

Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Inteligencia Artificial

Predicción de tráfico mediante aprendizaje
profundo y Transformers

Trabajo fin de máster presentado por:	Jon Inazio Sánchez Martínez
Tipo de trabajo:	Desarrollo
Director:	Omar Velázquez López
Fecha:	26 de junio del 2025

Resumen

En este Trabajo Fin de Máster se aborda el desarrollo de un sistema avanzado de predicción del flujo de tráfico en la provincia de Bizkaia mediante técnicas de aprendizaje profundo basadas en arquitecturas transformer. El dataset ha sido elaborado integrando múltiples fuentes de datos abiertos oficiales del Gobierno Vasco, incluyendo aforos de tráfico y variables meteorológicas proporcionadas por Euskalmet, lo que ha permitido enriquecer la modelización con información contextual relevante.

El modelo propuesto, fundamentado en la arquitectura Trafficformer, se ha entrenado y evaluado exhaustivamente frente a un modelo base MLP, demostrando mejoras sustanciales en la capacidad predictiva. En los experimentos realizados sobre distintos conjuntos de sensores, Trafficformer ha logrado reducir el error absoluto medio (MAE) en test hasta un 25–30 % respecto al MLP, alcanzando valores de MAE inferiores a 0.003 vehículos normalizados y un R^2 superior a 0.96 en los mejores escenarios, frente a R^2 de 0.87–0.90 para el MLP. Además, el modelo ha mostrado mayor robustez frente a la variabilidad meteorológica y la heterogeneidad espacial de la red viaria.

Entre las principales contribuciones destacan la integración efectiva de datos heterogéneos, la aplicación y adaptación de técnicas de deep learning de última generación al contexto de la predicción de tráfico real y la publicación de un pipeline reproducible. Se discuten también las limitaciones encontradas y se proponen líneas de trabajo futuro orientadas a la incorporación de nuevos tipos de datos y la extensión a entornos urbanos más complejos.

Palabras clave: Predicción del tráfico, Aprendizaje profundo, Transformers, Datos abiertos, Comunidad Autónoma del País Vasco, Bizkaia

Abstract

This Master's Thesis presents the development of an advanced traffic flow forecasting system in the province of Bizkaia using deep learning techniques based on transformer architectures. The dataset was constructed by integrating multiple official Open Data sources from the Basque Government, including traffic count data and meteorological variables from Euskalmet, thus enriching the modeling process with relevant contextual features.

The proposed model, based on the Trafficformer architecture, was thoroughly trained and benchmarked against a baseline MLP model, achieving substantial improvements in predictive capability. Across various sensor sets, Trafficformer reduced the mean absolute error (MAE) on the test set by 25–30% compared to MLP, reaching MAE values below 0.003 (normalized vehicles) and test R^2 scores above 0.96 in the best cases, versus R^2 of 0.87–0.90 for the MLP. The model also demonstrated increased robustness to meteorological variability and the spatial heterogeneity of the road network.

Key contributions include the effective integration of heterogeneous data, the adaptation of state-of-the-art deep learning techniques to real-world traffic forecasting, and the publication of a reproducible pipeline. The limitations encountered are discussed, and future work is proposed to incorporate new data sources and extend the model to more complex urban scenarios.

Keywords: Traffic forecasting, Deep learning, Transformers, Open data, Autonomous Community of the Basque Country, Bizkaia

Índice de contenidos

Índice de figuras	v
Índice de tablas	vii
1 Introducción	1
1.1 Motivación del trabajo	1
1.2 Planteamiento del problema	1
2 Contexto y estado del arte	4
2.1 Técnicas existentes para la predicción del tráfico	4
2.2 Ventajas del uso de Transformers frente a otras arquitecturas	13
3 Objetivos y metodología de trabajo	15
3.1 Objetivos del proyecto	15
3.2 Infraestructura y tecnologías empleadas	16
3.3 Metodología de trabajo	18
4 Construcción del dataset y arquitectura del sistema	21
4.1 Fuentes de datos y análisis de disponibilidad	21
4.2 Recolección, arquitectura y patrones de diseño empleados	28
4.3 Decisiones de construcción del dataset	31
5 Desarrollo experimental, comparación de modelos y resultados	41
5.1 Planteamiento experimental y justificación	41
5.2 Entorno de desarrollo y reproducibilidad	46
5.3 Preparación de datos y pipeline	50
5.4 Preparación y tratamiento del dataset	51
5.5 Descripción de los modelos	55
5.6 Diseño experimental y combinaciones	60
5.7 Proceso de entrenamiento y selección de mejores modelos	62
6 Evaluación, comparación de modelos y resultados	66
6.1 Resultados experimentales	66
6.2 Discusión y análisis crítico	74
6.3 Análisis avanzado de resultados por fuente de datos	79
6.4 Limitaciones y validez de los experimentos	81
7 Conclusiones y trabajo futuro	83

7.1 Conclusiones	83
7.2 Trabajo Futuro	84
Acrónimos y Abreviaturas	85
Bibliografía	87
Anexo A – Descripción de sensores meteorológicos	92
Anexo B – Código fuente del generador de snapshots	94
Anexo C – Plantilla de infraestructura para AWS	100
Anexo D – Código fuente de la arquitectura Trafficformer	106
Anexo E – Listado detallado de combinaciones por modelo	118
Anexo F – Resultados detallados de los 120 experimentos	121
Anexo G – Análisis avanzado por fuente de datos	125

Índice de figuras

1	Red viaria de la CAPV	2
2	Arquitectura general del modelo Trafficformer Chang et al. (2025)	19
3	Ejemplo de error a la hora de consultar el API de Euskalmet.	23
4	Modelo de clases en UML de las entidades principales del sistema.	24
5	Diagrama de secuencia de integración y persistencia en MobilitySnapshot. .	36
6	Ubicación de aforos de la fuente Gobierno Vasco (source_id = 1)	42
7	Ubicación de aforos de la fuente Diputación Foral de Bizkaia (source_id = 2) .	43
8	Ubicación de aforos de la fuente Ayuntamiento de Bilbao (source_id = 5) . .	43
9	Captura del fichero pyproject.toml	47
10	Captura de Wandb	48
11	Esquema lógico del flujo de capas en el modelo MLP.	56
12	Aplicación de la máscara espacial en el mecanismo de atención de Trafficformer.	58
13	Esquema lógico del flujo de capas del modelo Trafficformer.	59
14	Curvas de evolución de pérdida para entrenamiento y validación, con indicación de la mejor época.	65
15	Curvas de entrenamiento para el modelo MLP con sourceld 1.	68
16	Curvas de entrenamiento para el modelo Trafficformer con sourceld 1. . .	69
17	Curvas de entrenamiento para el modelo MLP con sourceld 2.	70
18	Curvas de entrenamiento para el modelo Trafficformer con sourceld 2. . .	71
19	Curvas de entrenamiento para el modelo MLP con sourceld 5.	72
20	Curvas de entrenamiento para el modelo Trafficformer con sourceld 5. . .	73
21	Distribución espacial de sensores para Sourceld 1	75
22	Distribución espacial de sensores para Sourceld 2	76
23	Distribución espacial de sensores para Sourceld 5	77
24	Gráfico de dispersión (predicción vs. valores reales) para Sourceld 1. . . .	125
25	Histograma del error absoluto (residual) para Sourceld 1.	126
26	Boxplot del error absoluto por sensor para todos los sensores del Sourceld 1. .	126
27	Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 1.	126
28	Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 1.	127
29	Mapa espacial de errores (MAE) para sensores del Sourceld 1.	128
30	Gráfico de dispersión (predicción vs. valores reales) para Sourceld 2. . . .	129
31	Histograma del error absoluto (residual) para Sourceld 2.	129
32	Boxplot del error absoluto por sensor para todos los sensores del Sourceld 2. .	130

33	Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 2.	130
34	Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 2.	131
35	Mapa espacial de errores (MAE) para sensores del Sourceld 2.	132
36	Gráfico de dispersión (predicción vs. valores reales) para Sourceld 5.	133
37	Histograma del error absoluto (residual) para Sourceld 5.	133
38	Boxplot del error absoluto por sensor para todos los sensores del Sourceld 5.	134
39	Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 5.	134
40	Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 5.	135
41	Mapa espacial de errores (MAE) para sensores del Sourceld 5.	136

Índice de tablas

1	Comparativa entre arquitecturas en tareas de predicción del tráfico	14
3	Cobertura de datos por fuente y tipo.	22
5	Frecuencia de incidencias por tipo original	31
7	Agrupación final de incidencias para el modelo predictivo	32
9	Selección de sensores meteorológicos y su relevancia para la predicción de tráfico	34
11	Comparativa entre intervalos temporales posibles para el resampling	39
13	Comparación conceptual entre los modelos MLP y Trafficformer	45
15	Configuraciones experimentales evaluadas	61
17	Resultados de los mejores modelos por combinación sourceId–arquitectura .	66
20	Combinaciones evaluadas para el modelo MLP (por cada sourceId)	118
22	Combinaciones evaluadas para el modelo Trafficformer (por cada sourceId) .	120
24	Métricas principales de los 120 experimentos realizados (validación y test). . .	121

1 Introducción

1.1 Motivación del trabajo

La movilidad urbana eficiente constituye uno de los principales desafíos para las ciudades contemporáneas, especialmente en un contexto de creciente demanda de sostenibilidad, aumento del parque automovilístico y limitaciones estructurales de las infraestructuras viarias existentes. Una gestión adecuada del tráfico no solo repercute positivamente en la calidad de vida de la ciudadanía, sino que también tiene un impacto directo en la reducción del consumo energético, la mejora de la seguridad vial y la disminución de la contaminación atmosférica.

En este escenario, la predicción del tráfico emerge como una herramienta clave para anticiparse a situaciones de congestión, disruptores u otros eventos que puedan comprometer la fluidez de la circulación. La inteligencia Artificial y, en particular, del Aprendizaje Profundo, ha impulsado el desarrollo de modelos que integran datos en tiempo real y capturan patrones complejos mediante el uso de arquitecturas sofisticadas.

El creciente interés por aplicar estos avances en entornos reales de alta densidad urbana motiva la realización del presente trabajo, que se centra en el desarrollo de un modelo de predicción de tráfico con enfoque local en la provincia de Bizkaia, uno de los territorios más dinámicos y problemáticos desde el punto de vista de la movilidad en el norte de España.

1.2 Planteamiento del problema

La provincia de Bizkaia presenta un conjunto de particularidades que complican la gestión efectiva del tráfico. Su orografía abrupta limita la expansión de nuevas vías y condiciona la distribución del tráfico en corredores específicos, donde la saturación es frecuente. A esto se suma la alta densidad poblacional, especialmente en el área metropolitana de Bilbao, y una variabilidad meteorológica significativa, que puede afectar las condiciones de circulación de forma impredecible.

