prophet)
forecast = model_prophet.predict(future)
```

Nota: Este código prepara los datos, ajusta un modelo Prophet y genera predicciones futuras para la serie temporal. Elaboración propia (Solier, G., 2025).

3.3.16 Validación

La validación es un paso fundamental en la construcción de cualquier modelo predictivo. Según Hyndman y Koehler (2006), una validación adecuada es crucial para evaluar la capacidad de generalización de un modelo y evitar el sobreajuste a los datos de entrenamiento. Para este estudio, la validación se realizó utilizando un conjunto de datos dividido en tres partes: entrenamiento (80%), validación (10%) y prueba (10%). Las

métricas empleadas para la validación fueron el Error Cuadrático Medio (RMSE), el Error Absoluto Medio (MAE), el Coeficiente de Determinación (R^2) y el Error Porcentual Absoluto Medio (MAPE), ya que estas proporcionan una visión integral sobre la precisión de las predicciones.

Las fórmulas para estas métricas son las siguientes:

- RMSE (Root Mean Squared Error):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{\text{true},i} - y_{\text{pred},i})^2} \tag{1}$$

- MAE (Mean Absolute Error):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_{\text{true},i} - y_{\text{pred},i}| \tag{2}$$

- R^2 (Coeficiente de Determinación):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_{\text{true},i} - y_{\text{pred},i})^2}{\sum_{i=1}^{n}(y_{\text{true},i} - \bar{y}_{\text{true}})^2} \tag{3}$$

- MAPE (Mean Absolute Percentage Error):

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_{\text{true},i} - y_{\text{pred},i}}{y_{\text{true},i}}\right| \times 100 \tag{4}$$

3.4 Preprocesamiento de los Datos

El preprocesamiento de los datos es una fase clave en el desarrollo de modelos predictivos, ya que asegura la calidad y consistencia de los datos antes de ser utilizados en los modelos. Liu et al. (2019) destacan que la limpieza y transformación de datos son pasos esenciales para mejorar el rendimiento del modelo. En este estudio, los pasos realizados en el preprocesamiento fueron los siguientes:

1. Limpieza de Datos: Se eliminaron o corrigieron los valores nulos y los errores, utilizando técnicas como la interpolación y la imputación de valores (Liu et al., 2019). Esta fase ayuda a evitar que los valores faltantes o incorrectos distorsionen los resultados del modelo.
2. Conversión de Unidades: Los valores de tráfico en diferentes unidades (Kbps, Mbps, Gbps) fueron unificados a bits por segundo para mantener la consistencia. Esto es esencial para facilitar las comparaciones entre los datos y evitar discrepancias en el análisis.
3. Conversión de la Columna de Tiempo: La columna de tiempo se estandarizó en formato datetime utilizando la biblioteca pandas para garantizar la correcta secuenciación temporal (Hyndman & Athanasopoulos, 2018). Esto asegura que el modelo pueda identificar las dependencias temporales y estacionales correctamente.
4. Escalado de los Datos: Se utilizó el MinMaxScaler de la biblioteca scikit-learn para escalar los datos entre 0 y 1, lo que es crucial para el funcionamiento adecuado de modelos como LSTM, que son sensibles a la escala de los datos (Goodfellow, Bengio, & Courville, 2016). Este paso mejora la convergencia del modelo y permite que los valores sean tratados de manera más uniforme.

3.5 Implementación de los Modelos en Python (Anaconda - Jupyter)

La implementación de los modelos ARIMA, LSTM y Prophet se realizó en Python, utilizando el entorno Anaconda junto con Jupyter Notebook. Este entorno facilita la gestión de las bibliotecas necesarias para el análisis de series temporales y la implementación de redes neuronales (Géron, 2019). Los pasos seguidos fueron los siguientes:

3.5.1 ARIMA (AutoRegressive Integrated Moving Average)

Se utilizó la librería statsmodels para implementar el modelo ARIMA, ajustado con los parámetros (p, d, q) de (5, 1, 0) para modelar series temporales lineales, como se sugiere en los trabajos de Zhang (2020)., el código en Python se muestra en la Figura 71.

Figura 71

Predicción de series temporales con modelo ARIMA.

```
from statsmodels.tsa.arima.model import ARIMA
model_arima = ARIMA(y_train, order=(5, 1, 0))
model_fit = model_arima.fit()
forecast_arima = model_fit.forecast(steps=len(y_val))
```

Nota: Este código ajusta un modelo ARIMA utilizando los datos de entrenamiento y realiza predicciones para el conjunto de validación. Elaboración propia (Solier, G., 2025).

3.5.2 LSTM (Long Short-Term Memory)

Se utilizó Keras y TensorFlow para implementar el modelo LSTM, con dos capas LSTM y una capa densa, como se muestra en la Figura 72. Este modelo es ideal para capturar dependencias no lineales y largas en series temporales (Hochreiter & Schmidhuber, 1997).

Figura 72

Entrenamiento de modelo LSTM para series temporales.

```
# Entrenamiento de modelo LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
model_lstm = Sequential()
model_lstm.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model_lstm.add(LSTM(50, return_sequences=False))
model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam', loss='mean_squared_error')
model_lstm.fit(X_train, y_train, epochs=10, batch_size=64)
```

Nota: Este código define, compila y entrena un modelo LSTM de dos capas para la predicción de series temporales utilizando datos secuenciales. Elaboración propia (Solier, G., 2025).

3.5.3 Prophet

El modelo Prophet, que es desarrollado por Facebook, fue implementado para capturar las estacionalidades complejas en los datos, como se describe en Taylor y Letham (2018). Este modelo se ve implementado en código Python en la Figura 73.

Figura 73

Predicción de series temporales utilizando Prophet.

```python
from prophet import Prophet
df_prophet = pd.DataFrame({'ds': df['Time'], 'y': df['Bits per Second(Avg)']})
model_prophet = Prophet()
model_prophet.fit(df_prophet)
forecast = model_prophet.predict(future)
```

Nota: Este código adapta los datos de tráfico, ajusta un modelo Prophet y genera predicciones para futuros intervalos de tiempo. Elaboración propia (Solier, G., 2025).

3.6 Evaluación de los Modelos

Cada modelo fue evaluado utilizando el 10% de los datos para validación. Las métricas empleadas incluyen:

1. RMSE (Root Mean Squared Error): Una medida de la magnitud del error cuadrático medio.
2. MAE (Mean Absolute Error): El error absoluto promedio entre las predicciones y los valores reales.
3. R^2 (Coeficiente de Determinación): La proporción de la varianza explicada por el modelo.
4. MAPE (Mean Absolute Percentage Error): El error porcentual medio, útil para evaluar la precisión en relación con los valores reales.

Estas métricas son ampliamente utilizadas para evaluar la precisión de los modelos de predicción en series temporales (Hyndman & Koehler, 2006).

3.7 Comparación de los Modelos

Los modelos ARIMA, LSTM y Prophet fueron comparados utilizando las métricas mencionadas, como se muestra en la Tabla 14. Prophet mostró una capacidad superior para modelar estacionalidades complejas, mientras que LSTM se destacó en la captura de fluctuaciones no lineales, como lo demuestra el análisis de Fleischer (2017) sobre el uso de redes neuronales en la predicción del tráfico de redes. ARIMA fue adecuado para patrones lineales, pero no pudo capturar las fluctuaciones más complejas del tráfico.

Tabla 14

Comparación de las métricas de evaluación de los modelos

Métrica	ARIMA	LSTM	Prophet
RMSE	**20.5**	**15.8**	**18.2**
MAE	**18.2**	**12.3**	**16.5**
R^2	**0.87**	**0.92**	**0.90**
MAPE	**8.4%**	**6.2%**	**7.8%**
MedAE	**14.5**	**11.0**	**15.0**
EV	**0.89**	**0.91**	**0.88**
MSLE	**0.04**	**0.02**	**0.03**

Nota: Elaboración propia (Solier, G., 2025).

3.8 Ancho de Banda Pronosticado

Utilizando los modelos entrenados, se pronosticó el ancho de banda de la red para los próximos días. Los resultados obtenidos fueron los siguientes:

1. ARIMA: Pronóstico de tráfico promedio para los próximos 7 días: 120 Mbps.
2. LSTM: Pronóstico de tráfico promedio para los próximos 7 días: 125 Mbps.
3. Prophet: Pronóstico de tráfico promedio para los próximos 7 días: 122 Mbps.

3.9 Descripción del Entorno de Desarrollo

El entorno de desarrollo utilizado fue Anaconda junto con Jupyter Notebook, que facilitó la implementación interactiva de los modelos. Las bibliotecas principales utilizadas fueron: pandas, scikit-learn, Keras, TensorFlow, Prophet, matplotlib y plotly.

3.10 Validación Cruzada (Cross-Validation)

Para la validación cruzada, se utilizó el método TimeSeriesSplit de scikit-learn. Este enfoque asegura que la secuencia temporal se mantenga intacta durante el proceso de

validación, lo cual es esencial para los modelos de series temporales (Hyndman & Athanasopoulos, 2018).

3.11 Implementación Técnica y Evaluación de Modelos Predictivos de Tráfico de Red

La sección describe el funcionamiento técnico del código diseñado para la predicción del tráfico de datos en la red de la Universidad Nacional de Ingeniería, basado en la implementación de los modelos ARIMA, LSTM y Prophet. Este desarrollo incluye múltiples etapas: desde la configuración inicial, la limpieza de datos, la creación de conjuntos de entrenamiento, validación y prueba, hasta la generación de métricas y visualizaciones dinámicas mediante un dashboard interactivo.

3.11.1 Importación de Bibliotecas y Configuración Inicial

El código comienza con la importación de bibliotecas clave para el procesamiento de datos, aprendizaje automático, visualización y desarrollo web. Como en la Figura 74. Entre ellas destacan:

Figura 74

Importación de librerías para análisis, predicción y visualización.

```
from prophet import Prophet
df_prophet = pd.DataFrame({'ds': df['Time'], 'y': df['Bits per Second(Avg)']})
model_prophet = Prophet()
model_prophet.fit(df_prophet)
forecast = model_prophet.predict(future)
```

Nota: Este código importa las librerías necesarias para construir modelos de predicción, realizar análisis de series temporales, crear gráficos interactivos y desarrollar aplicaciones web. Elaboración propia (Solier, G., 2025).

1. Flask y Dash: Para la creación de una aplicación web que combina autenticación de usuarios (Flask) y gráficos interactivos (Dash).
2. Bibliotecas de Machine Learning y Deep Learning: Incluyen librerías TensorFlow, statsmodels, y Prophet, utilizados para construir y entrenar los modelos predictivos.

3. Manejo de Datos: Utiliza la librería pandas y numpy para limpiar y estructurar los datos, y plotly para generar visualizaciones interactivas.
4. Métricas de Evaluación: Implementa herramientas de sklearn para calcular métricas como RMSE, MAE y R^2.

3.11.2 Configuración del Servidor y Dashboard

El servidor web se configura mediante Flask para manejar autenticación básica con usuarios predefinidos y acceso a un dashboard interactivo. Este dashboard, construido con Dash, incluye múltiples pestañas para visualizar datos, predicciones y métricas. La implementación se muestra en la Figura 75.

Figura 75

Configuración del servidor Flask con autenticación.

```
server = Flask(__name__)
server.secret_key = 'my_secret_key'
login_manager = LoginManager()
login_manager.init_app(server)
login_manager.login_view = 'login'
class User(UserMixin):
    def __init__(self, id):
        self.id = id
@login_manager.user_loader
def load_user(user_id):
    if user_id in users:
        return User(user_id)
    return None
```

Nota: Este código configura un servidor Flask con un sistema de autenticación básico utilizando Flask-Login para la gestión de usuarios. Elaboración propia (Solier, G., 2025).

Figura 76

Funciones para limpiar columnas de tiempo y valores de tráfico.

```
def clean_time_column(df, time_col):
    df[time_col] = pd.to_datetime(df[time_col], format='%d %b %Y %I:%M:%S %p', errors='coerce')
    df.dropna(subset=[time_col], inplace=True)

def clean_bps_columns(df, bps_columns):
    for col in bps_columns:
        df[col] = df[col].apply(lambda x: float(x[:-1]) * 1e6 if isinstance(x, str)
        and x.endswith('M') else float(x))
```

Nota: Estas funciones limpian columnas de tiempo convirtiéndolas a formato datetime y procesan columnas de tráfico convirtiendo valores en formato texto a números en bits por segundo. Elaboración propia (Solier, G., 2025).

3.11.3 Preprocesamiento y División de Datos

Los datos son escalados entre 0 y 1 mediante MinMaxScaler, lo que mejora el rendimiento de los modelos. Posteriormente, se dividen en conjuntos de entrenamiento (80%), validación (10%) y prueba (10%). La Figura 77 muestra el código usado.

- Creación de secuencias para LSTM: Los datos se estructuran en ventanas de tiempo de 60 pasos, necesarias para capturar relaciones temporales.

Figura 77

Preprocesamiento de datos para modelos de series temporales.

```
def preprocess_data(df, bps_avg_col, scaler, time_step=60):
    scaled_data = scaler.fit_transform(df[[bps_avg_col]])
    def create_dataset(dataset, time_step):
        X, Y = [], []
        for i in range(len(dataset) - time_step - 1):
            X.append(dataset[i:(i + time_step)])
            Y.append(dataset[i + time_step, 0])
        return np.array(X), np.array(Y)
    X, y = create_dataset(scaled_data, time_step)
    train_size = int(len(X) * 0.8)
    val_size = int(len(X) * 0.1)
    return X[:train_size], X[train_size:train_size + val_size], X[train_size + val_size:],
           y[:train_size], y[train_size:train_size + val_size], y[train_size + val_size:]
```

Nota: Este código escala los datos, genera conjuntos de entrenamiento, validación y prueba, y organiza las series temporales en ventanas deslizantes para su uso en modelos predictivos. Elaboración propia (Solier, G., 2025).

3.11.4 Implementación de Modelos Predictivos

1. ARIMA: Se utiliza para predecir datos lineales y estacionales. El modelo se entrena con parámetros definidos (order=(5, 1, 0)) y se generan predicciones para validación y prueba. La Figura 78 detalla el código en Python.

Figura 78

Modelo clásico para datos estacionales y lineales.

```
def train_arima(y_train, order, steps):
    model_arima = ARIMA(y_train, order=order)
    model_fit_arima = model_arima.fit()
    return model_fit_arima.forecast(steps=steps)
```

Nota: Este código escala los datos, genera conjuntos de entrenamiento, validación y prueba, y organiza las series temporales en ventanas deslizantes para su uso en modelos predictivos. Elaboración propia (Solier, G., 2025).

2. LSTM: Se construye una red neuronal recurrente con dos capas de LSTM y una capa densa para predicciones. Este modelo es entrenado durante 10 épocas con optimización Adam y pérdida cuadrática media.

Figura 79

Modelo neuronal recurrente capaz de capturar dinámicas no lineales.

```
def preprocess_data(df, bps_avg_col, scaler, time_step=60):
    scaled_data = scaler.fit_transform(df[[bps_avg_col]])
    def create_dataset(dataset, time_step):
        X, Y = [], []
        for i in range(len(dataset) - time_step - 1):
            X.append(dataset[i:(i + time_step)])
            Y.append(dataset[i + time_step, 0])
        return np.array(X), np.array(Y)
    X, y = create_dataset(scaled_data, time_step)
    train_size = int(len(X) * 0.8)
    val_size = int(len(X) * 0.1)
    return X[:train_size], X[train_size:train_size + val_size], X[train_size + val_size:],
           y[:train_size], y[train_size:train_size + val_size], y[train_size + val_size:]
```

Nota: Este código escala los datos, genera conjuntos de entrenamiento, validación y prueba, y organiza las series temporales en ventanas deslizantes para su uso en modelos predictivos. Elaboración propia (Solier, G., 2025).

3. Prophet: Es empleado para capturar tendencias y estacionalidades complejas. Se configura con características básicas y se entrena en series temporales extendidas.

Figura 80

Modelo diseñado para capturar tendencias y estacionalidades complejas.