En la figura 1 se aprecia la complejidad de la red viaria en la CAPV. En diversas zonas como los accesos a Bilbao, los túneles del Kadagua o los enlaces con la A-8, se producen episodios de congestión recurrentes, especialmente en horas punta y durante condiciones meteorológicas adversas. Aunque Bizkaia dispone de una infraestructura ITS notable (sensores, cámaras,

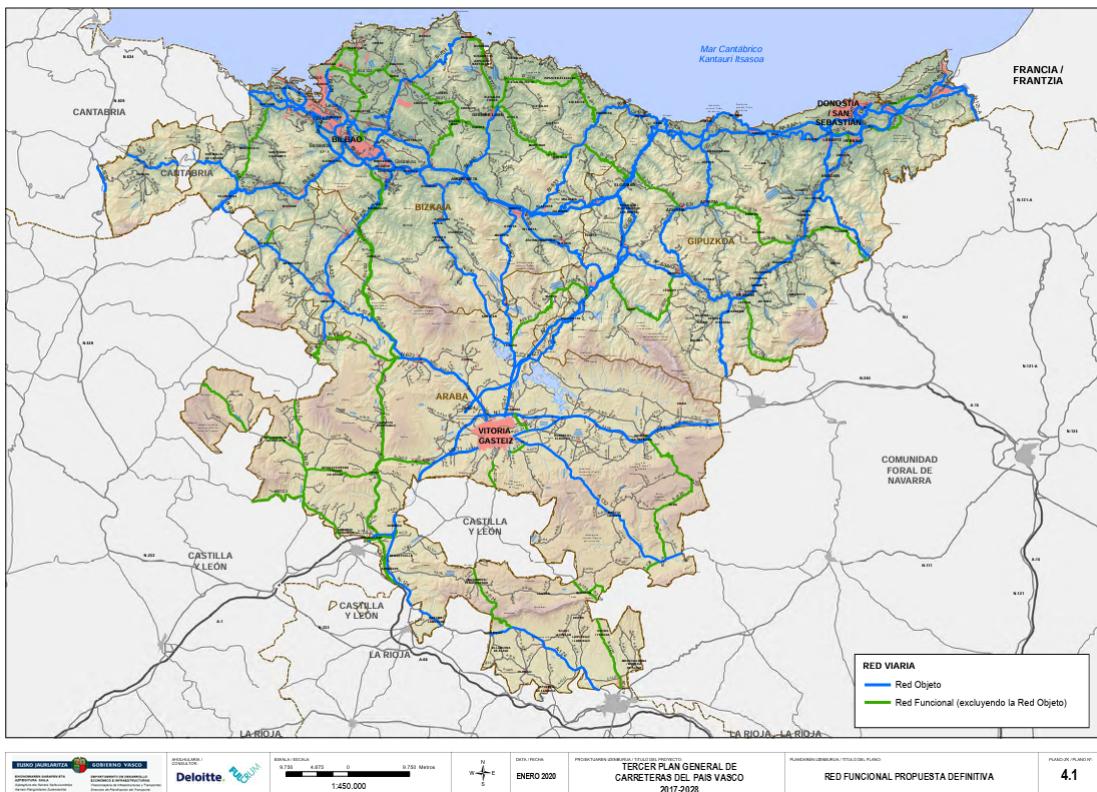


Figura 1: Red viaria de la CAPV, extraído de Gobierno Vasco, Eusko Jaurlaritza (2020)

estaciones meteorológicas), no se cuenta aún con herramientas predictivas suficientemente precisas que permitan anticipar estos episodios y facilitar la toma de decisiones en tiempo real.

La complejidad inherente a los datos de tráfico, caracterizados por correlaciones espacio-temporales, no linealidades y gran volumen, exige modelos capaces de capturar dichas relaciones con eficacia. Las aproximaciones tradicionales, basadas en regresión o aprendizaje automático clásico, resultan insuficientes en este contexto. Se necesita un enfoque moderno, capaz de integrar múltiples fuentes de datos y aprender representaciones complejas.

En respuesta a los retos mencionados, este trabajo propone el diseño, desarrollo y evaluación de un modelo de predicción del tráfico en Bizkaia basado en redes neuronales con arquitectura Transformer. Esta arquitectura ha demostrado un rendimiento sobresaliente en tareas de modelado secuencial y captura de dependencias a largo plazo, gracias a su mecanismo de atención, lo que la convierte en una candidata idónea para abordar la predicción de tráfico con alta precisión.

El modelo se alimentará de datos abiertos sobre tráfico y meteorología, publicados por orga-

nismos como Open Data Euskadi y Euskalmet, lo que garantiza la transparencia, replicabilidad y aplicabilidad del sistema propuesto. A través de un enfoque empírico, se evaluará la eficacia del modelo frente a enfoques tradicionales, utilizando métricas estándar y se estudiará su potencial para integrarse en sistemas actuales de gestión del tráfico.

Este documento se estructura de la siguiente manera: el capítulo 2 presenta una revisión del estado del arte, analizando los principales trabajos relacionados y su aplicabilidad al contexto de Bizkaia; el capítulo 3 define los objetivos del trabajo y la metodología seguida; los capítulos siguientes abordarán el desarrollo del proyecto y conclusiones finales, aunque esto es un trabajo por determinar.

2 Contexto y estado del arte

En los últimos años, la predicción del tráfico se ha convertido en un campo de investigación clave dentro de los Sistemas Inteligentes de Transporte, impulsado por la disponibilidad creciente de datos en tiempo real y los avances en el aprendizaje automático. El objetivo principal es anticipar las condiciones del tráfico con suficiente precisión como para facilitar la toma de decisiones, tanto por parte de los operadores como de los usuarios de la red vial.

La problemática de la predicción del tráfico en entornos urbanos y regionales como Bizkaia presenta una elevada complejidad, debido a la naturaleza altamente dinámica y no lineal del flujo vehicular, así como por la influencia de factores exógenos como el clima, los eventos especiales o los accidentes. Esta complejidad ha motivado el desarrollo de una gran variedad de enfoques, desde modelos estadísticos clásicos hasta sofisticadas arquitecturas de aprendizaje profundo.

Este capítulo tiene como objetivo ofrecer un panorama de las principales técnicas, modelos y tecnologías utilizadas en la predicción del tráfico. A partir de una revisión sistemática de la literatura científica más relevante, se presentarán los enfoques predominantes y se analizará su aplicabilidad al caso concreto de Bizkaia. Finalmente, se establecerá un marco comparativo que servirá como punto de partida para justificar la solución propuesta en este trabajo.

2.1 Técnicas existentes para la predicción del tráfico

La literatura especializada distingue entre dos grandes familias de métodos para la predicción del tráfico: los modelos basados en estadística y los modelos basados en aprendizaje automático, con especial atención a los métodos de aprendizaje profundo. A continuación, se presenta una clasificación preliminar de las principales técnicas utilizadas:

- **Modelos estadísticos:** AutoRegressive Integrated Moving Average, Kalman Filter, regresiones lineales.
- **Modelos clásicos de machine learning:** Support Vector Regression, Random Forests, K-Nearest Neighbors.
- **Redes neuronales profundas:**

- Recurrent Neural Networks o Redes Neuronales Recurrentes, Long Short-Term Memory y Gated Recurrent Units.
- Convolutional Neural Networks o Redes Neuronales Convolucionales, para capturar relaciones espaciales.
- Graph Neural Networks, incluyendo Graph Convolutional Networks y Graph Attention Networks, adaptadas a redes viarias.
- Modelos Transformer.

En las siguientes secciones, se revisarán con más detalle los fundamentos y aplicaciones de estas técnicas, poniendo el foco en los trabajos más relevantes que hayan utilizado estos enfoques en entornos comparables al del presente proyecto.

Modelos estadísticos tradicionales

Los modelos estadísticos tradicionales han sido fundamentales en la predicción del tráfico, especialmente en contextos donde se dispone de datos históricos limitados o se requiere una interpretación sencilla de los resultados. Estos modelos, incluyendo ARIMA, filtros de Kalman y regresiones lineales, permiten capturar patrones temporales y tendencias en los datos de tráfico, ofreciendo una base sólida para el desarrollo de sistemas de transporte inteligentes.

Los modelos AutoRegressive Integrated Moving Average son herramientas estadísticas utilizadas para analizar y predecir series temporales. En el ámbito del tráfico, han demostrado ser eficaces para prever flujos vehiculares a corto plazo, especialmente en situaciones con patrones estacionales o tendencias lineales.

Por ejemplo, Katambire et al. (2023) aplicaron modelos ARIMA y LSTM para predecir el flujo de tráfico en la intersección de Muhima, Kigali, concluyendo que la combinación de ambos modelos mejora la precisión de las predicciones.

Asimismo, Kumar & Vanajakshi (2015) propusieron un esquema de predicción utilizando el modelo Seasonal AutoRegressive Integrated Moving Average para prever el flujo de tráfico a corto plazo con datos limitados, demostrando que es posible obtener predicciones precisas utilizando solo tres días de datos históricos.

El filtro de Kalman es un algoritmo recursivo que estima el estado de un sistema dinámico a partir de una serie de mediciones observadas, que contienen ruido y otras inexactitudes.

En la predicción del tráfico, se utiliza para estimar y prever el flujo vehicular en tiempo real, adaptándose a cambios abruptos y condiciones variables.

Momin et al. (2023) emplearon el filtro de Kalman para predecir el flujo de tráfico a corto plazo en una carretera urbana de Dhaka, Bangladesh. El modelo logró un Mean Absolute Percentage Error o Error Porcentual Absoluto Medio del 14.62 %, indicando una precisión aceptable para aplicaciones prácticas.

La regresión lineal es una técnica estadística que modela la relación entre una variable dependiente y una o más variables independientes. En el contexto del tráfico, se ha utilizado para prever velocidades y flujos vehiculares basándose en variables como el tiempo, la densidad y la ocupación de la vía.

Por ejemplo, Liu et al. (2020) desarrollaron un modelo de predicción del tiempo de congestión del tráfico utilizando análisis de regresión múltiple y análisis de supervivencia. El estudio demostró que el modelo de regresión lineal múltiple puede predecir con precisión el tiempo de congestión del tráfico, con un grado de ajuste entre el valor predicho y el valor real superior a 0.96. Este enfoque permitió identificar las características de distribución y duración de la congestión, proporcionando una base sólida para la predicción del tráfico en entornos urbanos.

Sin embargo, estudios recientes han señalado limitaciones en la regresión lineal para capturar relaciones no lineales complejas en los datos de tráfico. Por ejemplo, en un análisis exhaustivo, se comparó el rendimiento de la regresión lineal con modelos más avanzados como Random Forests y XGBoost, encontrando que la regresión lineal presenta un ajuste deficiente y errores significativos en la predicción de velocidades de tráfico.