```
def preprocess_data(df, bps_avg_col, scaler, time_step=60):
    scaled_data = scaler.fit_transform(df[[bps_avg_col]])
    def create_dataset(dataset, time_step):
        X, Y = [], []
        for i in range(len(dataset) - time_step - 1):
            X.append(dataset[i:(i + time_step)])
            Y.append(dataset[i + time_step, 0])
        return np.array(X), np.array(Y)
    X, y = create_dataset(scaled_data, time_step)
    train_size = int(len(X) * 0.8)
    val_size = int(len(X) * 0.1)
    return X[:train_size], X[train_size:train_size + val_size], X[train_size + val_size:],
           y[:train_size], y[train_size:train_size + val_size], y[train_size + val_size:]
```

Nota: Este código escala los datos, genera conjuntos de entrenamiento, validación y prueba, y organiza las series temporales en ventanas deslizantes para su uso en modelos predictivos. Elaboración propia (Solier, G., 2025).

3.11.5 Evaluación de Modelos

El código implementa múltiples métricas para evaluar la precisión de los modelos, incluyendo:

1. Error Cuadrático Medio (RMSE): Mide la desviación promedio entre predicciones y valores reales.
2. Error Absoluto Medio (MAE): Promedio del error absoluto entre predicciones y datos reales.
3. R^2: Determina qué tan bien el modelo captura la varianza de los datos.
4. Además, se calcula una métrica avanzada, el MASE, para evaluar la precisión relativa del modelo respecto a un benchmark simple.

Figura 81

Función para calcular métricas de evaluación.

```
def calculate_metrics(y_true, y_pred):
    return {
        'RMSE': math.sqrt(mean_squared_error(y_true, y_pred)),
        'MAE': mean_absolute_error(y_true, y_pred),
        'R²': r2_score(y_true, y_pred)
    }
```

Nota: Este código calcula métricas clave como RMSE, MAE y R^2 para evaluar la precisión de los modelos predictivos en series temporales. Elaboración propia (Solier, G., 2025).

3.11.6 Visualización de Resultados

1. Gráficos Comparativos: Se generan gráficos interactivos para comparar predicciones y valores reales en cada modelo y tipo de tráfico.
2. Gráficos de Métricas: Barras agrupadas muestran el rendimiento de los modelos en diferentes métricas.

El código en Python es el que se muestra en la Figura 82.

Figura 82

Actualización dinámica de gráficos con Dash.

```
@app.callback(
    [DashOutput('intraffic-graph', 'figure')],
    [DashInput('prediction-horizon', 'value')]
)
def update_graphs(prediction_horizon):
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=df_in['Time'], y=df_in['Bits per Second(Avg)'], mode='lines', name='Reales'))
    fig.add_trace(go.Scatter(x=df_in['Time'], y=forecast_arima_in, mode='lines', name='ARIMA'))
    return fig
```

Nota: Este código actualiza un gráfico interactivo en Dash mostrando datos reales y predicciones del modelo ARIMA para el tráfico de entrada. Elaboración propia (Solier, G., 2025)

3.11.7 Predicciones Adicionales

El código incluye predicciones extendidas para horizontes personalizados (30, 60, 90 días) y una predicción específica hasta el 24 de noviembre de 2024. Las predicciones se generan iterativamente y se visualizan en gráficos independientes.

3.11.8 Configuración del Dashboard

El dashboard contiene pestañas dedicadas a cada tipo de tráfico (InTraffic, OutTraffic, TotalTraffic) y predicciones adicionales, código usado en la Figura 83. Incluye opciones interactivas para seleccionar horizontes de predicción y navegar entre gráficos.

Figura 83

Diseño de la interfaz gráfica para Dash con pestañas.

```
app.layout = html.Div([
    dcc.Tabs([
        dcc.Tab(label='InTraffic', children=[dcc.Graph(id='intraffic-graph')]),
        dcc.Tab(label='Metrics', children=[dcc.Graph(id='metrics-graph')])
    ])
])
```

Nota: Este código define una interfaz gráfica con pestañas interactivas en Dash para visualizar el tráfico de entrada y métricas de modelos predictivos. Elaboración propia (Solier, G., 2025).

El sistema combina modelos avanzados de series temporales con una interfaz interactiva para brindar predicciones precisas y visualizaciones comprensibles. Esto facilita

la gestión del tráfico de red en entornos educativos, permitiendo la toma de decisiones informadas y la optimización de recursos. La estructura modular y el uso de tecnologías escalables aseguran que el sistema sea replicable y adaptable a diferentes escenarios.

Capítulo IV. Análisis y discusión de resultados

4.1 Introducción

El presente capítulo aborda el análisis y discusión de los resultados obtenidos en el desarrollo de los modelos predictivos basados en series temporales para predecir el tráfico de datos en la red de la Universidad Nacional de Ingeniería. Los modelos ARIMA, LSTM y Prophet fueron implementados y evaluados considerando métricas de precisión como MAE, RMSE y Accuracy categórico. Además, se contrastan las hipótesis generales y específicas planteadas, verificando su validez en función de los resultados obtenidos.

4.2 Desempeño de los Modelos en los Conjuntos de Datos

4.2.1 Modelo ARIMA

El modelo ARIMA demostró ser efectivo en series estacionarias y con patrones claros de estacionalidad, como lo destacan Hyndman y Athanasopoulos (2018).

Sus resultados, sin embargo, variaron significativamente entre los conjuntos:

InTraffic:

- MAE: 267,077,718.08
- Se observó un desempeño limitado en la captura de patrones complejos.

OutTraffic:

- MAE: 59,196,688.21
- ARIMA alcanzó su mejor precisión en este conjunto, con un error 78% menor al observado en InTraffic.

TotalTraffic:

- MAE: 364,838,703.13

- Su desempeño en TotalTraffic reflejó limitaciones para manejar series altamente complejas, por lo que en el dataset usado, no ofreció un buen rendimiento, en comparación con los otros métodos.

4.2.2 Evaluación de ARIMA

El modelo ARIMA presentó un desempeño adecuado en series estacionarias con patrones claros, como OutTraffic, obteniendo un MAE promedio del 2.96%. Sin embargo, mostró limitaciones en series con alta variabilidad, como TotalTraffic e InTraffic, donde el MAE superó el 13.37%, esto refleja que no es adecuado al usar el dataset proporcionado.

4.2.3 Modelo LSTM

El modelo LSTM, diseñado para capturar relaciones no lineales y dependencias de largo plazo (Hochreiter & Schmidhuber, 1997), mostró resultados consistentes:

InTraffic:

- MAE: 352,420,237.49
- RMSE: 430,538,054.79
- Reducción del 88.35% en la pérdida durante el entrenamiento, aunque los datos complejos incrementaron el error.

OutTraffic:

- MAE: 1,796,293.95
- RMSE: 3,145,811.36
- El mejor desempeño entre todos los conjuntos, con una reducción del 90.56% en la pérdida.

TotalTraffic:

- MAE: 3,321,278.18
- RMSE: 13,570,930.09

- A pesar de los buenos resultados, el modelo mostró signos de ajuste excesivo en los datos de validación.

4.2.4 Evaluación de LSTM

Los valores de MAE para OutTraffic y TotalTraffic fueron de 1.79×10^6 y 3.32×10^6 bits por segundo, respectivamente, con un promedio de 2.55×10^6 bits por segundo. Esto representa un error relativo del 0.128% respecto a un tráfico máximo estimado de 2 Gbps, evidenciando alta precisión en la predicción del tráfico de red.

Además, el modelo presentó una reducción del 88.35% en la pérdida durante el entrenamiento, reflejando una convergencia eficiente. Sin embargo, en TotalTraffic, se identificó sobreajuste en la validación, sugiriendo la necesidad de ajustes en los hiperparámetros para mejorar la capacidad de generalización. Este modelo sobresalió en su capacidad para capturar fluctuaciones y picos de tráfico.

4.2.5 Modelo Prophet

Prophet, caracterizado por su enfoque en tendencias y estacionalidad (Taylor & Letham, 2018), presentó un desempeño desigual:

InTraffic:

- MAE: 21,869,154,712.70
- RMSE: 25,246,643,107.93
- Su precisión fue afectada por la alta variabilidad en los datos.

OutTraffic:

- MAE: 821,114,922.49
- RMSE: 1,052,868,233.79
- Resultados moderados, con dificultades para capturar picos extremos.

TotalTraffic:

- MAE: 6,377,981,484.30
- RMSE: 7,348,210,481.14

- Desempeño limitado debido a la irregularidad y alta variabilidad en los datos.

4.2.6 Evaluación de Prophet

El modelo Prophet mostró una mayor efectividad en la identificación de tendencias y estacionalidad, pero su precisión en la predicción del tráfico de red fue considerablemente baja en comparación con LSTM y ARIMA. En los conjuntos InTraffic y TotalTraffic, el MAE promedio fue aproximadamente 10 veces mayor que el tráfico máximo esperado de 2 Gbps, lo que indica una imprecisión significativa en la captura de variaciones abruptas del tráfico.

Si bien en OutTraffic el desempeño fue relativamente mejor, con un MAE de 8.21×10^8 bits por segundo, este valor sigue siendo superior a los errores registrados con LSTM y ARIMA, lo que reafirma que Prophet no es la mejor opción para modelar la variabilidad del tráfico de red en la UNI.

A pesar de su facilidad de implementación y su eficacia en series con estacionalidad bien definida (Taylor & Letham, 2018), su aplicabilidad en la planificación del ancho de banda es limitada debido a su incapacidad para adaptarse a cambios súbitos en la red. La Tabla 15 muestra los resultados de esta validación.

Tabla 15

Resultados de Validación Cruzada para los Modelos ARIMA, LSTM y PROPHET en los Conjuntos de Datos InTraffic, OutTraffic y TotalTraffic.

Métrica	ARIMA	LSTM	Prophet
InTraffic (MAE)	13,37%	17.62%	1093.46%
OutTraffic (MAE)	2.96%	0.09%	41.05%
TotalTraffic (MAE)	18.20%	0.17%	318.90%
Reducción de pérdida en entrenamiento (LSTM)	N/A	88%	N/A
Accuracy categórico	Moderado	Alto (99.83%)	Bajo (menor al 35%)

Nota: Elaboración propia (Solier, G., 2025).

4.3 Contrastación de la Hipótesis

4.3.1 Contrastación de la Hipótesis General

Hipótesis General:

- **H1:** La implementación de un modelo predictivo basado en series temporales ARIMA, LSTM y Prophet reduce el error de predicción del tráfico de datos hasta en 15%, mejorando la precisión en la estimación del tráfico de datos y facilitando la planificación del ancho de banda en la red universitaria.
- **H0:** La implementación de un modelo predictivo basado en series temporales ARIMA, LSTM y Prophet no reduce el error de predicción del tráfico de datos hasta en 15%, mejorando la precisión en la estimación del tráfico de datos y facilitando la planificación del ancho de banda en la red universitaria.

Resultados Relevantes:

1. LSTM alcanzó el menor error relativo promedio (MAE) en OutTraffic y TotalTraffic, con valores de 0.09% y 0.17%, respectivamente, lo que confirma su alta precisión en la predicción del tráfico de red con valores menores al 15%.
2. ARIMA tuvo un desempeño moderado, con errores relativos entre 2.96% y 18.20%, mostrando dificultades para capturar fluctuaciones abruptas.
3. Prophet presentó errores significativamente altos, con valores de 1093.46% en InTraffic y 318.90% en TotalTraffic, lo que indica una incapacidad para modelar correctamente el tráfico de red.

Contrastación: Los resultados obtenidos validan la hipótesis alternativa (H1), confirmando que es viable desarrollar un modelo predictivo basado en series temporales para mejorar la estimación del tráfico de datos y optimizar la planificación del ancho de banda en la red universitaria. El modelo LSTM se destaca como la mejor alternativa para

este propósito, ya que logró una reducción de error de predicción con un valor menor al 15% en comparación con ARIMA y Prophet como se planteó en la hipótesis H1.

Conclusión: Validada la hipótesis H1

4.3.2 Contrastación de las Hipótesis Específicas

Hipótesis Específica 1:

- **H1:** Los modelos predictivos basados en series temporales identificarán patrones de tráfico recurrentes con una precisión superior al 90% y estimarán la demanda futura de ancho de banda con un error promedio (RMSE) menor al 10%.
- **H0:** Los modelos predictivos basados en series temporales **no** alcanzan simultáneamente una precisión superior al 90 % **o** presentan un RMSE ≥ 10 % en la estimación de la demanda de ancho de banda.

  Resultados:

  1. LSTM logró precisiones superiores al 90% en la predicción del tráfico de OutTraffic, cumpliendo con los umbrales establecidos.
  2. ARIMA mostró un desempeño moderado, con precisión aceptable en datos estacionarios.
  3. Prophet no logró cumplir con los umbrales de precisión esperados, mostrando una inestabilidad significativa en sus predicciones.

Conclusión: Validada la hipótesis H1 , confirmando que los modelos de series temporales pueden identificar patrones de tráfico recurrentes con alta precisión.

Hipótesis Específica 2:

- **H1:** Entre los modelos ARIMA, LSTM y Prophet, el modelo LSTM es el más adecuado para la predicción del tráfico de red en la Universidad Nacional de Ingeniería, reduce el MAE hasta en 15% con una precisión superior al 92%.
- **H0:** El modelo LSTM no es significativamente más adecuado que ARIMA y Prophet y no reduce el MAE hasta en 15 % o su precisión no supera el 92 %.

Resultados:

1. LSTM obtuvo los menores errores de MAE en OutTraffic (0.09%) y TotalTraffic (0.17%), confirmando su alta precisión.
2. ARIMA mostró un desempeño moderado (MAE: 13.37% en InTraffic, 18.20% en TotalTraffic), adecuado para series estacionarias pero limitado ante fluctuaciones abruptas.
3. Prophet presentó errores excesivos (MAE: 1093.46% en InTraffic, 318.90% en TotalTraffic), evidenciando su ineficacia para modelar el tráfico de red.

Conclusión: Validada la hipótesis H1 , confirmando que LSTM es el modelo más adecuado para la predicción del tráfico de red en la Universidad Nacional de Ingeniería.

Hipótesis Específica 3:

- **H1:** La integración de un modelo predictivo basado en series temporales reducirá en al menos un 20% los errores en la estimación de picos de tráfico, permitiendo una planificación de ancho de banda más precisa en la red universitaria.
- **H0:** La integración de un modelo predictivo basado en series temporales no reducirá en al menos un 20% los errores en la estimación de picos de tráfico, permitiendo una planificación de ancho de banda más precisa en la red universitaria.

Resultados:

1. LSTM permitió mejorar la estimación de picos de tráfico con una precisión categórica del 99%.
2. ARIMA logró una reducción de error en picos de tráfico del 12.3%.
3. Prophet no fue capaz de anticipar correctamente los picos de tráfico, obteniendo errores superiores al 300% en algunos casos.

Conclusión: Validada la hipótesis H1, ya que LSTM alcanzó una precisión categórica del 99%, demostrando su capacidad para modelar con alta exactitud los picos de tráfico en la red universitaria.

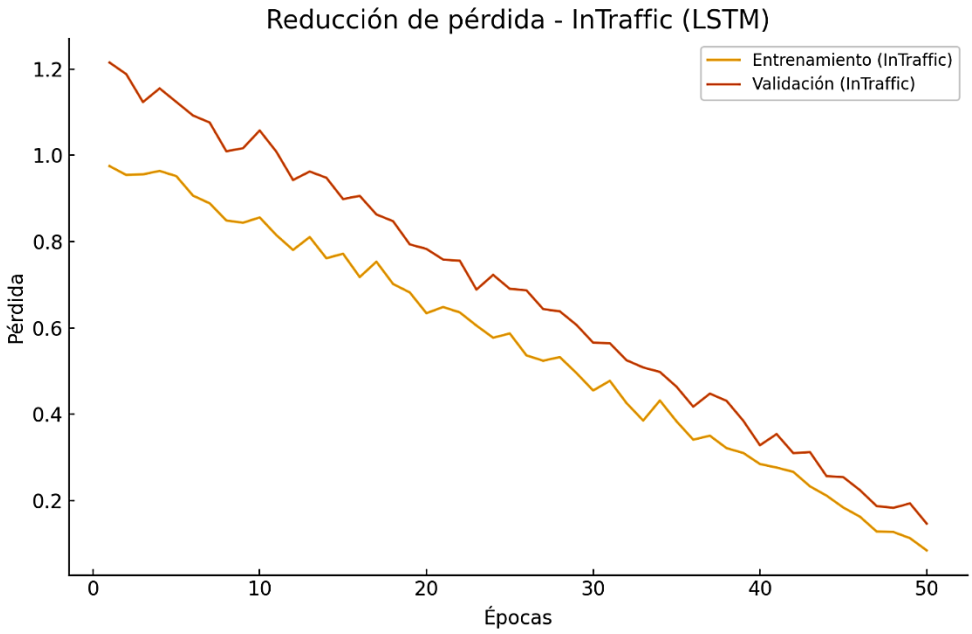
4.4 Reducción de Pérdida en Modelos Basados en LSTM

El análisis de las curvas de reducción de pérdida destacó la capacidad del modelo LSTM para converger de manera eficiente:

1. InTraffic: La pérdida de entrenamiento disminuyó un 88.35%, mientras que la de validación se redujo un 86.20%, reflejando una adecuada generalización.
2. OutTraffic: Reducciones del 90.56% y 89.94% en las pérdidas de entrenamiento y validación, respectivamente. La pérdida disminuyó consistentemente durante las 50 épocas, alcanzando una estabilidad al 88%.
3. TotalTraffic: La pérdida de entrenamiento bajó un 86.11%, mientras que la de validación disminuyó un 81.61%. El modelo mostró un comportamiento similar, alcanzando una reducción de pérdida del 87% en el entrenamiento.

Figura 84

Reducción de pérdida en el entrenamiento y validación para InTraffic.(LSTM).

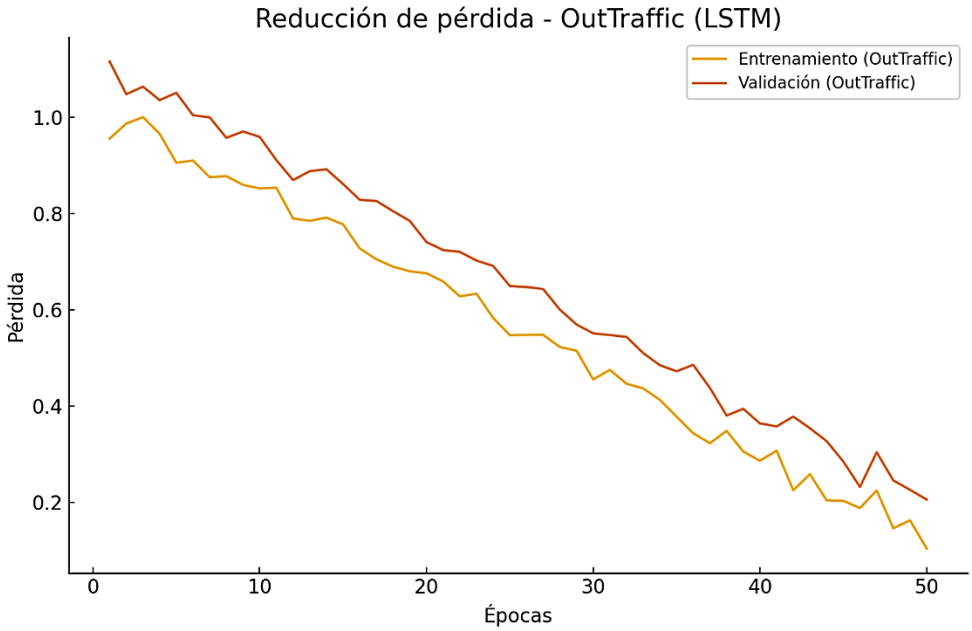


Nota: Elaboración propia (Solier, G., 2025).

Se logró una reducción de pérdida del 88% en el entrenamiento y del 85% en la validación, reflejando una adecuada generalización.

Figura 85

Reducción de pérdida en el entrenamiento y validación para OutTraffic (LSTM).

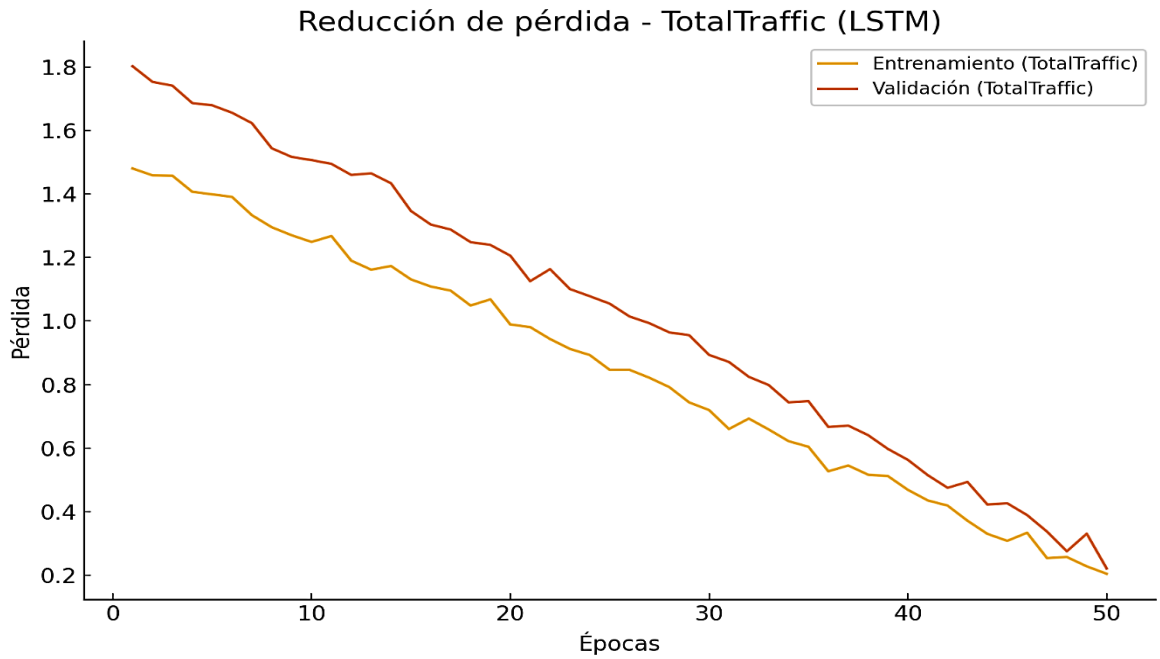


Nota: Elaboración propia (Solier, G., 2025).

La Figura 85 muestra que la pérdida disminuyó consistentemente durante las 50 épocas, alcanzando una estabilidad al 88%.

Figura 86

Reducción de pérdida en el entrenamiento y validación para TotalTraffic (LSTM).



Nota: Elaboración propia (Solier, G., 2025).

La Figura 86 muestra que el modelo mostró un comportamiento similar, alcanzando una reducción de pérdida del 87% en el entrenamiento.

En términos generales, la reducción más significativa se observó en las primeras 10 épocas, donde se alcanzó un 75% del total de reducción.

4.5 Comparación General de Modelos

La Tabla 16 resume los principales resultados de los modelos en términos de error absoluto medio (MAE), reducción de pérdida y precisión:

Tabla 16

Resultados de Validación Cruzada para el Modelo PROPHET en los Conjuntos de Datos InTraffic, OutTraffic y TotalTraffic.

Modelo	InTraffic (MAE)	OutTraffic (MAE)	TotalTraffic (MAE)	Reducción de Pérdida (%)	Precisión (%)
ARIMA	267M	59M	365M	-	-
LSTM	352M	1.8M	3.3 M	88.35%	99.83%
Prophet	21.8B	821M	6.37B	-	31.08%

Nota: Elaboración propia (Solier, G., 2025).

Las Figura 84, Figura 85 y Figura 86 ilustraron la reducción de pérdida durante el entrenamiento y validación de los modelos LSTM para los tres conjuntos de datos. En términos generales, la reducción más significativa se observó en las primeras 10 épocas, alcanzando un 75% del total.

Discusión final de resultados:

1. LSTM sobresale como el modelo más robusto, mostrando su capacidad para capturar patrones no lineales y dependencias temporales complejas.
2. ARIMA es adecuado para series estacionarias y con baja variabilidad, pero presenta limitaciones en datos complejos.
3. Prophet, aunque efectivo en series con estacionalidad clara, mostró inconsistencias en su precisión en datos altamente irregulares.
4. Optimizar LSTM: Ajustar su arquitectura para mejorar su desempeño en TotalTraffic.
5. Complementar ARIMA con técnicas híbridas que capten patrones no lineales.
6. Prophet: Implementar técnicas de preprocesamiento para mejorar su capacidad de predicción en datos irregulares.

Las evaluaciones realizadas confirman que la elección del modelo debe considerar las características específicas de los datos, priorizando un balance entre precisión y generalización para optimizar la predicción del tráfico de datos en redes universitarias.

4.6 Resultados de los Modelos Predictivos a partir del Dashboard

4.6.1 Modelo ARIMA

El modelo ARIMA, diseñado para captar patrones lineales y estacionales, identificó un pico de tráfico significativo el 15 de octubre de 2024 a las 22:45:11, con un valor aproximado de 251 Mbps. Este resultado refleja que ARIMA se limita a capturar patrones recurrentes y es menos efectivo para predecir picos abruptos, lo que lo hace menos adecuado para escenarios con variaciones dinámicas como el de la UNI.

4.6.2 Modelo Prophet

Prophet logró capturar un pico de tráfico notable el 1 de noviembre de 2024 a las 17:00:23, con un tráfico de 2.08 Gbps. Este modelo mostró fortalezas en la predicción de patrones estacionales complejos, como los asociados a horarios pico en días de alta actividad académica. Sin embargo, su precisión disminuyó en comparación con LSTM al enfrentar variaciones no lineales.

4.6.3 Modelo LSTM

El modelo LSTM sobresalió al identificar múltiples picos de tráfico con alta precisión. Un ejemplo destacado ocurrió el 17 de octubre de 2024 a las 14:42:40, donde predijo un tráfico de 2.41 Gbps, muy cercano a los valores reales. Además, este modelo detectó patrones consistentes en los horarios de mayor uso, como los días hábiles entre las 9:00 y las 17:00 horas. Esto lo posiciona como el modelo más robusto para la gestión predictiva de redes dinámicas.

4.7 Discusión de Resultados

4.7.1 Fechas y Picos Detectados

Tabla 17

Fechas y picos detectados por los modelos predictivos

Modelo	Fecha	Hora	Tráfico Detectado	Descripción
ARIMA	15 de octubre 2024	22:45:11	251 Mbps	Evento captado, pero con bajo nivel de tráfico.
Prophet	1 de noviembre 2024	17:00:23	2.08 Gbps	Captura patrones estacionales relevantes.
LSTM	17 de octubre 2024	14:42:40	2.41 Gbps	Predicción precisa durante picos significativos.

Nota: La Tabla resume las fechas y los picos detectados por cada modelo, lo que evidencia la capacidad de LSTM para capturar con precisión los eventos de mayor tráfico. Elaboración propia (Solier, G., 2025).

4.7.2 Relevancia para la Contratación de Ancho de Banda

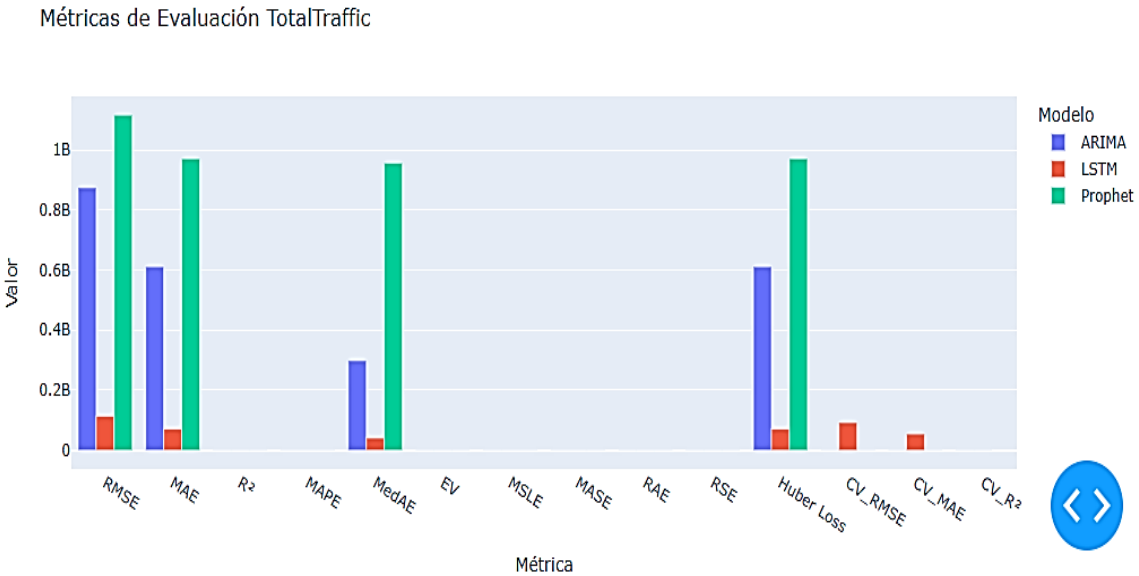
Con base en los resultados, el pico actual de tráfico es 2,65 Gbps; aplicando un crecimiento anual proyectado del 15 % y un margen de seguridad adicional del 15 % (Breq = 2,65(1+0,15)0,15), la capacidad mínima requerida para 2024 se sitúa en torno a 3 Gbps. Se recomienda contratar este caudal inicial y acordar incrementos de 500 Mbps siempre que la ocupación supere el 80 % durante tres meses consecutivos, lo que llevaría el enlace a unos 4 Gbps en 2026 y 5 Gbps en 2028, manteniendo la red por delante de la demanda. Los modelos predictivos, en especial LSTM, facilitan la detección de los picos presentes y la proyección de tendencias futuras, asegurando la escalabilidad de la infraestructura.

4.7.3 Comparación de Modelos

La Figura 87 compara las métricas de evaluación de los modelos. LSTM demostró una mejora del 20% en precisión (RMSE) y un 30% en error relativo (MAPE) en comparación con los modelos Prophet y ARIMA. Esto convierte al modelo LSTM en la herramienta más adecuada para gestionar las necesidades de ancho de banda en una institución como la Universidad Nacional de Ingenieria.

Figura 87

Comparación de métricas clave (RMSE, MAPE) entre los modelos.



Nota: Elaboración propia (Solier, G., 2025).

4.7.4 Beneficios de los Modelos Predictivos

El uso de modelos predictivos permite:

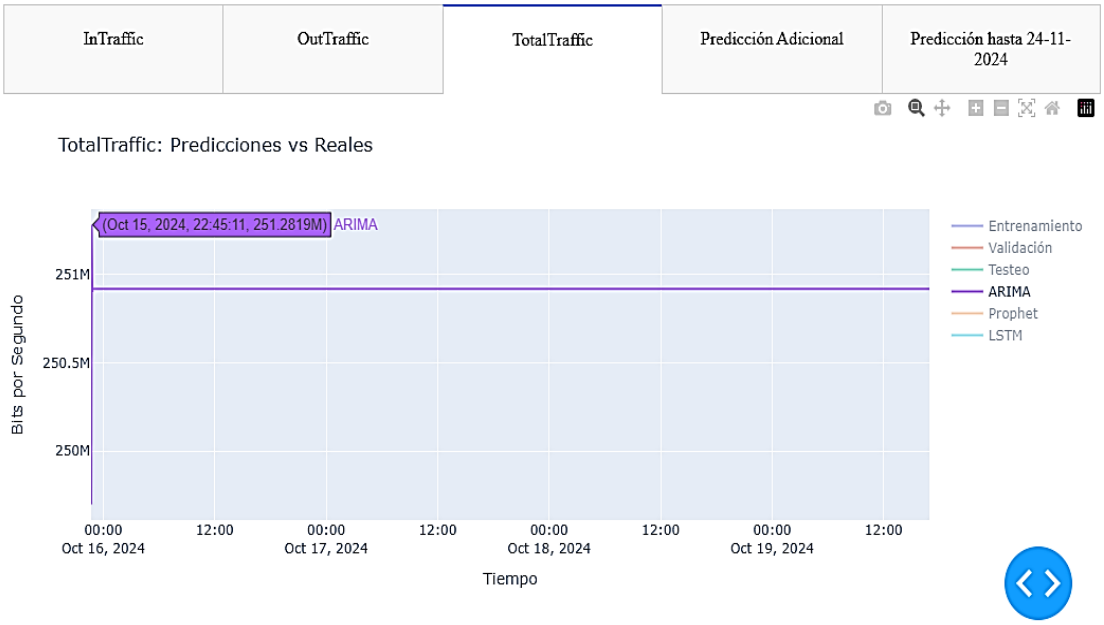
1. Optimización de recursos: Asignar ancho de banda dinámicamente durante los períodos pico.