Modelos clásicos de Machine Learning

Los modelos clásicos de machine learning son métodos estadísticos avanzados capaces de abordar problemas complejos y no lineales. A continuación se describen brevemente tres técnicas destacadas: Support Vector Regression, Random Forests y K-Nearest Neighbors, incluyendo ejemplos recientes de aplicaciones en la predicción del tráfico.

En cuanto a Support Vector Regression (SVR), es una técnica basada en Support Vector Machines (SVM) que utiliza una función kernel para transformar el espacio de entrada original a uno de mayor dimensión, permitiendo modelar relaciones no lineales. El objetivo del SVR es iden-

tificar una función que tenga, como máximo, un error preestablecido (denominado *margen*) respecto a los datos reales.

En el contexto de predicción de tráfico, SVR se ha mostrado eficaz debido a su robustez ante ruido y capacidad de generalización con muestras pequeñas. Por ejemplo, un estudio reciente aplicó SVR para predecir el volumen de tráfico a corto plazo utilizando datos de flujo vehicular recolectados en áreas urbanas, mostrando una precisión significativa en comparación con métodos LSTM de Omar et al. (2024).

Random Forests (RF) es un algoritmo basado en árboles de decisión, que genera múltiples árboles de forma independiente, utilizando subconjuntos aleatorios de datos de entrenamiento (bagging) y variables aleatorias. La predicción final se obtiene por consenso, promediando las predicciones individuales de cada árbol, lo que reduce considerablemente el riesgo de sobreajuste.

En el ámbito del tráfico, RF es capaz de manejar grandes volúmenes de datos y capturar relaciones no lineales y complejas. Un ejemplo de aplicación es el estudio realizado por Wu (2024). El estudio tiene como objetivo predecir el flujo de tráfico a corto plazo, considerando patrones espaciales y temporales. Tras el preprocesamiento de datos con el Transformador Cuantil y la exploración de la correlación del flujo de tráfico, se identificaron los hiperparámetros óptimos del modelo mediante la búsqueda de cuadrícula de validación cruzada. El modelo RF demostró el mejor rendimiento, alcanzando una alta precisión en la predicción del flujo de tráfico.

K-Nearest Neighbors (KNN) es uno de los algoritmos más sencillos dentro de los métodos de aprendizaje supervisado. Este método predice el valor de una observación nueva en función de los valores de las K observaciones más cercanas del conjunto de entrenamiento. La cercanía se determina generalmente mediante una métrica de distancia, siendo la distancia euclíadiana la más comúnmente utilizada.

A pesar de su simplicidad, KNN es muy eficaz en la predicción de tráfico cuando los patrones de flujo muestran alta dependencia espacial y temporal. Recientemente, Aditya et al. (2020) emplearon KNN para la predicción del tráfico en Bandung, Indonesia, empleando este método e integrado con la aplicación de simulación de movilidad urbana SUMO. El estudio buscó mitigar la congestión de tráfico anual en la ciudad, particularmente en períodos vacacionales. Utilizaron datos históricos de tráfico de Jl. Riau Bandung para predecir el nivel de congestión. La evaluación del rendimiento del método, usando una división de datos para entrenamiento

y prueba, demostró una precisión muy alta con diferentes valores de 'k' vecinos considerados.

Aprendizaje profundo

El *Deep Learning* o aprendizaje profundo es un área de *Machine Learning* que utiliza redes neuronales artificiales de múltiples capas para modelar patrones complejos en los datos. Tras varias décadas de relativo estancamiento debido a las limitaciones computacionales y a las críticas vertidas por Minsky y Papert en los años 60 en el libro *Perceptrons: An Introduction to Computational Geometry* Minsky & Papert (1969), las redes neuronales experimentaron un resurgimiento a partir de la década de los 2000, impulsado por el incremento exponencial de la capacidad de cómputo, la disponibilidad de grandes volúmenes de datos y el avance en algoritmos de entrenamiento. Este renacimiento del interés en las redes profundas se consolidó con el trabajo de Hinton & Salakhutdinov (2006), donde se introdujo una técnica de preentrenamiento capa por capa utilizando autoencoders y máquinas de Boltzmann restringidas para facilitar el entrenamiento de redes neuronales profundas.

Sin embargo, fue en 2012 cuando el aprendizaje profundo irrumpió definitivamente en la comunidad científica, con la publicación de AlexNet, una red convolucional profunda que ganó con gran margen la competición ImageNet Large Scale Visual Recognition Challenge (ILSVRC). En este trabajo, Krizhevsky, Sutskever y Hinton demostraron que las redes profundas entrenadas con Graphics Processing Unit podían superar ampliamente los métodos tradicionales en tareas de visión por computador Krizhevsky et al. (2012). Este hito marcó el inicio de una nueva era para el aprendizaje profundo, consolidándolo como una de las herramientas más poderosas dentro del campo de la inteligencia artificial.

Actualmente, el aprendizaje profundo se caracteriza por la capacidad de representar funciones no lineales muy complejas gracias a estructuras como las redes neuronales profundas de tipo Multi Layer Perceptron o Perceptrones Multi Capa. Estas redes constan de múltiples capas ocultas, donde cada capa procesa información progresivamente más abstracta y permite descubrir patrones intrínsecos en grandes volúmenes de datos.

Redes neuronales recurrentes

Dentro del aprendizaje profundo, una clase especial de redes neuronales, conocidas como Recurrent Neural Networks o Redes Neuronales Recurrentes (RNN), ha demostrado ser particu-

larmente efectiva para modelar secuencias temporales. Las RNN están diseñadas para capturar dependencias temporales a largo plazo gracias a su estructura recurrente, que permite que la salida de una etapa de la red influya en etapas posteriores.

Las variantes más importantes y populares de las RNN son las redes Long Short-Term Memory y Gated Recurrent Units. Las LSTM fueron propuestas por Hochreiter & Schmidhuber (1997), y están especialmente diseñadas para manejar el problema del desvanecimiento del gradiente mediante el uso de puertas que controlan la información almacenada en la memoria de la red. Por otro lado, las redes GRU, introducidas por Cho et al. (2014), simplifican el modelo LSTM al combinar algunas de sus puertas, proporcionando un rendimiento similar con menos parámetros y una complejidad computacional reducida.

Diversos estudios han aplicado exitosamente las redes neuronales recurrentes a problemas específicos de predicción de tráfico, destacando las arquitecturas LSTM y GRU.

Un ejemplo notable es el trabajo realizado por Zhao et al. (2017), en el que utilizaron una red neuronal LSTM para la predicción de flujo de tráfico a corto plazo. En su investigación, entrenaron un modelo con datos históricos de volumen vehicular recolectados en sensores, alcanzando un Mean Relative Error o Error Medio Relativo (MRE) de tan solo un 6,41 %, demostrando así la efectividad del enfoque LSTM para capturar patrones complejos en series temporales del tráfico.

En cuanto a la aplicación de redes GRU, cabe destacar el estudio llevado a cabo por Ma et al. (2023), quienes diseñaron un algoritmo híbrido basado en la combinación de CNN y GRU para predecir la velocidad del tráfico. Este modelo obtuvo una media del MAPE de aproximadamente 8,60 %, reflejando una considerable precisión en la predicción del flujo vehicular al integrar tanto características espaciales como temporales del tráfico.

Ambos estudios evidencian cómo el uso de redes neuronales recurrentes permite modelar con alta precisión las dependencias temporales complejas inherentes al tráfico vehicular, posicionándolas como una alternativa destacada frente a métodos clásicos o más tradicionales. Estos modelos son especialmente útiles en contextos de predicción a corto plazo, donde la precisión y la velocidad de respuesta son críticas para una gestión eficiente del tráfico y la toma de decisiones en tiempo real.

Graph Neural Networks

Las Graph Neural Networks surgen de la necesidad de extender las capacidades del aprendizaje profundo a datos no euclidianos, como los grafos, que representan relaciones complejas entre entidades. A diferencia de las redes neuronales tradicionales, que operan sobre datos estructurados en rejillas (como imágenes o secuencias), las GNN están diseñadas para trabajar directamente con la estructura de los grafos, tal y como se explica por Scarselli et al. (2009), permitiendo capturar dependencias tanto locales como globales entre nodos.

Las GNN se basan en el principio de *message passing*, donde cada nodo actualiza su representación en función de sus vecinos. Entre las arquitecturas más destacadas se encuentran:

- **Graph Convolutional Networks:** Fueron introducidas en Kipf & Welling (2017). Estas redes generalizan las convoluciones a grafos, permitiendo una agregación eficiente de la información de los vecinos.
- **Graph Attention Networks:** Incorporan mecanismos de atención (como se verá más adelante) para ponderar la importancia de cada vecino en la actualización del nodo central. Fueron introducidas en Veličković et al. (2018).
- **Gated Graph Neural Networks:** Utilizan mecanismos de puertas, similares a las LSTM, para controlar el flujo de información entre nodos.

La predicción del flujo de tráfico es un desafío clave en los sistemas de transporte inteligentes, donde es esencial anticipar las condiciones del tráfico para optimizar la movilidad urbana. Las GNN son particularmente adecuadas para esta tarea debido a que las redes de carreteras pueden modelarse naturalmente como grafos, donde los nodos representan intersecciones o sensores, y las aristas representan las conexiones viales.

Un estudio destacado en este ámbito es *Improving Traffic Density Forecasting in Intelligent Transportation Systems Using Gated Graph Neural Networks*, de Khan et al. (2023). En este trabajo, los autores comparan diferentes arquitecturas de ? para la predicción de la densidad del tráfico, incluyendo GCN, GraphSAGE y GGNN. Los resultados muestran que las acrshort superan a las demás arquitecturas en términos de precisión, con un RMSE de 9.15 y un MAE de 7.1, destacando su capacidad para capturar dinámicamente las dependencias espaciales y temporales en los datos de tráfico.

Otro estudio relevante es *TrafficStream: A Streaming Traffic Flow Forecasting Framework Based on Graph Neural Networks and Continual Learning* de Chen et al. (2021). Este trabajo propone un marco de predicción de flujo de tráfico en tiempo real que combina GNN con aprendizaje continuo, permitiendo adaptarse a cambios en la red de tráfico y patrones de flujo a lo largo del tiempo. El modelo utiliza estrategias como la reactivación de datos históricos y el suavizado de parámetros para mantener la precisión de las predicciones en entornos dinámicos.

Redes Neuronales con Transformers

El avance hacia modelos más potentes y versátiles dentro del aprendizaje profundo encontró un punto de inflexión crucial en 2017 con la publicación del influyente artículo *Attention is all you need* por Vaswani et al. (2017). Esta publicación revolucionó el campo del aprendizaje automático al introducir la arquitectura Transformer, una propuesta que eliminaba por completo el uso de estructuras recurrentes como RNN o LSTM, en favor de un novedoso mecanismo de atención que permitía modelar relaciones de largo alcance en las secuencias de entrada, mediante el cómputo paralelo.