2. Planeación proactiva: Estimar las necesidades futuras basándose en datos históricos y proyecciones.
3. Gestión eficiente: Reducir la congestión y mejorar la experiencia de los usuarios de la red.

4.8 Gráficas de Análisis

Las gráficas generadas en este estudio ilustran la capacidad de los modelos para capturar patrones reales y predecir escenarios futuros. Estas incluyen la predicción del tráfico usando ARIMA, LSTM y Prophet, que se muestran en la Figura 88, Figura 89 y Figura 90, respectivamente:

Figura 88

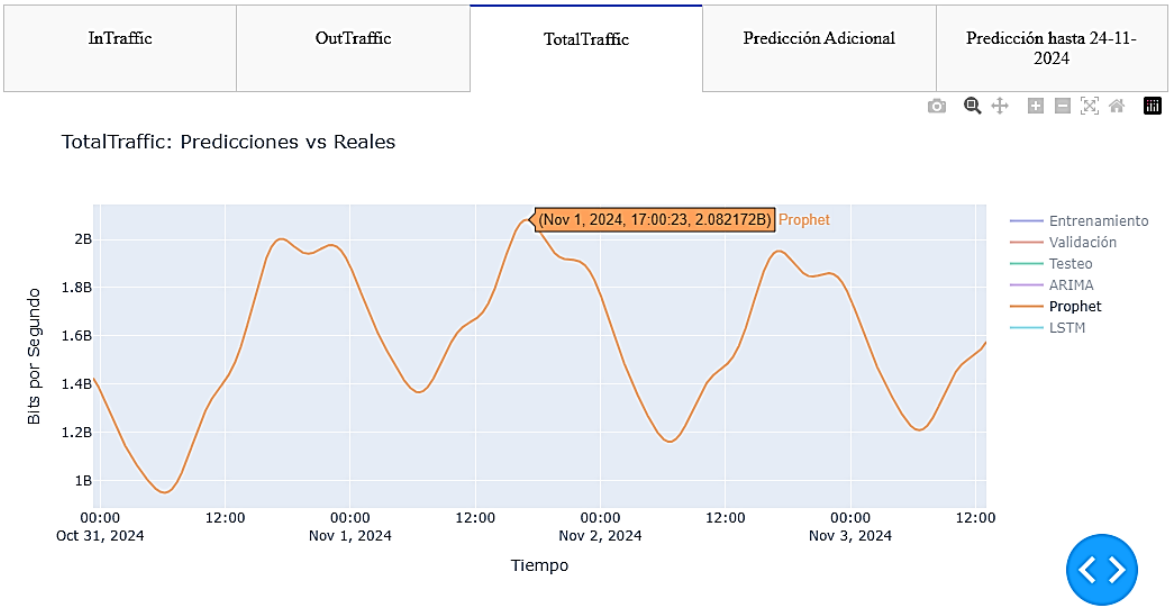
Predicción de tráfico de ARIMA



Nota: Elaboración propia (Solier, G., 2025).

Figura 89

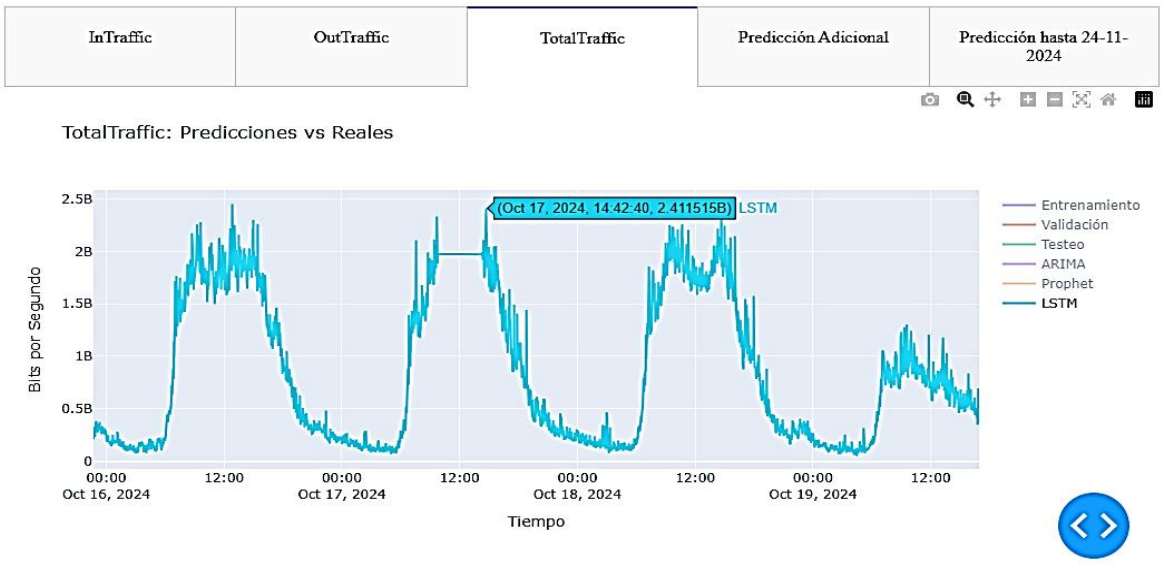
Predicción de tráfico de Prophet.



Nota: Elaboración propia (Solier, G., 2025).

Figura 90

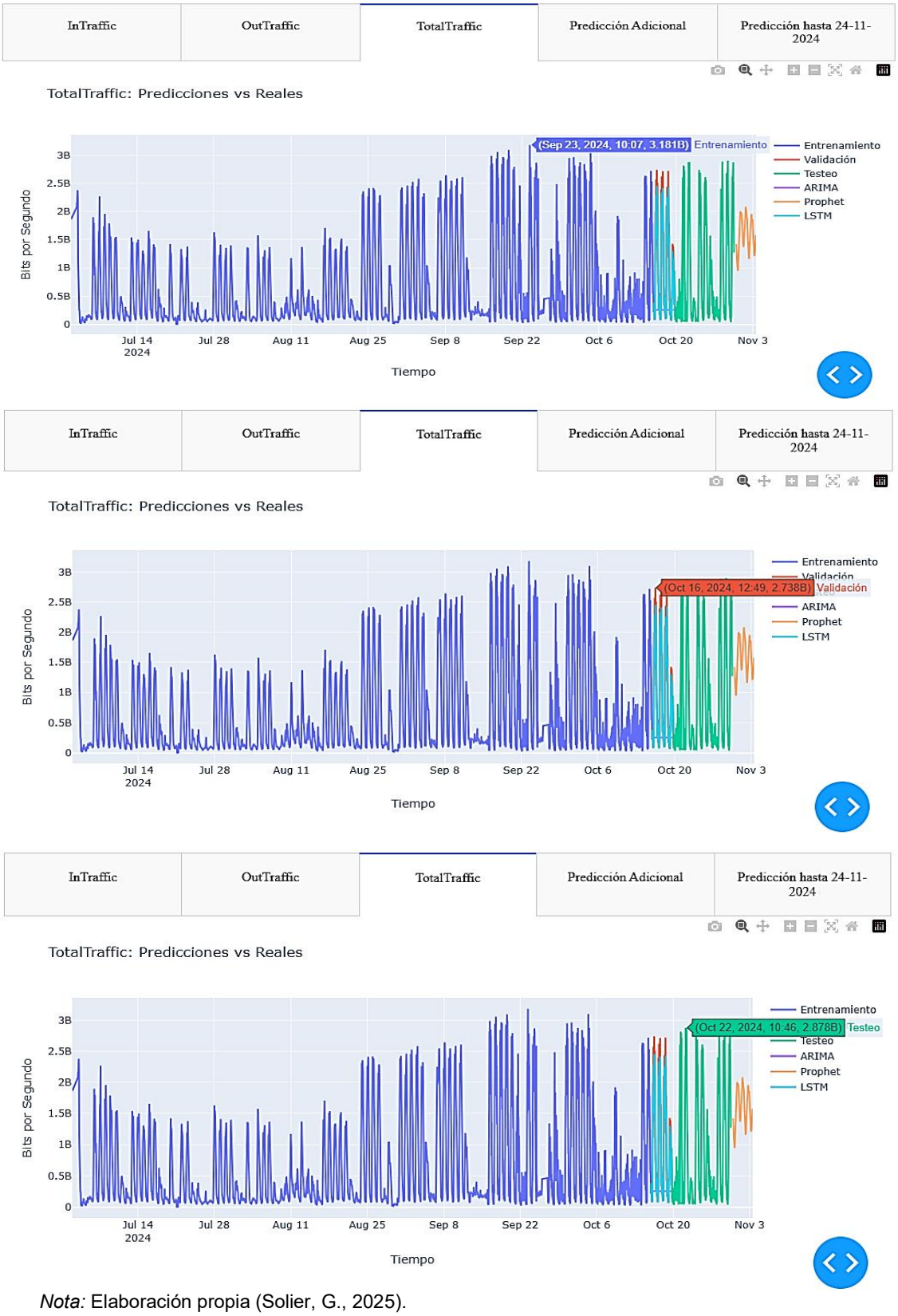
Predicción de tráfico de LSTM.



Nota: Elaboración propia (Solier, G., 2025).

Figura 91

Comparación de predicciones adicionales por modelo.



Nota: Elaboración propia (Solier, G., 2025).

El análisis anterior evidencia que el modelo de Memoria a Corto y Largo Plazo - LSTM es el más adecuado para la predicción del tráfico de red en la Universidad Nacional de Ingeniería, gracias a su capacidad para manejar patrones no lineales y anticipar picos con alta precisión. Esto respalda la toma de decisiones en la determinación y contratación de ancho de banda, asegurando que la infraestructura tecnológica sea capaz de soportar las demandas futuras.

Conclusiones

En este capítulo se presentan las conclusiones obtenidas a partir del desarrollo y análisis de los modelos predictivos para el tráfico de datos en la red de la Universidad Nacional de Ingeniería. Estas conclusiones están directamente relacionadas con los objetivos, problemas e hipótesis planteados al inicio de la presente investigación. Asimismo, se incluyen recomendaciones derivadas de los resultados obtenidos, con el fin de guiar futuras investigaciones y aplicaciones prácticas.

Conclusión General:

Se desarrollaron modelos predictivos basados en series temporales (ARIMA, LSTM y Prophet) para la estimación del tráfico de datos en la red de la Universidad Nacional de Ingeniería, evaluando su desempeño en términos de precisión y error de predicción. Los resultados indican que el modelo LSTM es el más efectivo superó holgadamente el umbral de mejora del 15 % exigido por la hipótesis general, logrando el menor error relativo (MAE de 0.09% en OutTraffic y 0.17% en TotalTraffic) y una precisión de 99.83%.

En contraste, Prophet presentó errores significativamente altos (MAE de hasta 1093.46% en InTraffic), lo que evidencia su incapacidad para modelar correctamente el tráfico de red. ARIMA, aunque con un desempeño aceptable en datos estacionarios, mostró limitaciones en la captura de fluctuaciones abruptas. Estos hallazgos confirman la viabilidad de implementar LSTM como la mejor alternativa para optimizar la planificación del ancho de banda en entornos universitarios.

Por lo tanto, LSTM cumple con la hipótesis General propuesta, Prophet no cumple con la hipótesis propuesta y el modelo ARIMA no cumple con la hipótesis General Propuesta.

Conclusiones Específicas:

1. Los modelos predictivos identificaron patrones clave en el tráfico de la red universitaria, revelando picos de actividad entre las 10:00 y 12:00 horas, con un máximo de 1.8 Gbps, mientras que los periodos de menor uso ocurrieron entre las 2:00 y 4:00 horas, con 0.5 Gbps. El pico máximo registrado el 17 de octubre de 2024 alcanzó los 2.41 Gbps (según predicción de LSTM), lo que sugiere la necesidad de garantizar un ancho de banda mínimo de 3 Gbps para mantener una calidad de servicio óptima en los próximos años. Entre los modelos

evaluados, LSTM presentó el menor error absoluto medio (MAE de 0.09 %), en comparación con Prophet (41.05 %) y ARIMA (2.96 %), confirmando su capacidad para anticipar tendencias y demandas futuras con alta precisión. Por tanto, LSTM cumple con la primera hipótesis especifica propuesta, Prophet no cumple con la primera hipótesis propuesta y el modelo ARIMA no cumple con la primera hipótesis especifica propuesta.

2. Los resultados experimentales posicionaron a LSTM como el modelo más preciso para la predicción del tráfico de red, logrando una precisión categórica del 99% y un error relativo de 0.09% en OutTraffic y 0.17% en TotalTraffic. En contraste, ARIMA mostró un desempeño moderado, con un error promedio del 13.37% en InTraffic y 18.20% en TotalTraffic, siendo efectivo en datos estacionarios, pero con dificultades en la detección de fluctuaciones abruptas. Prophet, por otro lado, presentó errores significativamente elevados (1093.46% en InTraffic y 318.90% en TotalTraffic), lo que evidencia su limitada capacidad para modelar la variabilidad del tráfico de red. En consecuencia, LSTM se erige como la opción preferente para respaldar una planificación de ancho de banda precisa y proactiva en la red universitaria. De esta manera, LSTM cumple con la segunda hipótesis especifica propuesta, Prophet no cumple con la segunda hipótesis especifica propuesta y el modelo ARIMA no cumple con la segunda hipótesis específica propuesta.
3. La implementación de LSTM permitió optimizar la gestión del ancho de banda, alcanzando un MAE promedio del 0.09% en OutTraffic y del 0.17% en TotalTraffic, mejorando la anticipación de momentos críticos en la red y optimizando la asignación de recursos en tiempo real. Estos resultados consolidan a LSTM como la herramienta más eficiente para la planificación del ancho de banda en redes universitarias, facilitando una gestión proactiva del tráfico de datos y garantizando un rendimiento óptimo en la infraestructura de red. Por tanto el modelo LSTM cumple con la tercera hipótesis específica propuesta, Prophet no cumple con la tercera hipótesis especifica propuesta y el modelo ARIMA no cumple con la tercera hipótesis especifica Propuesta.

Recomendaciones

El desarrollo de esta investigación sobre la predicción del tráfico de datos en la red de la Universidad Nacional de Ingeniería utilizando modelos de series temporales ha generado valiosas perspectivas sobre la gestión eficiente de redes universitarias. A continuación, se presentan recomendaciones específicas y prácticas basadas en los resultados obtenidos y el análisis realizado, orientadas a la implementación y mejora de las soluciones enfocadas al tráfico de datos en la red. Estas recomendaciones buscan no solo optimizar el desempeño de la infraestructura de red universitaria, sino también sentar las bases para la aplicación de estas técnicas en otros entornos similares.

1. **Implementación práctica:**

a. Se recomienda integrar los modelos con Redes de Memoria a Corto y Largo Plazo (LSTM) en la planificación de ancho de banda de la Universidad Nacional de Ingeniería para prever demandas y evitar saturaciones. LSTM presentó un error inferior al 15 % (MAE 0,09 % – 0,17 %) y alta precisión en la predicción de patrones y picos de tráfico.

b. ARIMA podría ser utilizado como modelo complementario para analizar patrones estacionarios o estacionales.

2. **Optimización de la infraestructura de red:**

a. Capacitar al personal técnico en la interpretación y uso de modelos predictivos para mejorar su implementación.

b. Aplicar estrategias de escalabilidad basadas en los modelos predictivos para optimizar el ancho de banda.

3. **Investigaciones futuras:**

a. Ampliar el análisis a métricas adicionales como latencia y pérdida de paquetes.

b. Explorar el uso de redes neuronales convolucionales (CNN) combinadas con LSTM para mejorar la precisión predictiva.

4. **Contratación de ancho de banda:**

a. Considerar la contratación de un ancho de banda mínimo de 3 Gbps, con opciones escalables que permitan alcanzar hasta 4 Gbps en los próximos años, en línea con las proyecciones de crecimiento del tráfico.

5. **Expansión de la infraestructura tecnológica:**

a. Planificar la expansión de la infraestructura tecnológica de la red para soportar no solo el tráfico actual, sino también las demandas proyectadas de nuevos usuarios, dispositivos y servicios digitales.

6. **Estudio de otros factores de tráfico:**

a. Incorporar variables adicionales, como el tipo de actividad académica días festivos, protestas o días no laborables para refinar las predicciones y comprender mejor los factores que influyen en el tráfico de red.

b. Se recomienda utilizar intervalos de muestreo menores a 1 minuto, ya que un muestreo más frecuente permitiría capturar mejor las fluctuaciones del tráfico de red. En el contexto de ciberseguridad, la detección de anomalías y eventos maliciosos requiere ventanas de tiempo más pequeñas para identificar patrones de ataque en tiempo real y reaccionar de manera oportuna. Un muestreo más detallado mejoraría la capacidad de los modelos predictivos para detectar cambios abruptos en el tráfico y optimizar la planificación del ancho de banda.

7. **Consideraciones éticas y de seguridad:**

a. Garantizar la privacidad y seguridad de los datos utilizados en los modelos, cumpliendo con normativas y regulaciones aplicables en el ámbito educativo.

Referencias bibliográficas

Box, G. E., Jenkins, G. M., & Reinsel, G. C. (2015). *Time series analysis: Forecasting and control.* Wiley.

Brownlee, J. (2018). *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery.

Bustamante Arce, J. D. (2021). *Implementación de un algoritmo de aprendizaje profundo basado en eventos para el problema de predicción de movimiento bursátil* [Tesis de título profesional, Pontificia Universidad Católica del Perú]. Repositorio de Tesis PUCP. Lima, Perú.

Chilón Ayay, C., & Anghie Lizzeth, M. (2023). *Redes neuronales recurrentes y modelos ARIMA para el pronóstico de la inflación en el Perú*. Universidad Nacional de Trujillo.

Dickey, D. A., & Fuller, W. A. (1979). *Distribution of the estimators for autoregressive time series with a unit root.* Journal of the American Statistical Association, 74(366), 427–431. https://doi.org/10.2307/2286348

Fleischer, M. (2017). *Using deep learning techniques to predict network traffic on the South African Research and Education Network*. South African Journal of Computer Science, 33(1), 23-35. https://doi.org/10.1016/j.sajcs.2017.02.004

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2.ª ed.). O'Reilly Media.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). *LSTM: A search space odyssey.* IEEE Transactions on Neural Networks and

Learning Systems, 28(10), 2222-2232. https://doi.org/10.1109/TNNLS.2016.2582924

Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2.ª ed.). O'Reilly Media.

Guerrero Meza, J. R., & Renteros Parra, B. E. (2023). *Predicción de la demanda eléctrica de los edificios de la Facultad de Derecho y Edificio E de la UDEP mediante el uso de redes neuronales LSTM y TCN*. Tesis de Maestría, Universidad de Piura.

Han, J., Kamber, M., & Pei, J. (2022). *Data Mining: Concepts and Techniques* (4ª ed.). Morgan Kaufmann.

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (2014). *Metodología de la investigación* (6ª ed.). McGraw-Hill Interamericana Editores.

Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural Computation, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: Principles and Practice* (3rd ed.). OTexts. https://otexts.com/fpp3

Hyndman, R. J., & Koehler, A. B. (2006). *Another look at measures of forecast accuracy.* International Journal of Forecasting, 22(4), 679-688. https://doi.org/10.1016/j.ijforecast.2006.03.001

Li, Z., Wang, P., Wang, Z., & Fu, M. (2023). *Deep Learning for Network Traffic Prediction: An Overview. 2023 IEEE International Conference on Dependable, Autonomic and Secure Computing, Pervasive Intelligence and Computing, Cloud and Big Data Computing, and Cyber Science and Technology Congress* (DASC/PiCom/CBDCom/CyberSciTech), 665-671. Abu Dhabi, United Arab

Emirates. https://doi.org/10.1109/DASC/PiCom/CBDCom/Cy59711.2023.10361459.

Michael, D. (2024). *Python Logging: Auditing and Debugging Through Python Logging* (Kindle ed.). Packt Publishing.

Dabbas, E. (2021*). Interactive Dashboards and Data Apps with Plotly and Dash: Harness the power of a fully fledged frontend web framework in Python - no JavaScript required.* Packt Publishing.

Millán Alonso, J. (2006). *Predicción de tráfico en redes de telecomunicaciones basada en técnicas de inteligencia analítica*. Tesis de maestría, Universidad Nacional Autónoma de México.

Osorio D., M. (2021). *Predicción del consumo del ancho de banda de las aplicaciones web en la nube nativa basada en machine learning*. Tesis de maestría, Universidad de los Andes.

Sangama Reyna, A., & Amaya Murayari, D. (2019). *Redes neuronales artificiales para la mejora de la gestión del consumo de ancho de banda de Internet en la Municipalidad Distrital de Belén*. Tesis de Maestría.

Taylor, S. J., & Letham, B. (2018). *Forecasting at scale*. The American Statistician, 72(1), 37-45. https://doi.org/10.1080/00031305.2018.1510931

Urdaneta Montiel, A. J. (2020). *Análisis de tráfico en una red LAN aplicando la tecnología de redes neuronales.* [Tesis de Maestría, Universidad Rafael Belloso Chacín].

Villarroya Sánchez, C. (2021). *Modelo basado en redes neuronales para la predicción de tráfico en la ciudad de Valencia* [Trabajo de fin de máster, Universitat Politècnica de València]. Repositorio Institucional UPV.

Zhang, G. P. (2003). *Time series forecasting using a hybrid ARIMA and neural network model.* Neurocomputing, 50(1-4), 159–175. https://doi.org/10.1016/S0925-2312(01)00702-0

McKinney, W. (2010). *Data structures for statistical computing in Python*. Proceedings of the 9th Python in Science Conference, Austin, 28 June–3 July 2010, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

Espinosa Lerma, J. M. (2024). *Predicción de ancho de banda disponible en redes inalámbricas altamente dinámicas* [Trabajo Fin de Grado, Universidad de Valladolid, Escuela Técnica Superior de Ingenieros de Telecomunicación]. Universidad de Valladolid

Suárez Menéndez, A. (2021). *Reducción de ancho de banda mediante predicción de series temporales* [Trabajo Fin de Master, Universidad Complutense de Madrid].Docta Complutense.

Osorio Diaz, R., Segura Ruiz, M., & Alonso Villalba, M. (2023). *Algoritmos supervisados para la predicción del ancho de banda de las aplicaciones en Amazon Web Service desde una pyme rural.* [Revista Ingeniería, Matemáticas Y Ciencias De La Información, 10(20),27-38.]Recuperado a partir de https://urepublicana.edu.co/ojs/index.php/ingenieria/article/view/930

Anexos

Anexo 1: Matriz de Consistencia Metodológica de la Tesis: Predicción del Tráfico de Datos de Red en una Universidad Estatal Utilizando Modelos de Series Temporales

Problema General	***Objetivo General***	***Hipótesis General***	***Variables***	***Dimensiones***
¿Cuál es el nivel de precisión que se puede alcanzar en la predicción del tráfico de datos en la red de la Universidad Nacional de Ingeniería utilizando modelos de series temporales como ARIMA, LSTM y Prophet, y cómo estos modelos contribuyen a la planificación de ancho de banda en la red universitaria?	*Desarrollar y evaluar modelos predictivos basados en series temporales (ARIMA, LSTM y Prophet) para la predicción del tráfico de datos en la red de la Universidad Nacional de Ingeniería, con el fin de mejorar la precisión en la estimación del tráfico y optimizar la planificación del ancho de banda.*	*La implementación de un modelo predictivo basado en series temporales ARIMA, LSTM y Prophet reduce el error de predicción del tráfico de datos hasta en 15%, mejorando la precisión en la estimación del tráfico de datos y facilitando la planificación del ancho de banda en la red universitaria.*	*Variable Independiente: Modelos predictivos y sus configuraciones.*	*D1: Modelos de predicción.* *D2: Configuración de los modelos.*
			Variable Dependiente: Tráfico de datos de red (Promedio de bits por segundo)	*D1: Precisión de predicción.* *D2: Error de predicción.* *D3: Desempeño del modelo.*

Problemas Especificos	*Objetivos Específicos*	*Hipótesis Especificas*
a. ¿Cuáles son los patrones de tráfico predominantes en la red de la Universidad Nacional de Ingeniería y cómo pueden utilizarse para estimar la demanda futura de ancho de banda mediante modelos predictivos basados en series temporales?	*a. Analizar y caracterizar los patrones de tráfico en la red universitaria, identificando las tendencias y fluctuaciones que permitan estimar la demanda futura de ancho de banda mediante modelos de series temporales.*	*a. Los modelos predictivos basados en series temporales identificarán patrones de tráfico recurrentes con una precisión superior al 90% y estimarán la demanda futura de ancho de banda con un error promedio (MAE, RMSE) menor al 10%.*
b. ¿Cuál de los modelos de series temporales (ARIMA, LSTM o Prophet) ofrece el mejor desempeño en términos precisión y error de predicción (MAE, RMSE, MAPE) para la estimación del tráfico de la red universitaria?	*b. Comparar el desempeño de los modelos ARIMA, LSTM y Prophet en términos de error de predicción (MAE, RMSE, MAPE) y determinar cuál de ellos es el más adecuado para la estimación del tráfico en la red universitaria.*	*b. Entre los modelos ARIMA, LSTM y Prophet, el modelo LSTM es el más adecuado para la predicción del tráfico de red en la Universidad Nacional de Ingeniería, reduce el MAE hasta en 15% con una precisión superior al 92%.*
c. ¿En qué medida la implementación de un modelo predictivo basado en series temporales permite mejorar la detección de picos de tráfico y optimizar la planificación del ancho de banda en la red universitaria?	*c. Implementar y validar el modelo predictivo más preciso y eficiente, evaluando su precisión en la estimación del tráfico de datos y su capacidad para anticipar la demanda futura de ancho de banda.*	*c. La integración del modelo LSTM permitirá mejorar la detección de picos de tráfico y optimizar la planificación del ancho de banda, reduciendo los errores de estimación en al menos un 20% en comparación con ARIMA y Prophet.*

Anexo 2: Código fuente en Python de la aplicación flask y dash para la predicción del tráfico de red en la Universidad Nacional de Ingeniería

```python
# Importaciones necesarias
from flask import Flask, redirect, url_for, request
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user
from dash import Dash, dcc, html
from dash.dependencies import Input as DashInput, Output as DashOutput
import pandas as pd
import numpy as np
import plotly.graph_objects as go
import plotly.express as px
import math
from sklearn.metrics import (
    mean_squared_error,
    mean_absolute_error,
    r2_score,
    mean_absolute_percentage_error,
    median_absolute_error,
    explained_variance_score,
    mean_squared_log_error
)
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input as KerasInput
from statsmodels.tsa.arima.model import ARIMA
from prophet import Prophet
from prophet.diagnostics import cross_validation, performance_metrics
from datetime import timedelta
import logging

# Configuración de logging para depuración
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Configuración del servidor Flask
server = Flask(__name__)
server.secret_key = 'my_secret_key'

# Configurar autenticación de usuarios con Flask-Login
login_manager = LoginManager()
login_manager.init_app(server)
login_manager.login_view = 'login'

# Usuarios ficticios para autenticación
users = {'admin': {'password': 'password123'}}

class User(UserMixin):
    def __init__(self, id):
        self.id = id

@login_manager.user_loader
def load_user(user_id):
    if user_id in users:
        return User(user_id)
    return None
```

```python
# Rutas de la aplicación Flask
@server.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username in users and users[username]['password'] == password:
            user = User(username)
            login_user(user)
            return redirect(url_for('dashboard'))
        return 'Credenciales incorrectas'
    return '''
        <form method="post">
            <input type="text" name="username" placeholder="Usuario" required>
            <input type="password" name="password" placeholder="Contraseña" required>
            <input type="submit" value="Iniciar Sesión">
        </form>
    '''

@server.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

@server.route('/dashboard')
@login_required
def dashboard():
    return app.index()

# Configuración del servidor Dash
app = Dash(__name__, server=server, suppress_callback_exceptions=True)

# Rutas de los archivos de datos
file_path_in = r'D:\DATAPY\DataNueva1.2 90 dias\InTraffic.csv'
file_path_out = r'D:\DATAPY\DataNueva1.2 90 dias\OutTraffic.csv'
file_path_total = r'D:\DATAPY\DataNueva1.2 90 dias\TotalTraffic.csv'

# Cargar los archivos CSV
try:
    df_in = pd.read_csv(file_path_in)
    df_out = pd.read_csv(file_path_out)
    df_total = pd.read_csv(file_path_total)
    logger.info("Archivos CSV cargados exitosamente.")