La clave de los Transformers reside en el **multi-head self-attention**, que otorga al modelo la capacidad de ponderar dinámicamente la importancia relativa de distintos elementos dentro de una secuencia. Este mecanismo, además de ofrecer un rendimiento computacional más eficiente, mejora la capacidad de aprendizaje del modelo frente a secuencias largas o ruidosas, lo que resulta particularmente útil en dominios complejos como la predicción del flujo del tráfico urbano.

A diferencia de las RNN, que deben procesar las secuencias de manera secuencial, los Transformers permiten el aprendizaje paralelo y la captura simultánea de dependencias tanto locales como globales, lo que ha demostrado ser especialmente relevante para modelar patrones espacio-temporales complejos.

El uso de modelos Transformer en el ámbito de los sistemas inteligentes de transporte ha crecido significativamente en los últimos años, debido a su capacidad para capturar interacciones complejas entre nodos de una red vial y su evolución en el tiempo.

Un estudio reciente y particularmente relevante para este trabajo es el desarrollado por Chang et al. (2025), titulado *Transformer-based short-term traffic forecasting model considering traf-*

fic spatiotemporal correlation. En este artículo, los autores presentan **Trafficformer**, un modelo Transformer adaptado específicamente a la predicción del tráfico a corto plazo, integrando correlaciones espacio-temporales mediante máscaras espaciales y representaciones topológicas de la red viaria.

En cuanto a la arquitectura, el modelo Trafficformer consta de tres módulos fundamentales:

- (1) Extracción de características temporales mediante Multi Layer Perceptron o Perceptrones Multi Capa.
- (2) Interacción espacial basada en codificadores Transformer con máscaras de atención topológicas.
- (3) Predicción de velocidades mediante una red MLP final.

Esta arquitectura fue evaluada con el conjunto de datos del *Seattle Loop Detector Dataset*, superando a modelos clásicos como ARIMA, SVR y también a redes profundas como LSTM+MLP y TGG-LSTM, tanto en precisión (MAE, MAPE, RMSE) como en eficiencia computacional.

La inclusión de una máscara espacial, que filtra interacciones irrelevantes basándose en la topología vial y el tiempo de viaje entre nodos, permitió al modelo enfocarse en relaciones espacialmente significativas, lo que se tradujo en una mejora del 18 % en precisión frente a modelos equivalentes sin esta optimización. Esta capacidad de interpretar relaciones espaciales relevantes es fundamental en contextos como el tráfico urbano, donde las dependencias no son uniformes ni euclidianas, y dependen del trazado real de la red viaria.

El trabajo de Chang et al. (2025) demuestra que los modelos basados en Transformers no sólo son competitivos, sino que se posicionan como una opción de referencia para tareas de predicción del tráfico, permitiendo una mejor generalización, mayor interpretabilidad y adaptabilidad frente a cambios dinámicos en la red.

Este enfoque supone un salto cualitativo respecto a técnicas previas como LSTM, GRU o incluso GNN, al combinar lo mejor de los modelos de secuencia (captura temporal) con mecanismos estructurados de atención espacial. Además, su arquitectura modular y altamente paralelizable lo convierte en un candidato ideal para despliegues en entornos cloud, edge o híbridos, como los requeridos en la infraestructura del proyecto que nos ocupa.

Por todo ello, este modelo ha sido seleccionado como piedra angular sobre la cual se desarrollará la propuesta metodológica del presente trabajo, tanto en la fase de experimentación como en el diseño arquitectónico del modelo final.

2.2 Ventajas del uso de Transformers frente a otras arquitecturas

La arquitectura Transformer representa un avance significativo respecto a los modelos secuenciales (LSTM o GRU) y estructurales (como las GNN), tanto desde el punto de vista teórico como práctico.

En primer lugar, los modelos secuenciales dependen fuertemente del procesamiento secuencial, lo que limita la parallelización durante el entrenamiento y puede llevar a problemas de desvanecimiento o explosión del gradiente (leer en Wikipedia, la enciclopedia libre (2025)) en secuencias largas. Aunque han demostrado buen rendimiento en predicción temporal, su capacidad para modelar relaciones espaciales complejas es limitada. Por otro lado, las GNN destacan en la modelización espacial, pero presentan dificultades cuando se requiere combinar relaciones topológicas con dinámicas temporales de forma eficaz.

Los Transformers, y en particular la arquitectura Trafficformer, superan estas limitaciones al:

- **Separar explícitamente los componentes espaciales y temporales:** Trafficformer utiliza una MLP para extracción temporal y un codificador Transformer para interacción espacial, optimizando cada fase por separado.
- **Utilizar multi-head self-attention con enmascaramiento espacial:** Esto permite al modelo centrarse solo en las interacciones viales relevantes, mejorando la eficiencia y la precisión.
- **Permitir entrenamiento completamente paralelo:** Gracias al mecanismo de atención, el modelo puede ser entrenado de manera más rápida que una RNN convencional.

Los resultados experimentales de Chang et al. (2025) muestran que Trafficformer supera consistentemente a las otras propuestas mencionadas anteriormente en múltiples métricas de evaluación como MAE, RMSE y MAPE. Por ejemplo, en el dataset del *Seattle Loop Detector*, se observó una mejora de hasta el 18 % en error medio absoluto frente a los mejores modelos

recurrentes. Además, la arquitectura Transformer mostró una mayor capacidad de generalización frente a cambios dinámicos del tráfico. En la tabla 1 se puede ver a modo resumido todo lo dicho anteriormente.

Tabla 1: Comparativa entre arquitecturas en tareas de predicción del tráfico

Aspecto	RNNs (LSTM, GRU)	(LSTM, GNNs)	Transformers
Procesamiento secuencial vs. paralelo	Procesamiento secuencial con paralelización limitada	No aplicable (estructura estática)	Paralelización total mediante mecanismo de atención
Modelado espacial y temporal	Modelado temporal fuera, pero poco eficiente para relaciones espaciales	Excelente modelado espacial, dificultad para integrar dinámica temporal	Modelado explícito y eficiente para componentes espaciales y temporales
Rendimiento empírico	Rendimiento limitado en benchmarks de tráfico	Rendimiento moderado en tareas espaciales, sensible a ruido temporal	Mejores resultados en métricas MAE, RMSE y MAPE, mayor capacidad de generalización

En resumen, los modelos Transformer no solo ofrecen ventajas computacionales, sino que proporcionan una representación más rica y eficiente de las correlaciones espacio-temporales que caracterizan al problema de la predicción del tráfico urbano.

3 Objetivos y metodología de trabajo

En este capítulo se exponen los objetivos perseguidos con el desarrollo del presente Trabajo Fin de Máster, pasando por la infraestructura y tecnologías seleccionadas, y acabando con la metodología empleada para su ejecución. El trabajo se enmarca dentro del área del aprendizaje profundo, aplicados al problema de la predicción del flujo de tráfico urbano. El propósito es diseñar una solución capaz de anticipar el estado de la red viaria a corto plazo en la provincia de Bizkaia, utilizando datos abiertos y públicos de fuentes institucionales, lo cual permitirá evaluar tanto la capacidad de generalización de los modelos como su utilidad práctica en un contexto real.

Como se analizó en el capítulo anterior, la predicción del tráfico urbano enfrenta importantes retos derivados de la alta variabilidad temporal y la compleja dependencia espacial de los datos. Entre los trabajos analizados, destaca el modelo Trafficformer de Chang et al. (2025), que introduce un enfoque basado en Transformers mejorado mediante máscaras espaciales para filtrar ruido y enfocar la atención en interacciones relevantes entre nodos de la red. Aunque el modelo original fue diseñado para predecir velocidades de tráfico, su arquitectura resulta igualmente aplicable a la predicción de volúmenes de tráfico, esto es, la cantidad de vehículos que circulan por un punto dado en cada intervalo temporal, mediante una adaptación del preprocesamiento y del objetivo del modelo.

3.1 Objetivos del proyecto

El objetivo general de este Trabajo Fin de Máster es el diseño, implementación y evaluación de un sistema de predicción del tráfico urbano a corto plazo mediante el uso de técnicas de aprendizaje profundo, con especial énfasis en las redes neuronales de tipo Transformer. La solución desarrollada debe ser capaz de predecir con precisión el estado futuro del tráfico en diversos puntos de la red viaria de Bizkaia, aprovechando el valor añadido que ofrecen los datos abiertos procedentes de fuentes públicas, como los portales de Open Data del Gobierno Vasco. Para ello, se debe implementar y entrenar un modelo de predicción inspirado en el modelo Trafficformer, adaptado a volúmenes de tráfico. Este objetivo principal responde a la necesidad creciente de disponer de herramientas tecnológicas que permitan mejorar la gestión de la movilidad en entornos urbanos, facilitar la toma de decisiones estratégicas y operativas en

tiempo real, y anticiparse a situaciones potenciales de congestión. La precisión de la predicción se configura como un factor clave en la eficacia de sistemas ITS modernos, especialmente en contextos densamente poblados como el área metropolitana de Bilbao.

Como primer objetivo específico se pretende analizar el conjunto de datos disponibles en el catálogo de datos abiertos del Gobierno Vasco ubicado en Gobierno Vasco, Eusko Jaurlaritza (2025a). Dentro del mismo, se pretende identificar y analizar el Api de tráfico de Gobierno Vasco, Eusko Jaurlaritza (2025b), además de otros factores contextuales como la meteorología, que se ha demostrado influyente en la dinámica del tráfico y cuyo se ubica en Gobierno Vasco, Eusko Jaurlaritza (2025c).

Asimismo, se plantea como segundo objetivo específico la validación empírica de los modelos mediante experimentación con datos reales, evaluando su rendimiento con métricas reconocidas en el ámbito de la predicción, tales como el error cuadrático medio o RMSE, el error absoluto medio o MAE o el error porcentual absoluto medio o MAPE. La comparación entre distintos enfoques permitirá identificar las ventajas y limitaciones de cada uno, así como proponer mejoras que potencien su aplicabilidad.

Por último, se considera el tercer objetivo específico del trabajo ofrecer una reflexión crítica sobre el impacto potencial de este tipo de soluciones en el ámbito de la movilidad urbana y su eventual integración en sistemas de ayuda a la decisión de carácter público o privado. El valor añadido que se pretende aportar no reside únicamente en la capacidad predictiva del modelo, sino también en su posible contribución al desarrollo de una movilidad más eficiente, sostenible e inteligente.

3.2 Infraestructura y tecnologías empleadas

En este apartado se van a exponer todos los recursos seleccionados que van a servir de soporte para la consecución de los objetivos y el desarrollo del proyecto.

Para ello, se va a utilizar una infraestructura híbrida compuesta por recursos locales y servicios en la nube. Como equipo de desarrollo se va a seleccionar el equipo personal, que se compone por un procesador Intel Core i7-10700K de 10^a generación desbloqueado, con 32 GB de memoria RAM DDR4, haciendo uso de 2 discos SSD NVMe (512 GB + 1 TB) y un HDD de 2 TB como almacenamiento, equipado con una GPU Nvidia GTX 1070 con soporte CUDA y corriendo el Sistema Operativo Windows 11 Pro.

Este equipo permitirá realizar tareas de programación, experimentación y entrenamiento preliminar, si bien se anticipa que no será suficiente para entrenar modelos de gran tamaño o con grandes cantidades de datos.

Por ello, se deberá emplear una infraestructura remota para realizar los distintos entrenamientos. Dada la necesidad de cómputo intensivo, se contempla el uso de una instancia EC2 en Amazon Web Services (AWS), con una AMI optimizada para entrenamiento de modelos con PyTorch (por ejemplo, la AMI de Deep Learning de AWS con soporte GPU Tesla T4 o V100). Esta infraestructura permitirá acelerar significativamente el proceso de entrenamiento y validación.

Asimismo, se va a hacer uso de un servidor doméstico con TrueNAS Scale Electric Eel como sistema operativo, equipado con un procesador Intel Core i5-7400 de 7^a generación, con 16 GB de RAM DDR4 y dos discos duros de 4 TB configurados en RAID 1. Este servidor albergará una base de datos MongoDB para almacenamiento estructurado de datos y MinIO como sistema de almacenamiento tipo S3, útil para almacenamiento masivo y backups. Todos esos servicios van a funcionar como contenedores Docker.

En cuanto al software, se van a seleccionar los lenguajes de programación Kotlin (con Gradle como herramienta de construcción) y Python (con Poetry como gestor de dependencias y de empaquetamiento). En cuanto a los entornos de desarrollo, se ha decidido hacer uso de la suite de Jetbrains, puesto que facilitan licencias para estudiantes y son muy completas (ofreciendo muchas funcionalidades y asistentes que hacen más productiva tu jornada). Estos IDE son IntelliJ IDEA (para el desarrollo con Kotlin) y PyCharm (para el desarrollo con Python). Para el control de versiones se va a emplear Git, con repositorios en GitHub o GitLab.

Para el desarrollo del modelo de predicción de tráfico se ha optado por utilizar PyTorch como backend principal. Esta decisión se fundamenta en la flexibilidad que ofrece esta biblioteca para la implementación de arquitecturas avanzadas, como redes neuronales recurrentes (LSTM, GRU), redes convolucionales (CNN), Graph Attention Networks (GAT) y modelos basados en Transformers. Además, PyTorch cuenta con una comunidad investigadora muy activa y un ecosistema maduro que incluye librerías especializadas como PyTorch Geometric o HuggingFace Transformers, altamente relevantes para el ámbito de este trabajo. A diferencia de otros entornos como TensorFlow/Keras, que destacan por su facilidad de uso en fases de prototipado, PyTorch ofrece un control más explícito sobre el flujo de datos y la personalización del entrenamiento, lo que resulta especialmente valioso en proyectos que requieren ajustar arquitecturas

de manera específica para capturar correlaciones espaciotemporales complejas en el tráfico urbano.

Estas decisiones se ven respaldadas por análisis comparativos recientes, como el realizado por DataCamp, donde se destaca que *PyTorch* suele ser más rápido y proporciona mejores capacidades de depuración que *Keras*. Estas características lo hacen especialmente adecuado para entornos de investigación y desarrollo de modelos complejos citedatacamp2023.

Además, UnfoldAI ofrece un análisis detallado sobre las fortalezas y debilidades de los principales frameworks de aprendizaje profundo. En su estudio comparativo se destaca que *PyTorch* proporciona una mayor flexibilidad y control, lo que lo convierte en la opción preferida en entornos de investigación, mientras que *Keras* sobresale por su simplicidad y facilidad de uso, resultando ideal para usuarios principiantes y tareas de prototipado rápido UnfoldAI (2024).

3.3 Metodología de trabajo

Para la consecución de los objetivos, tomando como referencia la solución Trafficformer, se ha pensado en estructurar el proyecto de la siguiente manera:

- Adquisición y preprocesamiento de datos.
 - **Datos:** se utilizarán datos abiertos provenientes del portal Open Data de Euskadi, principalmente de sensores que registran el número de vehículos que atraviesan puntos específicos de la red viaria en intervalos regulares.
 - **Preprocesamiento:** se realizará la agregación temporal si es necesario (por ejemplo, a intervalos de 5 minutos), imputación de valores perdidos, normalización y generación de ventanas deslizantes para construir las series históricas.
- Arquitectura del modelo. La arquitectura seguirá los principios del modelo Trafficformer, la cual se puede apreciar en la figura 2, con los siguientes componentes adaptados:
 - **Extracción temporal:** un MLP de dos capas se encargará de extraer patrones no lineales de las series de volumen de vehículos por punto de control.
 - **Máscara espacial:** se construirá una matriz de adyacencia basada en la topología de la red viaria, utilizando distancias reales o tiempos de viaje para definir relaciones de vecindad relevantes.

- **Codificador Transformer:** mediante multi-head attention, el modelo extrae relaciones espaciales entre nodos cercanos, permitiendo una representación contextualizada del tráfico.
- **Predicción:** se proyectan las representaciones espacio-temporales a una predicción del número de vehículos que circularán por cada punto de la red en el siguiente intervalo.
- Entrenamiento y evaluación.
 - **Métricas:** se utilizarán MAE, RMSE y MAPE para comparar la predicción de conteos de vehículos.
 - **Modelos base:** se tendrá como referencia el modelo MLP desarrollado anteriormente.
 - **Validación:** se aplicará validación temporal (walk-forward) para simular condiciones reales de predicción.

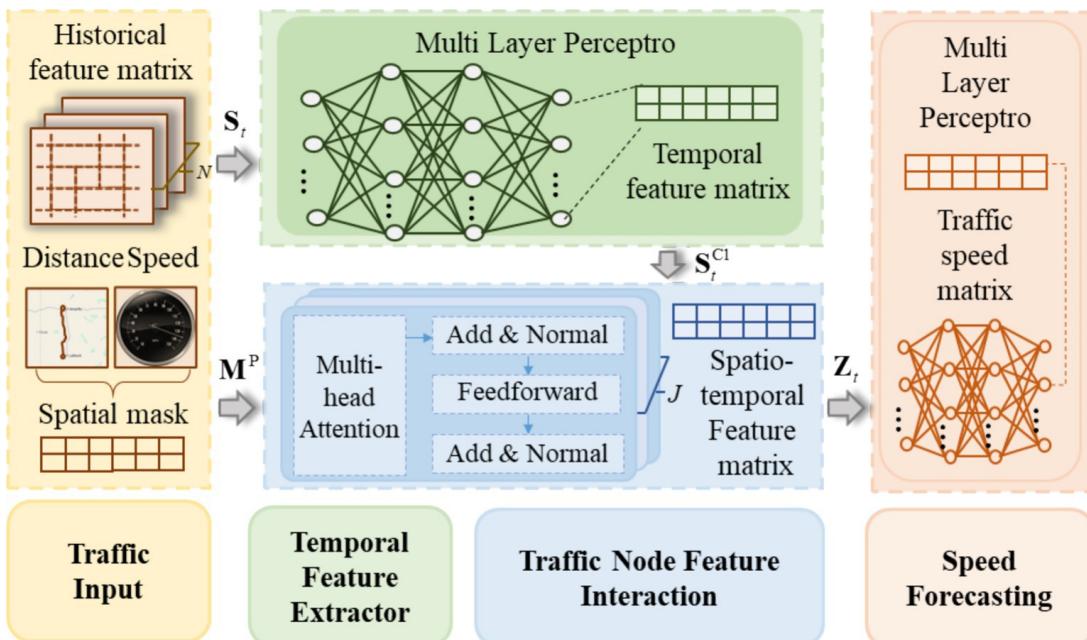


Figura 2: Arquitectura general del modelo Trafficformer Chang et al. (2025)

Como consecuencia, se prevé llevar a cabo las siguientes tareas.

- Extracción y almacenamiento de datos. Desarrollo de un módulo en Kotlin para conectar con las públicas de tráfico y meteorología de Open Data Euskadi. Almacenamiento de los datos crudos en MongoDB y objetos binarios (imágenes, CSVs temporales) en MinIO.

- Preprocesamiento y transformación de datos. Conversión de datos crudos en estructuras listas para entrenamiento, mediante scripts en Python. Aplicación de técnicas de normalización, resampleo temporal y gestión de datos faltantes.
- Entrenamiento de modelos neuronales. Implementación de un modelo base MLP para establecer una línea base de rendimiento. Posterior implementación de un modelo Transformer adaptado al tráfico, con atención espacio-temporal y uso de máscaras de conectividad vial.
- Evaluación comparativa. Evaluación de los modelos con un conjunto de métricas estándar (MAE, RMSE, MAPE). Comparativa entre modelos para seleccionar el más adecuado para el caso de uso.
- Validación y conclusiones. Validación en escenarios reales y extracción de conclusiones sobre la utilidad del modelo. Estudio de la viabilidad de su aplicación en entornos reales, como el soporte a ITS o planificación urbana.

4 Construcción del dataset y arquitectura del sistema

En este capítulo se describe el diseño, desarrollo e implementación del sistema responsable de generar el dataset empleado en el modelo de predicción de tráfico. Se parte de la identificación y análisis de las fuentes de datos disponibles, que incluyen mediciones de flujo vehicular, condiciones meteorológicas e incidencias viales. A continuación, se detallan las decisiones técnicas adoptadas en la arquitectura del sistema de integración, los patrones de diseño utilizados y el procedimiento empleado para consolidar todas las observaciones en una única estructura homogénea: la clase MobilitySnapshot. Finalmente, se justifican aspectos clave como la granularidad temporal del dataset, los criterios de selección de variables y las estrategias de persistencia.

4.1 Fuentes de datos y análisis de disponibilidad

Durante la primera fase del proyecto, se realizó un análisis exhaustivo de las fuentes de datos abiertas disponibles para la provincia de Bizkaia en el ámbito de los ITS. La fuente de datos principal se corresponde con el API de Tráfico del portal de Open Data del Gobierno Vasco Gobierno Vasco, Eusko Jaurlaritza (2025b). Se identificaron tres orígenes principales de datos:

- **Gobierno Vasco:** mediante el API de Open Data Euskadi, se obtuvo información de aforos de tráfico e incidencias viales.
- **Ayuntamiento de Bilbao:** también a través de Open Data, se extrajeron series temporales de datos de tráfico en tiempo real.
- **Diputación Foral de Bizkaia:** al no disponer de un endpoint público, se logró contactar con los técnicos responsables y obtener acceso a sus datos mediante ficheros descargables proporcionados manualmente.

La cobertura de los datos obtenidos se representa en la tabla 3, extraída y adaptada de la documentación técnica del repositorio del proyecto.