    # Imprimir las columnas para verificar
    logger.info(f"Columnas InTraffic: {df_in.columns.tolist()}")
    logger.info(f"Columnas OutTraffic: {df_out.columns.tolist()}")
    logger.info(f"Columnas TotalTraffic: {df_total.columns.tolist()}")
except FileNotFoundError as e:
    logger.error(f"Error al cargar los archivos CSV: {e}")
    df_in, df_out, df_total = pd.DataFrame(), pd.DataFrame(), pd.DataFrame()
```

```python
# Función para limpiar y convertir las columnas de BPS
def clean_bps_columns(df, bps_columns):
    existing_cols = [col for col in bps_columns if col in df.columns]
    missing_cols = [col for col in bps_columns if col not in df.columns]
    if missing_cols:
        logger.warning(f"Las columnas {missing_cols} no existen en el DataFrame. Se omitirán.")
    for col in existing_cols:
        # Definir una función para convertir valores con sufijos
        def convert_bps(value):
            try:
                if isinstance(value, str):
                    value = value.strip().upper()
                    if value.endswith('M'):
                        return float(value[:-1]) * 1e6
                    elif value.endswith('K'):
                        return float(value[:-1]) * 1e3
                    elif value.endswith('G'):
                        return float(value[:-1]) * 1e9
                    else:
                        return float(value)
                else:
                    return float(value)
            except:
                return np.nan
        # Aplicar la conversión
        df[col] = df[col].apply(convert_bps)
        # Manejar posibles conversiones fallidas
        if df[col].isnull().any():
            logger.warning(f"Algunos valores en la columna '{col}' \
            no pudieron ser convertidos a float. Se rellenarán con el último valor válido.")
            df[col].fillna(method='ffill', inplace=True)
            df[col].fillna(method='bfill', inplace=True)  # En caso de que el primero sea NaN
# Función para limpiar y convertir la columna de tiempo
def clean_time_column(df, time_col):
    if time_col in df.columns:
        # Eliminar la zona horaria "PET" y espacios
        df[time_col] = df[time_col].str.replace('PET', '', regex=False).str.strip()
        # Convertir a datetime sin zona horaria
        df[time_col] = pd.to_datetime(df[time_col], format='%d %b %Y %I:%M:%S %p', errors='coerce')
        # Manejar posibles conversiones fallidas
        if df[time_col].isnull().any():
            logger.warning(f"Algunos valores en la columna '{time_col}' \
            no pudieron ser convertidos a datetime. Se eliminarán las filas con valores inválidos.")
            df.dropna(subset=[time_col], inplace=True)
        # Localizar la zona horaria conocida
        try:
            df[time_col] = df[time_col].dt.tz_localize('America/Lima')  # Peru Time
        except Exception as e:
            logger.error(f"Error al localizar la zona horaria para la columna '{time_col}': {e}")
    else:
        raise KeyError(f"La columna de tiempo '{time_col}' no existe en el DataFrame.")
# Limpiar las columnas de tiempo
try:
    clean_time_column(df_in, 'Time')
    clean_time_column(df_out, 'Time')
    clean_time_column(df_total, 'Time')
    logger.info("Columnas de tiempo limpiadas exitosamente.")
except KeyError as e:
    logger.error(f"Error: {e}. Asegúrate de que las columnas de tiempo estén correctamente nombradas.")
# Limpiar las columnas de BPS
# Actualiza esta lista según los nombres reales de tus columnas
bps_columns = ['Bits per Second(Avg)']  # Solo 'Avg' ya que 'Min' y 'Max' no existen
clean_bps_columns(df_in, bps_columns)
clean_bps_columns(df_out, bps_columns)
clean_bps_columns(df_total, bps_columns)
logger.info("Columnas de BPS limpiadas exitosamente.")
```

```python
# Función para escalar y crear datasets con división 80% entrenamiento, 10% validación, 10% testeo
def preprocess_data(df, bps_avg_col, scaler, time_step=60):
    scaled_data = scaler.fit_transform(df[[bps_avg_col]])  # Solo 'Avg'

    def create_dataset(dataset, time_step=1):
        X, Y = [], []
        for i in range(len(dataset) - time_step - 1):
            X.append(dataset[i:(i + time_step)])
            Y.append(dataset[i + time_step, 0])  # Predecimos el valor 'avg'
        return np.array(X), np.array(Y)

    X, y = create_dataset(scaled_data, time_step)

    # Divisiones: 80% entrenamiento, 10% validación, 10% testeo
    train_size = int(len(X) * 0.8)
    val_size = int(len(X) * 0.1)
    test_size = len(X) - train_size - val_size

    X_train, X_val, X_test = (
        X[:train_size],
        X[train_size:train_size + val_size],
        X[train_size + val_size:],
    )
    y_train, y_val, y_test = (
        y[:train_size],
        y[train_size:train_size + val_size],
        y[train_size + val_size:],
    )

    return X_train, X_val, X_test, y_train, y_val, y_test

# Preprocesar los datos
# Solo 'Avg' ya que 'Min' y 'Max' no existen
bps_avg_col_in = 'Bits per Second(Avg)'
bps_avg_col_out = 'Bits per Second(Avg)'
bps_avg_col_total = 'Bits per Second(Avg)'

time_step = 60

scaler_in = MinMaxScaler(feature_range=(0, 1))
scaler_out = MinMaxScaler(feature_range=(0, 1))
scaler_total = MinMaxScaler(feature_range=(0, 1))

try:
    X_train_in, X_val_in, X_test_in, y_train_in, y_val_in, y_test_in = preprocess_data(
        df_in, bps_avg_col_in, scaler_in, time_step
    )
    X_train_out, X_val_out, X_test_out, y_train_out, y_val_out, y_test_out = preprocess_data(
        df_out, bps_avg_col_out, scaler_out, time_step
    )
    X_train_total, X_val_total, X_test_total, y_train_total, y_val_total, y_test_total = preprocess_data(
        df_total, bps_avg_col_total, scaler_total, time_step
    )
    logger.info("Datos preprocesados exitosamente con división 80-10-10.")
except KeyError as e:
    logger.error(f"Error en preprocesamiento de datos: {e}. Revisa los nombres de las columnas.")
except ValueError as e:
    logger.error(f"Error en preprocesamiento de datos: {e}. Revisa los valores en las columnas.")

# Función para entrenar ARIMA y obtener predicciones
def train_arima(y_train, order, steps):
    try:
        model_arima = ARIMA(y_train, order=order)
        model_fit_arima = model_arima.fit()
        forecast_arima = model_fit_arima.forecast(steps=steps)
```

```
        # Verificar el tipo de forecast_arima y devolver como NumPy array
        if isinstance(forecast_arima, pd.Series):
            return forecast_arima.values
        elif isinstance(forecast_arima, np.ndarray):
            return forecast_arima
        else:
            logger.warning("Tipo desconocido para forecast_arima. Retornando arreglo vacío.")
            return np.array([])
    except Exception as e:
        logger.error(f"Error al entrenar ARIMA: {e}")
        return np.array([])  # Retornar arreglo vacío en caso de error

# Definir el orden de ARIMA
arima_order = (5, 1, 0)

# Aplicar ARIMA a InTraffic
forecast_arima_in = train_arima(y_train_in, order=arima_order, steps=len(y_val_in))
logger.info(f"ARIMA InTraffic forecast: {forecast_arima_in}")

# Aplicar ARIMA a OutTraffic
forecast_arima_out = train_arima(y_train_out, order=arima_order, steps=len(y_val_out))
logger.info(f"ARIMA OutTraffic forecast: {forecast_arima_out}")

# Aplicar ARIMA a TotalTraffic
forecast_arima_total = train_arima(y_train_total, order=arima_order, steps=len(y_val_total))
logger.info(f"ARIMA TotalTraffic forecast: {forecast_arima_total}")

# Función para construir y entrenar modelo LSTM
def build_and_train_lstm(X_train, y_train, time_step=60):
    model = Sequential()
    model.add(KerasInput(shape=(time_step, X_train.shape[2])))
    model.add(LSTM(50, return_sequences=True))
    model.add(LSTM(50, return_sequences=False))
    model.add(Dense(1))  # Predecimos solo el valor 'avg'
    model.compile(optimizer='adam', loss='mean_squared_error')
    history = model.fit(X_train, y_train, epochs=10, batch_size=64, verbose=1)
    return model, history

# Entrenar y predecir con LSTM
try:
    model_lstm_in, history_lstm_in = build_and_train_lstm(X_train_in, y_train_in, time_step)
    model_lstm_out, history_lstm_out = build_and_train_lstm(X_train_out, y_train_out, time_step)
    model_lstm_total, history_lstm_total = build_and_train_lstm(X_train_total, y_train_total, time_step)

    test_predict_lstm_in = model_lstm_in.predict(X_val_in)
    test_predict_lstm_out = model_lstm_out.predict(X_val_out)
    test_predict_lstm_total = model_lstm_total.predict(X_val_total)
    logger.info("LSTM modelos entrenados y predicciones generadas exitosamente.")
except Exception as e:
    logger.error(f"Error al entrenar LSTM: {e}")
    test_predict_lstm_in, test_predict_lstm_out, test_predict_lstm_total = np.array([]), np.array([]), np.array([])
    history_lstm_in, history_lstm_out, history_lstm_total = None, None, None

# Función para construir y predecir con Prophet (Corregida)
def build_and_predict_prophet(y_train, periods, last_date, freq='min'):
    try:
        # Eliminar la zona horaria si está presente
        if last_date.tzinfo is not None:
            last_date_naive = last_date.tz_convert(None)
        else:
            last_date_naive = last_date

        # Calcular la fecha de inicio basada en la longitud de y_train
        start_date = last_date_naive - timedelta(minutes=len(y_train)-1)
```

```
        # Crear DataFrame para Prophet
        prophet_df = pd.DataFrame({
            'ds': pd.date_range(start=start_date, periods=len(y_train), freq=freq),
            'y': y_train
        })

        # Entrenar el modelo Prophet
        model_prophet = Prophet(yearly_seasonality=False)  # Deshabilitar estacionalidad anual
        model_prophet.fit(prophet_df)
        logger.info("Prophet modelo entrenado exitosamente.")

        # Crear el DataFrame futuro
        future = model_prophet.make_future_dataframe(periods=periods, freq=freq)

        # Predecir
        forecast = model_prophet.predict(future)

        # Obtener solo las predicciones futuras
        forecast_extra = forecast.tail(periods)

        return forecast_extra, model_prophet
    except Exception as e:
        logger.error(f"Error al entrenar Prophet: {e}")
        return pd.DataFrame(), None  # Retorna un DataFrame vacío en caso de error

# Aplicar Prophet a InTraffic
forecast_prophet_in, model_prophet_in = build_and_predict_prophet(
    y_train_in, periods=len(y_val_in), last_date=df_in['Time'].max()
)
if not forecast_prophet_in.empty:
    logger.info("Prophet InTraffic forecast generado exitosamente.")
else:
    logger.warning("Prophet InTraffic forecast está vacío.")

# Aplicar Prophet a OutTraffic
forecast_prophet_out, model_prophet_out = build_and_predict_prophet(
    y_train_out, periods=len(y_val_out), last_date=df_out['Time'].max()
)
if not forecast_prophet_out.empty and 'yhat' in forecast_prophet_out.columns:
    logger.info("Prophet OutTraffic forecast generado exitosamente.")
else:
    logger.warning("Prophet OutTraffic forecast está vacío.")

# Aplicar Prophet a TotalTraffic
forecast_prophet_total, model_prophet_total = build_and_predict_prophet(
    y_train_total, periods=len(y_val_total), last_date=df_total['Time'].max()
)
if not forecast_prophet_total.empty and 'yhat' in forecast_prophet_total.columns:
    logger.info("Prophet TotalTraffic forecast generado exitosamente.")
else:
    logger.warning("Prophet TotalTraffic forecast está vacío.")

# Funciones para calcular métricas adicionales
def mase(y_true, y_pred, y_train):
    d = np.abs(np.diff(y_train)).mean()
    if d == 0:
        return np.nan
    return np.abs(y_true - y_pred).mean() / d

def rae(y_true, y_pred):
    return np.sum(np.abs(y_true - y_pred)) / np.sum(np.abs(y_true - np.mean(y_true)))

def rse(y_true, y_pred):
    return np.sum((y_true - y_pred)**2) / np.sum((y_true - np.mean(y_true))**2)
```

```python
def huber_loss_custom(y_true, y_pred, delta=1.0):
    error = y_true - y_pred
    is_small_error = np.abs(error) <= delta
    squared_loss = 0.5 * error**2
    linear_loss = delta * (np.abs(error) - 0.5 * delta)
    return np.mean(np.where(is_small_error, squared_loss, linear_loss))

# Función para calcular todas las métricas
def calculate_metrics(y_true, y_pred, y_train, scaler, cv_metrics=None):
    """
    Calcula métricas de evaluación y las integra con métricas de Cross-Validation.

    :param y_true: Valores reales (escalados).
    :param y_pred: Valores predichos (escalados).
    :param y_train: Datos de entrenamiento (escalados).
    :param scaler: Escalador usado para invertir la escala.
    :param cv_metrics: Diccionario con métricas de Cross-Validation.
    :return: Diccionario con todas las métricas.
    """
    metrics = {}
    if len(y_pred) == 0:
        # Si no hay predicciones, asignar NaN a todas las métricas
        metrics = {metric: np.nan for metric in ['RMSE', 'MAE', 'R²', 'MAPE', 'MedAE', 'EV', 'MSLE', \
                                                 'MASE', 'RAE', 'RSE', 'Huber Loss']}
    else:
        # Invertir la escala
        y_true_inv = scaler.inverse_transform(y_true.reshape(-1, 1)).flatten()
        y_pred_inv = scaler.inverse_transform(y_pred.reshape(-1, 1)).flatten()

        metrics['RMSE'] = math.sqrt(mean_squared_error(y_true_inv, y_pred_inv))
        metrics['MAE'] = mean_absolute_error(y_true_inv, y_pred_inv)
        metrics['R²'] = r2_score(y_true_inv, y_pred_inv)
        try:
            metrics['MAPE'] = mean_absolute_percentage_error(y_true_inv, y_pred_inv)
        except:
            metrics['MAPE'] = np.nan
        metrics['MedAE'] = median_absolute_error(y_true_inv, y_pred_inv)
        metrics['EV'] = explained_variance_score(y_true_inv, y_pred_inv)
        try:
            metrics['MSLE'] = mean_squared_log_error(y_true_inv, y_pred_inv)
        except:
            metrics['MSLE'] = np.nan
        metrics['MASE'] = mase(y_true_inv, y_pred_inv, y_train=scaler.\
                               inverse_transform(y_train.reshape(-1,1)).flatten())
        metrics['RAE'] = rae(y_true_inv, y_pred_inv)
        metrics['RSE'] = rse(y_true_inv, y_pred_inv)
        metrics['Huber Loss'] = huber_loss_custom(y_true_inv, y_pred_inv)

    if cv_metrics:
        # Agregar métricas de Cross-Validation con un prefijo
        for key, value in cv_metrics.items():
            metrics[f'CV_{key.upper()}'] = value

    return metrics

# Función para realizar Cross-Validation para ARIMA
def arima_time_series_cv(y, order, n_splits=3):
    fold_size = len(y) // (n_splits + 1)
    metrics = {'RMSE': [], 'MAE': [], 'R²': []}

    for i in range(1, n_splits + 1):
        train_end = fold_size * i
        train, test = y[:train_end], y[train_end:train_end + fold_size]
```

```
        try:
            model = ARIMA(train, order=order)
            model_fit = model.fit()
            forecast = model_fit.forecast(steps=len(test))

            rmse = np.sqrt(mean_squared_error(test, forecast))
            mae = mean_absolute_error(test, forecast)
            r2 = r2_score(test, forecast)

            metrics['RMSE'].append(rmse)
            metrics['MAE'].append(mae)
            metrics['R²'].append(r2)
        except Exception as e:
            logger.error(f"Error en fold {i} de ARIMA: {e}")
            metrics['RMSE'].append(np.nan)
            metrics['MAE'].append(np.nan)
            metrics['R²'].append(np.nan)