Tabla 3: Cobertura de datos por fuente y tipo.

SouceId	Organización	Meters	Flows	Incidences
1	Gobierno Vasco	✓	✓	✓
2	Diputación Foral de Bizkaia	✋	✋	✓
3	Diputación Foral de Álava	✗	✗	✓
4	Diputación Foral de Gipuzkoa	✓	✗	✓
5	Ayuntamiento Bilbao	✓	✓	✓
6	Ayuntamiento Vitoria-Gasteiz	✓	✓	✓
7	Ayuntamiento de Donostia-San Sebastián	✓	✗	✓

Leyenda:

- ✓ Datos existentes y descargados correctamente.
- ✗ No existen datos para ese sourceId en OpenData.
- ✋ Datos obtenidos de forma externa e introducidos manualmente mediante programación.

El caso de uso actual trata de cubrir la provincia de Bizkaia, por lo que únicamente van a ser necesarias las fuentes de datos con identificadores sourceId 1, 2 y 5.

Las incidencias se obtuvieron completamente para todos los sourceId.

En cuanto a la obtención de los datos meteorológicos, si bien es cierto que existe el API de meteorología del portal de Open Data del Gobierno Vasco Gobierno Vasco, Eusko Jaurlaritza (2025c), éste presenta numerosos fallos a la hora de usarlo. Un claro ejemplo es la figura 3.

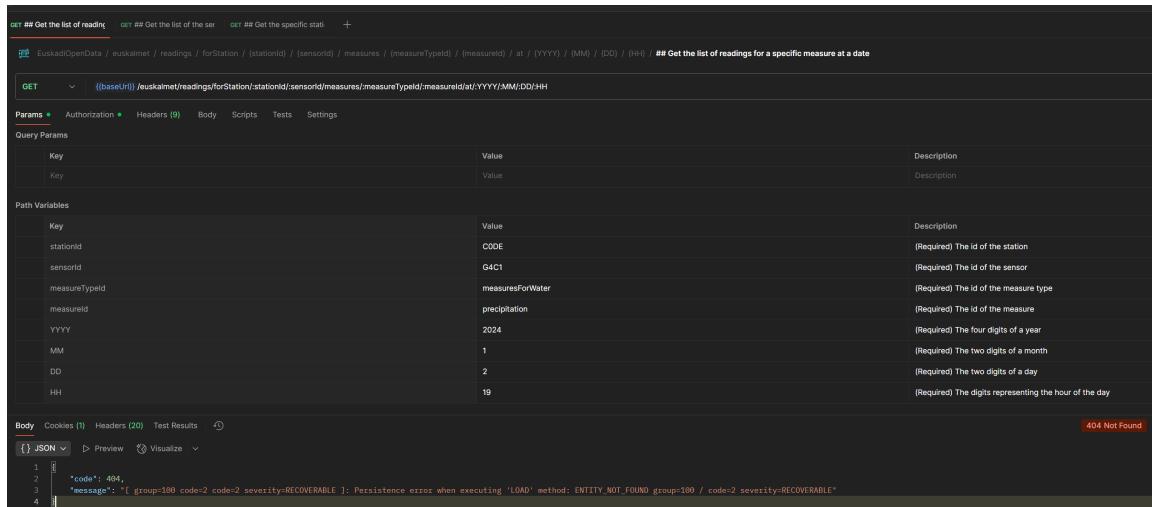


Figura 3: Ejemplo de error a la hora de consultar el API de Euskalmet.

Por ello, no se tenía la certeza de si se iban a obtener datos correctos, por lo que se pensó en maneras alternativas de obtener los propios datos. Así, en esa búsqueda de caminos se encontró, dentro del mismo portal de Open Data del Gobierno Vasco, las lecturas recogidas por las estaciones meteorológicas del año 2024 de forma bruta en formato XML, accesible a continuación Eusko Jaurlaritza / Gobierno Vasco (2024).

Modelo de datos almacenado

Una vez explicadas las fuentes de datos, se podría comenzar a explicar cómo se van a almacenar dichos datos. En la figura 4 se puede ver las clases persistidas en la base de datos.

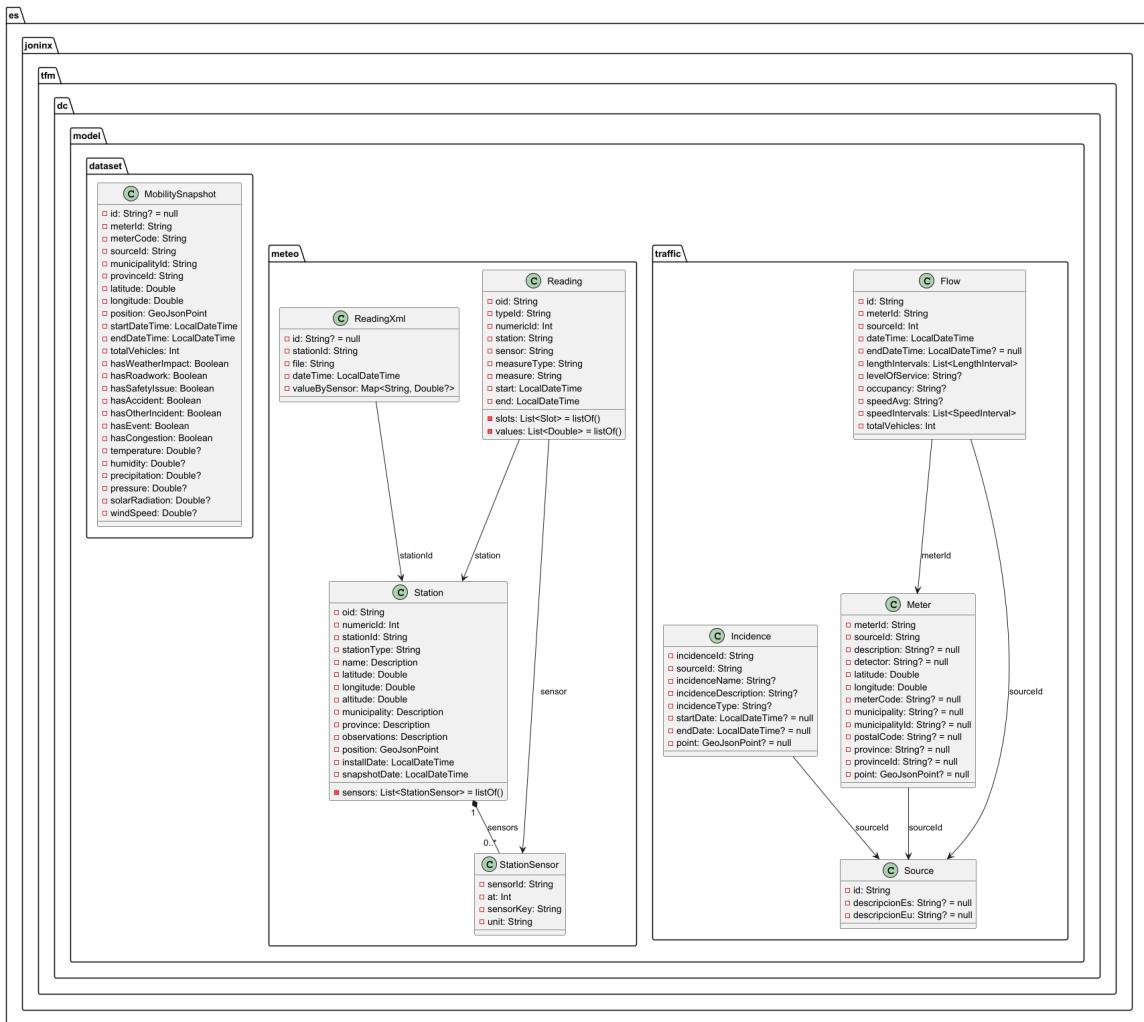


Figura 4: Modelo de clases en UML de las entidades principales del sistema.

A continuación, se describen las entidades persistidas en base de datos, divididas por paquetes funcionales según su origen y propósito dentro del sistema: tráfico, meteorología y dataset resultante. Los datos de incidencias se incluyen dentro del paquete de tráfico, puesto que así viene encasillado en el API consultado. Cada clase representa un documento en la base de datos MongoDB, adaptado a una estructura flexible y orientada a documentos.

4.1.0.1 Flow

Representa una medición de flujo de tráfico captada por un aforador en una fecha y hora determinadas. Entre sus atributos destacan:

- **id:** identificador único del registro.
- **meterId:** identificador del aforador que ha generado la medición.

- `sourceId`: origen de los datos (Gobierno Vasco, Ayuntamiento, etc.).
- `dateTime` y `endDateTime`: marca temporal de la medición. Puede que la marca de final no esté informada.
- `totalVehicles`: número total de vehículos detectados.
- `speedAvg`: velocidades medias de los vehículos detectados.
- `lengthIntervals` y `speedIntervals`: listas de intervalos por longitudes y velocidades, útiles para análisis más detallados. No se usa en este caso de uso.

4.1.0.2 Meter

Contiene la información estática de los aforadores:

- `meterId`, `sourceId`: identificadores del aforador y de la fuente de datos.
- `latitude`, `longitude` y `point`: ubicación geográfica. Útil el punto (tipo GeoJSON), puesto que es un índice geoespacial y se pueden realizar operaciones sobre el mismo (búsquedas por cercanía, distancias, etc.).
- `postalCode`, `provinceId`, `municipalityId`: metadatos administrativos.

4.1.0.3 Incidence

Recoge información sobre incidencias viales que pueden afectar al tráfico:

- `incidenceId`, `sourceId`: identificadores del evento y de la fuente de datos.
- `incidenceName` e `incidenceDescription`: información descriptiva textual de la incidencia.
- `incidenceType`, `incidenceLevel`: tipificación normalizada para su posterior uso como variable categórica.
- `cause`, `road`, `pkStart`, `pkEnd`, `cityTown`, `province`, `autonomousRegion`, `carRegistration`: información adicional de la incidencia. Recoge datos como la carretera, los puntos kilométricos de inicio y de fin e información administrativa. No se tratará en este caso de uso.

- `startDate`, `endDate`: intervalo de vigencia de la incidencia. Esta información es relevante.
- `point`: ubicación geoespacial (tipo GeoJSON). Útil para realizar búsquedas por cercanía.

4.1.0.4 Source

Representa los orígenes oficiales de datos, como el Gobierno Vasco o ayuntamientos. Sus campos identifican la organización y su descripción.

4.1.0.5 Reading

Es la clase principal para representar una lectura meteorológica horaria del API. Sin embargo, no se ha usado esta entidad para construir el dataset.

- `oid`: identificador de la lectura.
- `typeId`: tipo de la lectura.
- `station`: estación meteorológica emisora.
- `sensor`: sensor responsable de la medición (`StationSensor`).
- `measureType` y `measure`: categorización de la medición.
- `start`, `end`: momento para el cual se ha realizado la medición.
- `slots`, `values`: forma en la que tiene el API de almacenar las mediciones. Cada elemento del slot se corresponde a un elemento de values, en la misma posición, indicando lo que se mide y su valor.