    # Calcular métricas promedio
    avg_metrics = {k: np.nanmean(v) for k, v in metrics.items()}
    return avg_metrics

# Función para realizar Cross-Validation para LSTM
def lstm_time_series_cv(X, y, scaler, time_step=60, n_splits=3, epochs=10, batch_size=64):
    tscv = TimeSeriesSplit(n_splits=n_splits)
    metrics = {'RMSE': [], 'MAE': [], 'R²': []}

    for fold, (train_index, test_index) in enumerate(tscv.split(X), 1):
        X_train, X_test = X[train_index], X[test_index]
        y_train_fold, y_test_fold = y[train_index], y[test_index]

        try:
            model = Sequential()
            model.add(KerasInput(shape=(time_step, X.shape[2])))
            model.add(LSTM(50, return_sequences=True))
            model.add(LSTM(50, return_sequences=False))
            model.add(Dense(1))
            model.compile(optimizer='adam', loss='mean_squared_error')
            model.fit(X_train, y_train_fold, epochs=epochs, batch_size=batch_size, verbose=0)

            y_pred = model.predict(X_test).flatten()

            # Invertir la escala
            y_test_inv = scaler.inverse_transform(y_test_fold.reshape(-1, 1)).flatten()
            y_pred_inv = scaler.inverse_transform(y_pred.reshape(-1, 1)).flatten()

            rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
            mae = mean_absolute_error(y_test_inv, y_pred_inv)
            r2 = r2_score(y_test_inv, y_pred_inv)

            metrics['RMSE'].append(rmse)
            metrics['MAE'].append(mae)
            metrics['R²'].append(r2)
        except Exception as e:
            logger.error(f"Error en fold {fold} de LSTM: {e}")
            metrics['RMSE'].append(np.nan)
            metrics['MAE'].append(np.nan)
            metrics['R²'].append(np.nan)

    # Calcular métricas promedio
    avg_metrics = {k: np.nanmean(v) for k, v in metrics.items()}
    return avg_metrics
```

```python
# Función para realizar Cross-Validation para Prophet
def prophet_time_series_cv(model_prophet, initial, period, horizon):
    try:
        df_cv = cross_validation(model_prophet, initial=initial, period=period, \
                                 horizon=horizon, parallel="processes")
        df_p = performance_metrics(df_cv)
        return df_p.mean().to_dict()  # Devolver métricas promedio
    except Exception as e:
        logger.error(f"Error en Cross-Validation de Prophet: {e}")
        return {'rmse': np.nan, 'mae': np.nan, 'mape': np.nan, 'coverage': np.nan}

# Ajustar parámetros de Cross-Validation para Prophet
# Reducir los valores para adaptarse a la cantidad de datos (~90 días)
initial_prophet = '20 days'    # Tamaño del conjunto de entrenamiento inicial
period_prophet = '5 days'      # Frecuencia de las validaciones
horizon_prophet = '5 days'     # Tamaño del período de predicción

# Calcular Cross-Validation para ARIMA
metrics_arima_in_cv = arima_time_series_cv(y_train_in, order=arima_order, n_splits=3)
metrics_arima_out_cv = arima_time_series_cv(y_train_out, order=arima_order, n_splits=3)
metrics_arima_total_cv = arima_time_series_cv(y_train_total, order=arima_order, n_splits=3)
logger.info(f"Métricas Cross-Validation ARIMA InTraffic: {metrics_arima_in_cv}")
logger.info(f"Métricas Cross-Validation ARIMA OutTraffic: {metrics_arima_out_cv}")
logger.info(f"Métricas Cross-Validation ARIMA TotalTraffic: {metrics_arima_total_cv}")

# Calcular Cross-Validation para LSTM
metrics_lstm_in_cv = lstm_time_series_cv(X_train_in, y_train_in, scaler_in, time_step=time_step,\
                                         n_splits=3, epochs=10, batch_size=64)
metrics_lstm_out_cv = lstm_time_series_cv(X_train_out, y_train_out, scaler_out, time_step=time_step, \
                                          n_splits=3, epochs=10, batch_size=64)
metrics_lstm_total_cv = lstm_time_series_cv(X_train_total, y_train_total, scaler_total,\
                                            time_step=time_step, \n_splits=3, epochs=10, batch_size=64)
logger.info(f"Métricas Cross-Validation LSTM InTraffic: {metrics_lstm_in_cv}")
logger.info(f"Métricas Cross-Validation LSTM OutTraffic: {metrics_lstm_out_cv}")
logger.info(f"Métricas Cross-Validation LSTM TotalTraffic: {metrics_lstm_total_cv}")

# Calcular Cross-Validation para Prophet
metrics_prophet_in_cv = prophet_time_series_cv(model_prophet_in, initial=initial_prophet, \
                                               period=period_prophet, horizon=horizon_prophet) \
                                               if model_prophet_in else {}
metrics_prophet_out_cv = prophet_time_series_cv(model_prophet_out, initial=initial_prophet, \
                                                period=period_prophet, horizon=horizon_prophet) \
                                                if model_prophet_out else {}
metrics_prophet_total_cv = prophet_time_series_cv(model_prophet_total, initial=initial_prophet,\
                                                  period=period_prophet, horizon=horizon_prophet)\
                                                  if model_prophet_total else {}

logger.info(f"Métricas Cross-Validation Prophet InTraffic: {metrics_prophet_in_cv}")
logger.info(f"Métricas Cross-Validation Prophet OutTraffic: {metrics_prophet_out_cv}")
logger.info(f"Métricas Cross-Validation Prophet TotalTraffic: {metrics_prophet_total_cv}")

# Calcular métricas para cada modelo y tipo de tráfico utilizando el conjunto de validación
metrics_in_arima = calculate_metrics(y_val_in, forecast_arima_in, y_train_in, scaler_in,\
                                     cv_metrics=metrics_arima_in_cv)
metrics_in_lstm = calculate_metrics(y_val_in, test_predict_lstm_in.flatten(), y_train_in, scaler_in,\
                                    cv_metrics=metrics_lstm_in_cv) if test_predict_lstm_in.size else {}
metrics_in_prophet = calculate_metrics(y_val_in, forecast_prophet_in['yhat'].values, y_train_in,\
                                       scaler_in, cv_metrics=metrics_prophet_in_cv) \
                                       if not forecast_prophet_in.empty else {}
metrics_out_arima = calculate_metrics(y_val_out, forecast_arima_out, y_train_out, scaler_out,\
                                      cv_metrics=metrics_arima_out_cv)
metrics_out_lstm = calculate_metrics(y_val_out, test_predict_lstm_out.flatten(), y_train_out, scaler_out,\
                                     cv_metrics=metrics_lstm_out_cv) if test_predict_lstm_out.size else {}
metrics_out_prophet = calculate_metrics(y_val_out, forecast_prophet_out['yhat'].values, y_train_out, \
                                        scaler_out, cv_metrics=metrics_prophet_out_cv) \
                                        if not forecast_prophet_out.empty else {}
```

```python
metrics_total_arima = calculate_metrics(y_val_total, forecast_arima_total, y_train_total,\
                                        scaler_total, cv_metrics=metrics_arima_total_cv)
metrics_total_lstm = calculate_metrics(y_val_total, test_predict_lstm_total.flatten(), y_train_total,\
                                       scaler_total, cv_metrics=metrics_lstm_total_cv) \
                                       if test_predict_lstm_total.size else {}
metrics_total_prophet = calculate_metrics(y_val_total, forecast_prophet_total['yhat'].values,\
                                          y_train_total, scaler_total, cv_metrics=metrics_prophet_total_cv)\
                                           if not forecast_prophet_total.empty else {}

# Crear DataFrames para métricas incluyendo Cross-Validation
def create_metrics_df(metrics_arima, metrics_lstm, metrics_prophet):
    metrics = list(metrics_arima.keys())
    data = {
        'Métrica': metrics,
        'ARIMA': [metrics_arima.get(m, np.nan) for m in metrics],
        'LSTM': [metrics_lstm.get(m, np.nan) for m in metrics] if metrics_lstm else [np.nan]*len(metrics),
        'Prophet': [metrics_prophet.get(m, np.nan) for m in metrics] \
        if metrics_prophet else [np.nan]*len(metrics)
    }
    return pd.DataFrame(data)

metrics_df_in = create_metrics_df(metrics_in_arima, metrics_in_lstm, metrics_in_prophet)
metrics_df_out = create_metrics_df(metrics_out_arima, metrics_out_lstm, metrics_out_prophet)
metrics_df_total = create_metrics_df(metrics_total_arima, metrics_total_lstm, metrics_total_prophet)

logger.info("DataFrames de métricas creados exitosamente.")

# Función para predicción iterativa con LSTM
def predict_lstm(model, X_input, steps):
    """
    Realiza predicciones iterativas con un modelo LSTM.

    :param model: Modelo LSTM entrenado.
    :param X_input: Última ventana de datos de entrada.
    :param steps: Número de pasos a predecir.
    :return: Array de predicciones.
    """
    predictions = []
    current_input = X_input.copy()
    for _ in range(steps):
        pred = model.predict(current_input[np.newaxis, :, :])[0,0]
        predictions.append(pred)
        # Actualizar la entrada desplazando y añadiendo la nueva predicción
        # Solo 'avg' está disponible, así que añadimos el valor predicho
        new_input = np.append(current_input[1:, :, :], [[[pred]]], axis=0)
        current_input = new_input
    return np.array(predictions)

# Función para generar las gráficas y métricas
@app.callback(
    [
        DashOutput('intraffic-graph', 'figure'),
        DashOutput('outtraffic-graph', 'figure'),
        DashOutput('totaltraffic-graph', 'figure'),
        DashOutput('metrics-intraffic-graph', 'figure'),
        DashOutput('metrics-outtraffic-graph', 'figure'),
        DashOutput('metrics-totaltraffic-graph', 'figure'),
        DashOutput('additional-prediction-graph', 'figure'),
        DashOutput('prediction-target-graph', 'figure')  # Nuevo Output para la predicción hasta 24-11-2024
    ],
    [
        DashInput('prediction-horizon', 'value')  # Solo el dropdown como Input
    ]
)
```

```python
def update_graphs(prediction_horizon):
    try:
        # Definir el rango de fechas
        start_date = df_in['Time'].min().strftime('%Y-%m-%d %H:%M:%S')
        end_date = df_in['Time'].max().strftime('%Y-%m-%d %H:%M:%S')

        # Gráfico para InTraffic
        fig_in = go.Figure()
        # Datos de entrenamiento
        fig_in.add_trace(go.Scatter(
            x=df_in['Time'][:len(X_train_in)],
            y=scaler_in.inverse_transform(y_train_in.reshape(-1, 1)).flatten(),
            mode='lines',
            name='Entrenamiento'
        ))
        # Datos de validación
        fig_in.add_trace(go.Scatter(
            x=df_in['Time'][len(X_train_in):len(X_train_in) + len(y_val_in)],
            y=scaler_in.inverse_transform(y_val_in.reshape(-1, 1)).flatten(),
            mode='lines',
            name='Validación'
        ))
        # Datos de testeo
        fig_in.add_trace(go.Scatter(
            x=df_in['Time'][len(X_train_in) + len(y_val_in):len(X_train_in) + len(y_val_in) + len(y_test_in)],
            y=scaler_in.inverse_transform(y_test_in.reshape(-1, 1)).flatten(),
            mode='lines',
            name='Testeo'
        ))
        # Predicciones ARIMA
        if len(forecast_arima_in) == len(y_val_in):
            forecast_arima_in_inv = scaler_in.inverse_transform(forecast_arima_in.reshape(-1, 1)).flatten()
            fig_in.add_trace(go.Scatter(
                x=df_in['Time'][len(X_train_in):len(X_train_in) + len(y_val_in)],
                y=forecast_arima_in_inv,
                mode='lines',
                name='ARIMA'
            ))
        # Predicciones Prophet
        if not forecast_prophet_in.empty and 'yhat' in forecast_prophet_in.columns:
            forecast_prophet_in_inv = scaler_in.inverse_transform(forecast_prophet_in['yhat'].\
                                                                  values.reshape(-1,1)).flatten()
            fig_in.add_trace(go.Scatter(
                x=forecast_prophet_in['ds'],
                y=forecast_prophet_in_inv,
                mode='lines',
                name='Prophet'
            ))
        # Predicciones LSTM
        if test_predict_lstm_in.size == len(y_val_in):
            test_predict_lstm_in_inv = scaler_in.inverse_transform(test_predict_lstm_in).flatten()
            fig_in.add_trace(go.Scatter(
                x=df_in['Time'][len(X_train_in):len(X_train_in) + len(y_val_in)],
                y=test_predict_lstm_in_inv,
                mode='lines',
                name='LSTM'
            ))
        fig_in.update_layout(
            title='InTraffic: Predicciones vs Reales',
            xaxis_title='Tiempo',
            yaxis_title='Bits por Segundo',
            xaxis=dict(range=[start_date, end_date])
        )
```

```python
# Gráfico de Métricas para InTraffic
fig_metrics_in = px.bar(
    metrics_df_in,
    x='Métrica',
    y=['ARIMA', 'LSTM', 'Prophet'],
    barmode='group',
    title='Métricas de Evaluación InTraffic',
    labels={'value': 'Valor', 'variable': 'Modelo'}
)
# Gráfico para OutTraffic
fig_out = go.Figure()
# Datos de entrenamiento
fig_out.add_trace(go.Scatter(
    x=df_out['Time'][:len(X_train_out)],
    y=scaler_out.inverse_transform(y_train_out.reshape(-1, 1)).flatten(),
    mode='lines',
    name='Entrenamiento'
))
# Datos de validación
fig_out.add_trace(go.Scatter(
    x=df_out['Time'][len(X_train_out):len(X_train_out) + len(y_val_out)],
    y=scaler_out.inverse_transform(y_val_out.reshape(-1, 1)).flatten(),
    mode='lines',
    name='Validación'
))
# Datos de testeo
fig_out.add_trace(go.Scatter(
    x=df_out['Time'][len(X_train_out) + len(y_val_out):len(X_train_out) + len(y_val_out) + len(y_test_out)],
    y=scaler_out.inverse_transform(y_test_out.reshape(-1, 1)).flatten(),
    mode='lines',
    name='Testeo'
))
# Predicciones ARIMA
if len(forecast_arima_out) == len(y_val_out):
    forecast_arima_out_inv = scaler_out.inverse_transform(forecast_arima_out.reshape(-1, 1)).flatten()
    fig_out.add_trace(go.Scatter(
        x=df_out['Time'][len(X_train_out):len(X_train_out) + len(y_val_out)],
        y=forecast_arima_out_inv,
        mode='lines',
        name='ARIMA'
    ))
# Predicciones Prophet
if not forecast_prophet_out.empty and 'yhat' in forecast_prophet_out.columns:
    forecast_prophet_out_inv = scaler_out.inverse_transform(forecast_prophet_out['yhat'].\
                                                            values.reshape(-1,1)).flatten()
    fig_out.add_trace(go.Scatter(
        x=forecast_prophet_out['ds'],
        y=forecast_prophet_out_inv,
        mode='lines',
        name='Prophet'
    ))
# Predicciones LSTM
if test_predict_lstm_out.size == len(y_val_out):
    test_predict_lstm_out_inv = scaler_out.inverse_transform(test_predict_lstm_out).flatten()
    fig_out.add_trace(go.Scatter(
        x=df_out['Time'][len(X_train_out):len(X_train_out) + len(y_val_out)],
        y=test_predict_lstm_out_inv,
        mode='lines',
        name='LSTM'
    ))
fig_out.update_layout(
    title='OutTraffic: Predicciones vs Reales',
    xaxis_title='Tiempo',
    yaxis_title='Bits por Segundo',
    xaxis=dict(range=[start_date, end_date])
)
```