4.1.0.6 ReadingXml

Clase específica para lecturas meteorológicas históricas extraídas desde ficheros XML. A diferencia de `Reading`, puede contener estructuras agrupadas por múltiples sensores. Es la clase definitiva empleada para construir el dataset.

- `id`: identificador de la lectura.
- `stationId`: identificador de la estación meteorológica de la lectura.

- `file`: fichero de donde se ha extraído el dato de medición.
- `dateTime`: momento para el cual se ha realizado la medición.
- `valueBySensor`: forma en la que se almacenan las mediciones. El elemento clave identifica el tipo de medición y el valor indica el valor propio de la medición.

4.1.0.7 Station

Define las estaciones meteorológicas que proporcionan las lecturas:

- `stationId`: identificador único de la estación de medición.
- `name, municipality, province`: información administrativa.
- `latitude, longitude, altitude` y `position`: coordenadas geográficas.
- `sensors`: lista de sensores instalados.

4.1.0.8 StationSensor

Define la configuración física de un sensor en una estación:

- `sensorId, sensorKey`: clave de sensor y código técnico.
- `unit, at`: unidad de medida y altura del sensor (en centímetros).

4.1.0.9 MobilitySnapshot

Es la entidad clave que representa un punto de datos enriquecido para el modelo predictivo.
Cada snapshot incluye:

- `id`: identificador único del dato.
- `meterId, meterCode`: información identificativa del aforador del cual se ha extraído la información de Flow.
- `totalVehicles`: indica la cantidad de vehículos que han pasado por este punto entre en el espacio temporal definido.

- `latitude, longitude, position`: información geoespacial del snapshot. Nótese que el campo posición es de tipo GeoJson.
- `startDateTime, endDateTime`: indica entre qué momentos es válido este snapshot.
- `hasAccident, hasWeatherImpact, hasConstruction, etc`: indica si durante este espacio temporal cerca de este punto ha ocurrido alguna incidencia del tipo.
- `temperature, humidity, windSpeed, solarRadiation, pressure, etc`: indica los valores meteorológicos del propio snapshot.

Esta clase es el resultado final del proceso de integración de datos y constituye la base sobre la que se entrena el modelo de predicción de tráfico.

4.2 Recolección, arquitectura y patrones de diseño empleados

El software encargado de la recolección de datos se ha desarrollado en **Kotlin con Spring Boot**, aplicando principios de arquitectura hexagonal y múltiples patrones de diseño para garantizar su mantenibilidad y robustez. Entre los patrones de diseño empleados destacan:

- **Builder**: utilizado en la generación del dataset a través de la clase `MobilitySnapshotBuilder`.
- **Repository**: todas las operaciones de acceso a datos (`FlowRepository, MeterRepository`, etc.) se abstraen en repositorios desacoplados.
- **Service Layer**: la lógica de negocio reside en servicios como `FlowService, MeterService, IncidenceService`.
- **DTO (Data Transfer Object)**: se emplean DTOs para el intercambio de datos entre capas, evitando el acoplamiento con los modelos de persistencia.
- **Helper/Utility Classes**: utilidades para parseo, validación y transformación de datos (por ejemplo, para el tratamiento de ficheros XML meteorológicos).
- **Facade**: fachadas que agrupan operaciones complejas en interfaces sencillas, facilitando la integración con servicios externos.
- **Factory y Singleton**: empleados para instanciar objetos según el origen de datos y para servicios centrales, respectivamente.

El sistema de integración y procesamiento de datos ha sido desarrollado en Kotlin, un lenguaje moderno que combina características orientadas a objetos y funcionales. La clase `MobilitySnapshotGenerator`, pieza central del sistema, hace uso de varias construcciones idiomáticas que representan buenas prácticas del lenguaje y potencian la expresividad, la seguridad y la eficiencia en tiempo de desarrollo.

A continuación se destacan las características más reseñables del uso de Kotlin en dicha implementación:

- **Inferencia de tipos:** permite declarar variables sin especificar explícitamente su tipo cuando este es deducible por el compilador, lo que mejora la legibilidad. Por ejemplo:

```
val intervals = mutableListOf<Pair<LocalDateTime, LocalDateTime>>()
```

- **Funciones de orden superior y operadores de colección:** se hace uso extensivo de funciones como `mapNotNull`, `groupBy`, `sumOf`, y `forEachIndexed`, lo que refleja el paradigma funcional del lenguaje.

```
val groupedByMeter = flows.groupBy { it.meterId }
val snapshots = groupedByMeter.mapNotNull { (meterId, flowList) ->
    ...
}
```

- **Acceso seguro a valores opcionales con let:** se utiliza para gestionar accesos a valores almacenados en cachés evitando estructuras condicionales explícitas.

```
meterToStationCache[meterId] ?.let {
    log.debug("Cache HIT: meterId=$meterId → stationId=$it")
    return it
}
```

- **Operador Elvis (?:):** permite proporcionar valores por defecto ante posibles nulos, mejorando la seguridad frente a `NullPointerException`.

```
val meters = meterRepository.findAllBySourceIdIn(sourceIds)
    .collectMap { it.meterId }
    .block() ?: emptyMap()
```

- **Lambdas expresivas y destructuración de pares:** se utilizan para recorrer colecciones de forma clara, aprovechando las capacidades de Kotlin para trabajar con estructuras complejas.

```
intervals.forEachIndexed { idx, (intervalStart, intervalEnd) -> ...  
}
```

- **Inyección de dependencias vía constructor:** siguiendo los principios de inversión de dependencias (en otras palabras, un término que mezcla los conceptos de Inyección de Dependencias e Inversión de Control), se definen todos los componentes del servicio a través del constructor primario.

```
class MobilitySnapshotGeneratorService(  
    private val cfg: Cfg,  
    private val snapshotBuilder: MobilitySnapshotBuilder,  
    ...  
)
```

- **Uso de companion object para logging:** se emplea una instancia estática del logger utilizando el enfoque idiomático del lenguaje.

```
companion object {  
    val log: Logger = LogManager.getLogger(this::class.java)  
}
```

- **Uso responsable de colecciones mutables:** aunque se utilizan listas mutables para la generación de intervalos y resultados temporales, estas estructuras se manejan de forma controlada y local, siguiendo las recomendaciones del lenguaje.

El uso de estas características idiomáticas refleja una implementación robusta, alineada con las buenas prácticas modernas del ecosistema Kotlin, y facilita tanto el mantenimiento del código como su extensibilidad futura.

La decisión de utilizar una base de datos **NoSQL, MongoDB**, responde a la necesidad de trabajar con estructuras flexibles, heterogéneas y de gran volumen, propias del contexto del proyecto.

4.3 Decisiones de construcción del dataset

Tras la integración de las fuentes de datos de tráfico, meteorología e incidencias, se llevó a cabo un proceso de análisis y consolidación de información para la construcción del dataset final utilizado por el modelo de predicción. Este dataset, modelado a través de la clase MobilitySnapshot, resume en cada observación todos los factores que pueden influir en la intensidad del tráfico en un instante y ubicación concretos. En los siguientes apartados se detallan las decisiones tomadas en relación con cada fuente de información, comenzando por la tipificación de incidencias viales.

Obtención y estructuración de incidencias

A partir del análisis exploratorio de las incidencias disponibles en la base de datos, se identificaron distintos tipos reportados a lo largo del tiempo por las diferentes entidades públicas. La tabla 5 resume las categorías encontradas, junto con su frecuencia absoluta. Es importante tener en cuenta que estos datos corresponden a toda la CAPV, no solo a Bizkaia.

Tabla 5: Frecuencia de incidencias por tipo original

Frecuencia	Tipo de incidencia
41792	Puertos de montaña
8544	Obras
8391	Vialidad invernal tramos
7594	Seguridad vial
3155	Accidente
2005	Otras incidencias
203	Meteorológica
165	OTRO
135	OBRA
78	EVEN
3	Retención

Con base en esta información, se procedió a una consolidación tipológica siguiendo tres crite-

rios principales:

- Evitar la existencia de categorías con muy baja frecuencia.
- Unificar variantes ortográficas o nomenclaturas inconsistentes (por ejemplo, Obras y OBRA).
- Mantener las categorías que aportan valor explicativo al modelo de predicción.

El resultado es una agrupación validada que se resume en la tabla 7.

Tabla 7: Agrupación final de incidencias para el modelo predictivo

Categoría general	Tipos incluidos	Total casos	Variable sugerida
WEATHER	Puertos de montaña, Vialidad invernal tramos, Meteorológica	50.386	hasWeatherImpact
ROADWORK	Obras, OBRA	8.679	hasRoadwork
SAFETY	Seguridad vial	7.594	hasSafetyIssue
ACCIDENT	Accidente	3.155	hasAccident
OTHER	Otras incidencias, OTRO	2.170	hasOtherIncident
EVENT	EVEN	78	hasEvent
CONGESTION	Retención	3 (opcional)	hasCongestion

Las variables anteriores se incorporan al dataset como flags booleanos en la clase MobilitySnapshot, permitiendo representar si una incidencia de dicha categoría estaba activa o no en el momento de cada observación. Las variables mínimas propuestas son:

Listing 1: Variables mínimas de incidencias en MobilitySnapshot

```
val hasWeatherImpact: Boolean
```

```
val hasRoadwork: Boolean  
val hasSafetyIssue: Boolean  
val hasAccident: Boolean  
val hasOtherIncident: Boolean
```

Adicionalmente, y si el volumen de datos lo justifica en futuras versiones del dataset, se podrían incorporar:

Listing 2: Variables opcionales

```
val hasEvent: Boolean  
val hasCongestion: Boolean
```

Selección e integración de variables meteorológicas

El sistema de captación meteorológica empleado en este proyecto incluye un conjunto amplio y heterogéneo de sensores distribuidos en estaciones automáticas de observación repartidas por la CAPV. Cada estación contiene múltiples sensores, y cada sensor puede registrar una variable distinta a una altura determinada del suelo (por ejemplo, 0 cm, 1050 cm o 2200 cm).

Como se detalla en el Anexo A, se dispone de un amplio conjunto de sensores meteorológicos, cada uno con una codificación propia y una descripción técnica. En concreto, se incluyen más de 30 tipos distintos de sensores, desde condiciones atmosféricas hasta mediciones marinas o subterráneas. Algunos ejemplos de variables disponibles son: temperatura del aire (Tem.Aire), humedad relativa (Humedad), radiación solar (Irradia.), presión atmosférica (Presión), dirección del viento (Dir.Med.), visibilidad, y muchas otras relacionadas con agua o condiciones marítimas.