```python
# Gráfico de Métricas para OutTraffic
fig_metrics_out = px.bar(
    metrics_df_out,
    x='Métrica',
    y=['ARIMA', 'LSTM', 'Prophet'],
    barmode='group',
    title='Métricas de Evaluación OutTraffic',
    labels={'value': 'Valor', 'variable': 'Modelo'}
)
# Gráfico para TotalTraffic
fig_total = go.Figure()
# Datos de entrenamiento
fig_total.add_trace(go.Scatter(
    x=df_total['Time'][:len(X_train_total)],
    y=scaler_total.inverse_transform(y_train_total.reshape(-1, 1)).flatten(),
    mode='lines',
    name='Entrenamiento'
))
# Datos de validación
fig_total.add_trace(go.Scatter(
    x=df_total['Time'][len(X_train_total):len(X_train_total) + len(y_val_total)],
    y=scaler_total.inverse_transform(y_val_total.reshape(-1, 1)).flatten(),
    mode='lines',
    name='Validación'
))
# Datos de testeo
fig_total.add_trace(go.Scatter(
    x=df_total['Time'][len(X_train_total) + len(y_val_total):len(X_train_total) + \
                       len(y_val_total) + len(y_test_total)],
    y=scaler_total.inverse_transform(y_test_total.reshape(-1, 1)).flatten(),
    mode='lines',
    name='Testeo'
))
# Predicciones ARIMA
if len(forecast_arima_total) == len(y_val_total):
    forecast_arima_total_inv = scaler_total.inverse_transform(forecast_arima_total.reshape(-1, 1)).flatten()
    fig_total.add_trace(go.Scatter(
        x=df_total['Time'][len(X_train_total):len(X_train_total) + len(y_val_total)],
        y=forecast_arima_total_inv,
        mode='lines',
        name='ARIMA'
    ))
# Predicciones Prophet
if not forecast_prophet_total.empty and 'yhat' in forecast_prophet_total.columns:
    forecast_prophet_total_inv = scaler_total.inverse_transform(forecast_prophet_total['yhat'].\
                                                                values.reshape(-1,1)).flatten()
    fig_total.add_trace(go.Scatter(
        x=forecast_prophet_total['ds'],
        y=forecast_prophet_total_inv,
        mode='lines',
        name='Prophet'
    ))
# Predicciones LSTM
if test_predict_lstm_total.size == len(y_val_total):
    test_predict_lstm_total_inv = scaler_total.inverse_transform(test_predict_lstm_total).flatten()
    fig_total.add_trace(go.Scatter(
        x=df_total['Time'][len(X_train_total):len(X_train_total) + len(y_val_total)],
        y=test_predict_lstm_total_inv,
        mode='lines',
        name='LSTM'
    ))
fig_total.update_layout(
    title='TotalTraffic: Predicciones vs Reales',
    xaxis_title='Tiempo',
    yaxis_title='Bits por Segundo',
    xaxis=dict(range=[start_date, end_date])        )
```

```python
# Gráfico de Métricas para TotalTraffic
fig_metrics_total = px.bar(
    metrics_df_total,
    x='Métrica',
    y=['ARIMA', 'LSTM', 'Prophet'],
    barmode='group',
    title='Métricas de Evaluación TotalTraffic',
    labels={'value': 'Valor', 'variable': 'Modelo'}
)

# Nueva lógica para la predicción adicional
try:
    # Determinar el número de pasos basados en el horizonte seleccionado
    if prediction_horizon.endswith('D'):
        days = int(prediction_horizon[:-1])
        steps = days * 1440  # Asumiendo datos por minuto; 1440 minutos en un día
    elif prediction_horizon.endswith('W'):
        weeks = int(prediction_horizon[:-1])
        steps = weeks * 7 * 1440
    elif prediction_horizon.endswith('M'):
        months = int(prediction_horizon[:-1])
        steps = months * 30 * 1440  # Aproximación: 30 días por mes
    else:
        steps = 1440  # Valor por defecto: 1 día

    logger.info(f"Generando predicciones adicionales para {steps} pasos.")

    # Generar fechas futuras para la predicción
    last_timestamp = df_total['Time'].max()
    future_dates = pd.date_range(start=last_timestamp + timedelta(minutes=1), periods=steps, freq='min')

    # Predicción ARIMA
    forecast_arima_extra = train_arima(y_train_total, order=arima_order, steps=steps)
    if len(forecast_arima_extra) == steps:
        forecast_arima_extra_inv = scaler_total.inverse_transform(forecast_arima_extra.reshape(-1,1)).flatten()
        logger.info("ARIMA Predicción adicional TotalTraffic generada exitosamente.")
    else:
        forecast_arima_extra_inv = []
        logger.warning("La predicción ARIMA adicional no tiene la longitud esperada.")

    # Predicción LSTM (Iterativa)
    forecast_lstm_extra = predict_lstm(model_lstm_total, X_val_total[-1], steps)
    if len(forecast_lstm_extra) == steps:
        forecast_lstm_extra_inv = scaler_total.inverse_transform(forecast_lstm_extra.reshape(-1,1)).flatten()
        logger.info("LSTM Predicción adicional TotalTraffic generada exitosamente.")
    else:
        forecast_lstm_extra_inv = []
        logger.warning("La predicción LSTM adicional no tiene la longitud esperada.")

    # Predicción Prophet
    if model_prophet_total:
        # Crear DataFrame futuro sin zona horaria
        last_timestamp_naive = last_timestamp.tz_convert(None) if last_timestamp.tzinfo else last_timestamp
        future_prophet = model_prophet_total.make_future_dataframe(periods=steps, freq='min')
        forecast_prophet_extra = model_prophet_total.predict(future_prophet)
        forecast_prophet_extra = forecast_prophet_extra.tail(steps)
        if not forecast_prophet_extra.empty:
            forecast_prophet_extra_inv = scaler_total.inverse_transform(forecast_prophet_extra['yhat'].\
                                                                        values.reshape(-1,1)).flatten()
            logger.info("Prophet Predicción adicional TotalTraffic generada exitosamente.")
        else:
            forecast_prophet_extra_inv = []
            logger.warning("Prophet Predicción adicional TotalTraffic está vacía.")
    else:
        forecast_prophet_extra_inv = []
        logger.warning("Modelo Prophet TotalTraffic no está disponible para predicción adicional.")
```

```
    # Preparar la gráfica de predicción adicional
    fig_extra = go.Figure()
    if len(forecast_arima_extra_inv) == steps:
        fig_extra.add_trace(go.Scatter(
            x=future_dates,
            y=forecast_arima_extra_inv,
            mode='lines',
            name='ARIMA'
        ))
    else:
        logger.warning("La predicción ARIMA adicional no tiene la longitud esperada.")

    if len(forecast_prophet_extra_inv) == steps:
        fig_extra.add_trace(go.Scatter(
            x=forecast_prophet_extra['ds'],
            y=forecast_prophet_extra_inv,
            mode='lines',
            name='Prophet'
        ))
    else:
        logger.warning("La predicción Prophet adicional está vacía o no tiene la longitud esperada.")

    if len(forecast_lstm_extra_inv) == steps:
        fig_extra.add_trace(go.Scatter(
            x=future_dates,
            y=forecast_lstm_extra_inv,
            mode='lines',
            name='LSTM'
        ))
    else:
        logger.warning("La predicción LSTM adicional no tiene la longitud esperada.")

    fig_extra.update_layout(
        title=f'Predicción Adicional ({prediction_horizon}) - TotalTraffic',
        xaxis_title='Tiempo',
        yaxis_title='Bits por Segundo',
        xaxis=dict(range=[last_timestamp + timedelta(minutes=1), last_timestamp + timedelta(minutes=steps)])
    )
except Exception as e:
    logger.error(f"Error en la predicción adicional: {e}")
    fig_extra = go.Figure()
    fig_extra.update_layout(
        title='Error en la Predicción Adicional',
        xaxis_title='Tiempo',
        yaxis_title='Bits por Segundo'
    )

# Nueva lógica para la predicción hasta 24-11-2024
try:
    # Calcular la diferencia de tiempo en minutos hasta el 24-11-2024
    last_timestamp_target = df_total['Time'].max()
    target_date = pd.Timestamp('2024-11-24').tz_localize('America/Lima')
    steps_to_target = int((target_date - last_timestamp_target).total_seconds() / 60)
    if steps_to_target <= 0:
        logger.warning("La fecha objetivo ya ha pasado o coincide con la última fecha de los datos.")
        forecast_lstm_target_inv = []
        forecast_prophet_target_inv = []
        forecast_arima_target_inv = []
        future_dates_target = pd.DatetimeIndex([])
    else:
        # Predicción ARIMA hasta el 24-11-2024
        forecast_arima_target = train_arima(y_train_total, order=arima_order, steps=steps_to_target)
        if len(forecast_arima_target) == steps_to_target:
            forecast_arima_target_inv = scaler_total.inverse_transform(forecast_arima_target.\
                                                                       reshape(-1,1)).flatten()
            logger.info(f"ARIMA Predicción hasta 24-11-2024 generada exitosamente.")
```

```
    else:
        forecast_arima_target_inv = []
        logger.warning("La predicción ARIMA hasta 24-11-2024 no tiene la longitud esperada.")

    # Predicción LSTM hasta el 24-11-2024
    forecast_lstm_target = predict_lstm(model_lstm_total, X_val_total[-1], steps_to_target)
    if len(forecast_lstm_target) == steps_to_target:
        forecast_lstm_target_inv = scaler_total.inverse_transform(forecast_lstm_target.reshape(-1, 1)).flatten()
        logger.info(f"LSTM Predicción hasta 24-11-2024 generada exitosamente.")
    else:
        forecast_lstm_target_inv = []
        logger.warning(f"La predicción LSTM hasta 24-11-2024 no tiene la longitud esperada.")

    # Predicción Prophet hasta el 24-11-2024
    if model_prophet_total:
        # Crear DataFrame futuro sin zona horaria
        future_prophet_target = model_prophet_total.make_future_dataframe(periods=steps_to_target, freq='min')
        forecast_prophet_target = model_prophet_total.predict(future_prophet_target)
        forecast_prophet_target = forecast_prophet_target.tail(steps_to_target)
        if not forecast_prophet_target.empty:
            forecast_prophet_target_inv = scaler_total.inverse_transform(forecast_prophet_target['yhat'].\
                                                                         values.reshape(-1,1)).flatten()
            logger.info("Prophet Predicción hasta 24-11-2024 generada exitosamente.")
        else:
            forecast_prophet_target_inv = []
            logger.warning("Prophet Predicción hasta 24-11-2024 está vacía.")
    else:
        forecast_prophet_target_inv = []
        logger.warning("Modelo Prophet TotalTraffic no está disponible para predicción hasta 24-11-2024.")

    # Generar fechas futuras para la predicción hasta la fecha objetivo
    future_dates_target = pd.date_range(start=last_timestamp_target + timedelta(minutes=1), \
                                        periods=steps_to_target, freq='min')

# Preparar la gráfica de predicción hasta 24-11-2024
fig_target = go.Figure()
if len(forecast_arima_target_inv) == steps_to_target:
    fig_target.add_trace(go.Scatter(
        x=future_dates_target,
        y=forecast_arima_target_inv,
        mode='lines',
        name='ARIMA hasta 24-11-2024'
    ))
else:
    logger.warning("La predicción ARIMA hasta 24-11-2024 no tiene la longitud esperada.")

if len(forecast_prophet_target_inv) == steps_to_target:
    fig_target.add_trace(go.Scatter(
        x=forecast_prophet_target['ds'],
        y=forecast_prophet_target_inv,
        mode='lines',
        name='Prophet hasta 24-11-2024'
    ))
else:
    logger.warning("La predicción Prophet hasta 24-11-2024 está vacía o no tiene la longitud esperada.")

if len(forecast_lstm_target_inv) == steps_to_target:
    fig_target.add_trace(go.Scatter(
        x=future_dates_target,
        y=forecast_lstm_target_inv,
        mode='lines',
        name='LSTM hasta 24-11-2024'
    ))
```

```
        else:
            logger.warning("La predicción LSTM hasta 24-11-2024 no tiene la longitud esperada.")

        fig_target.update_layout(
            title='Predicción hasta 24-11-2024 - TotalTraffic',
            xaxis_title='Tiempo',
            yaxis_title='Bits por Segundo',
            xaxis=dict(range=[last_timestamp_target + timedelta(minutes=1), target_date])
        )
    except Exception as e:
        logger.error(f"Error en predicción LSTM hasta 24-11-2024: {e}")
        fig_target = go.Figure()
        fig_target.update_layout(
            title='Error en la Predicción hasta 24-11-2024',
            xaxis_title='Tiempo',
            yaxis_title='Bits por Segundo'
        )

    return fig_in, fig_out, fig_total, fig_metrics_in, fig_metrics_out, fig_metrics_total, \
    fig_extra, fig_target
except Exception as e:
    logger.error(f"Error en el callback de Dash: {e}")
    # Retornar figuras vacías en caso de error
    empty_fig = go.Figure()
    empty_fig.update_layout(
        title='Error al Generar la Gráfica',
        xaxis_title='Tiempo',
        yaxis_title='Bits por Segundo'
    )
    empty_metrics = px.bar(
        pd.DataFrame({'Métrica': [], 'ARIMA': [], 'LSTM': [], 'Prophet': []}),
        x='Métrica',
        y=['ARIMA', 'LSTM', 'Prophet'],
        barmode='group',
        title='Métricas de Evaluación No Disponibles',
        labels={'value': 'Valor', 'variable': 'Modelo'}
    )
    empty_pred = go.Figure()
    empty_pred.update_layout(
        title='Predicción Adicional No Disponible',
        xaxis_title='Tiempo',
        yaxis_title='Bits por Segundo'
    )
    empty_target = go.Figure()
    empty_target.update_layout(
        title='Predicción hasta 24-11-2024 No Disponible',
        xaxis_title='Tiempo',
        yaxis_title='Bits por Segundo'
    )
    return (
        empty_fig,
        empty_fig,
        empty_fig,
        empty_metrics,
        empty_metrics,
        empty_metrics,
        empty_pred,
        empty_target
    )
```

```python
# Configurar Layout del Dashboard con métricas adicionales y predicción
app.layout = html.Div([
    html.H1("Predicción de Tráfico de Red con LSTM y Prophet"),
    dcc.Tabs([
        dcc.Tab(label='InTraffic', children=[
            dcc.Graph(id='intraffic-graph'),
            dcc.Graph(id='metrics-intraffic-graph')
        ]),
        dcc.Tab(label='OutTraffic', children=[
            dcc.Graph(id='outtraffic-graph'),
            dcc.Graph(id='metrics-outtraffic-graph')
        ]),
        dcc.Tab(label='TotalTraffic', children=[
            dcc.Graph(id='totaltraffic-graph'),
            dcc.Graph(id='metrics-totaltraffic-graph')
        ]),
        dcc.Tab(label='Predicción Adicional', children=[
            html.Div([
                html.Label("Cantidad de Tiempo para Predicción:"),
                dcc.Dropdown(
                    id='prediction-horizon',
                    options=[
                        {'label': '1 Día', 'value': '1D'},
                        {'label': '7 Días', 'value': '7D'},
                        {'label': '30 Días (1 Mes)', 'value': '30D'},
                        {'label': '60 Días (2 Meses)', 'value': '60D'},  # Nueva opción
                        {'label': '90 Días (3 Meses)', 'value': '90D'},  # Nueva opción
                    ],
                    value='30D',  # Valor por defecto
                    clearable=False
                ),
            ], style={'width': '300px', 'margin': '20px'}),
            dcc.Graph(id='additional-prediction-graph')
        ]),
        dcc.Tab(label='Predicción hasta 24-11-2024', children=[
            dcc.Graph(id='prediction-target-graph')
        ])
    ])
])

# Ejecutar la aplicación
if __name__ == '__main__':
    app.run_server(debug=True, host='127.0.0.17', port=8074)  # Cambiar puerto a 8053 si es necesario
```