Dado el objetivo de este proyecto es predecir el flujo de tráfico urbano, se realizó una **selección cuidadosa de las variables meteorológicas más relevantes**, descartando aquellas que, por su naturaleza o localización (ej. marítimas), no presentaban una influencia directa sobre la movilidad terrestre urbana.

Las variables seleccionadas, junto con su justificación práctica, se muestran en la tabla 9.

Tabla 9: Selección de sensores meteorológicos y su relevancia para la predicción de tráfico

Categoría	SensorKey base	Justificación
Temperatura del aire	Tem_Aire__a_*	Influye en el comportamiento de conducción y el volumen de tráfico.
Humedad relativa	Humedad__a_*	Afecta la visibilidad y adherencia de los neumáticos.
Precipitación acumulada	Precip___a_*	Altamente correlacionada con congestiones y reducción de velocidad.
Presión atmosférica	Presion__a_*	Indicador indirecto de cambios climáticos significativos.
Radiación solar	Irradia___a_*	Relacionada con condiciones extremas de iluminación y temperatura.
Velocidad media del viento	Vel_Med__a_*	Indicador de fenómenos meteorológicos adversos (rachas, tormentas).

Para cada una de estas categorías, se escoge **el sensor más representativo o más cercano al suelo** disponible en cada estación. Este criterio asegura la mayor homogeneidad entre estaciones y la mayor cercanía posible a las condiciones experimentadas en carretera.

Finalmente, estas variables se integran dentro de cada observación del dataset final mediante su asociación espacio-temporal al flujo de tráfico correspondiente, quedando reflejadas como campos meteorológicos en la clase MobilitySnapshot:

Listing 3: Variables meteorológicas integradas en MobilitySnapshot

```

val temperature: Double?
val humidity: Double?
val precipitation: Double?
val pressure: Double?
val solarRadiation: Double?

```

```
val windSpeed: Double?
```

Estas variables permiten modelar de forma efectiva el efecto de las condiciones meteorológicas sobre la movilidad urbana, mejorando la capacidad predictiva del modelo de aprendizaje profundo.

Fusión e integración: la clase MobilitySnapshot

La entidad central para la construcción del dataset es la clase `MobilitySnapshot`, que representa un registro aglutinado por instante temporal y ubicación, integrando la información de:

- **Flujos de tráfico (Flow)**: variables como `meterId`, `dateTime`, `totalVehicles`, y metadatos geográficos.
- **Meteorología**: valores de los sensores seleccionados asociados espacial y temporalmente al flujo.
- **Incidencias**: información de incidencias activas cercanas en tiempo y espacio, codificadas según la tipología definida.

El proceso de generación de las observaciones enriquecidas se implementa en el servicio `MobilitySnapshot` dentro del método `generateSnapshots()`. Este método ejecuta de forma secuencial la construcción de objetos `MobilitySnapshot`, cada uno de los cuales encapsula una observación consolidada de movilidad para un instante y punto geográfico concretos. La lógica está diseñada para ser robusta y escalable, gestionando datos de múltiples fuentes (flujos, meteorología e incidencias) mediante el uso de procesamiento por intervalos y agrupación por sensor.

El flujo completo queda representado en la Figura 5, mediante un diagrama de secuencia UML, que ilustra claramente el intercambio entre repositorios, lógica de negocio y servicios de persistencia.

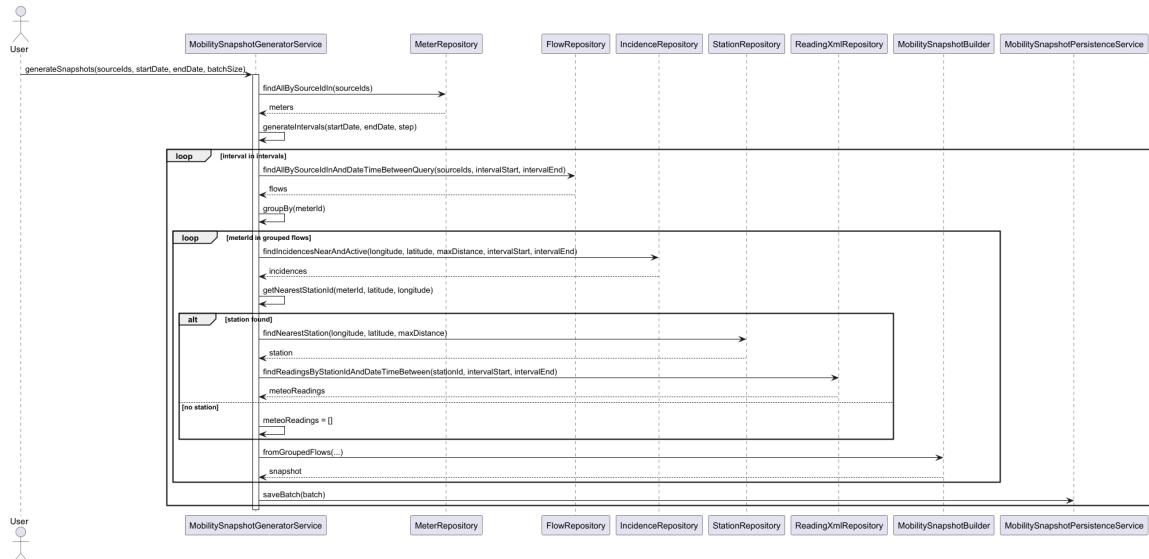


Figura 5: Diagrama de secuencia de integración y persistencia en MobilitySnapshot.

El código fuente completo del componente encargado de llevar a cabo esta integración puede consultarse en el Anexo B, donde se incluye la clase `MobilitySnapshotGeneratorService` desarrollada específicamente para este propósito.

El proceso se puede descomponer en los siguientes pasos:

1. **Inicialización:** se obtienen todos los aforadores activos (Meter) correspondientes a los identificadores de fuente (sourceIds) indicados.
2. **Ventanas temporales:** se generan intervalos de tiempo equiespaciados de 30 minutos entre las fechas de inicio y fin.
3. **Extracción de flujos:** para cada intervalo, se recuperan todos los flujos de vehículos (Flow) registrados en ese rango temporal y se agrupan por aforador (meterId).
4. **Procesamiento por aforador:** para cada grupo de flujos:
 - Se localiza el aforador correspondiente y se calcula el número total de vehículos en la ventana.
 - Se consultan las incidencias viales (Incidence) activas y geoespacialmente cercanas (500 metros) en el intervalo.
 - Se determina la estación meteorológica más cercana usando una cache interna. Si existe, se recuperan las lecturas (Reading) correspondientes al intervalo temporal.

5. **Construcción del snapshot:** con todos los datos anteriores, se construye el objeto MobilitySnapshot mediante el MobilitySnapshotBuilder.

6. **Persistencia por lotes:** los snapshots generados se agrupan en bloques de tamaño configurable (por defecto 500) y se guardan en la base de datos de forma transaccional.

Para procesar los datos por ventanas temporales consecutivas, se implementa una función utilitaria llamada generateIntervals(), que permite dividir un periodo de tiempo en subintervalos de duración fija. Este procedimiento es fundamental en el procesamiento de datos temporales y en la agregación de observaciones como MobilitySnapshot.

El algoritmo se describe de la siguiente forma.

Descripción del funcionamiento:

- La función recibe un instante inicial (start), un instante final (end) y una duración fija (step).
- Crea una lista vacía de intervalos temporales.
- Mediante un bucle, se van generando pares de fechas consecutivos, avanzando de step en step.
- Si el último intervalo excede el tiempo final, se ajusta automáticamente para que el límite superior sea exactamente end.
- Devuelve la lista completa de intervalos como pares de fechas (Pair<start, end>).

Ejemplo práctico:

Listing 4: Ejemplo de uso con intervalos de 30 minutos

```
generateIntervals(  
    start = 2024-01-01T00:00,  
    end = 2024-01-01T01:00,  
    step = Duration.ofMinutes(30)  
)
```

El resultado de este ejemplo sería:

```
[  
(2024-01-01T00:00, 2024-01-01T00:30),  
(2024-01-01T00:30, 2024-01-01T01:00)  
]
```

Este mecanismo permite particionar grandes volúmenes de datos históricos en unidades manejables, facilitando la agregación de información por tramos y reduciendo la carga computacional de los algoritmos de predicción.

Este proceso se ejecuta para todas las ventanas temporales dentro del periodo solicitado, permitiendo cubrir días, semanas o incluso meses de observaciones. Se emplean caches internas y colecciones reactivas para optimizar el rendimiento del sistema y permitir el escalado.

Consideraciones finales para la generación del dataset

Una decisión fundamental en la construcción del dataset es la selección del intervalo temporal con el que se van a agrupar las observaciones. Este proceso, conocido como *resampling temporal*, consiste en consolidar las mediciones de tráfico (Flow) en tramos de tiempo consecutivos y homogéneos. Esta práctica es común en el análisis de series temporales y presenta múltiples ventajas:

- **Reducción de dispersión:** ayuda a evitar huecos en las series temporales, suavizando el ruido y mejorando la capacidad de generalización del modelo.
- **Homogeneización del dataset:** garantiza que todas las observaciones estén espaciadas en el tiempo con la misma frecuencia, lo cual es esencial para el entrenamiento supervisado.
- **Facilitación de integración con otras fuentes:** permite alinear los flujos con los datos meteorológicos e incidencias, los cuales pueden tener frecuencias distintas o menos regulares.
- **Reducción de volumen de datos:** disminuye la cantidad de registros generados, lo que mejora la eficiencia tanto en almacenamiento como en entrenamiento de modelos.

La elección del intervalo temporal óptimo depende del equilibrio entre granularidad, capacidad computacional y riqueza informativa. Las alternativas más comunes son:

- **Intervalos de 1 hora:** proporcionan una agregación suficiente para análisis diarios o semanales, son compatibles con muchos datasets públicos, pero pueden enmascarar variaciones de corto plazo (como retenciones puntuales).
- **Intervalos de 30 minutos:** capturan mejor los picos de tráfico y permiten reflejar dinámicas más rápidas (por ejemplo, alteraciones por accidentes o climatología adversa). Este intervalo ha sido el seleccionado en este proyecto como punto de partida, ya que ofrece un buen compromiso entre granularidad y manejabilidad.
- **Intervalos de 15 minutos o menos:** pueden resultar útiles si se dispone de datos de alta frecuencia, pero también pueden introducir más ruido o generar datasets demasiado voluminosos para ciertos entornos de cómputo.

Tabla 11: Comparativa entre intervalos temporales posibles para el resampling

Criterio	15 minutos	30 minutos	1 hora
Granularidad	Alta	Media	Baja
Captura de picos	Muy buena	Buena	Limitada
Volumen de datos	Alto	Medio	Bajo
Riesgo de ruido	Alto	Bajo-Medio	Bajo
Compatibilidad con fuentes externas	Menor	Alta	Alta
Recomendado para	Predicción muy reactiva o microanálisis	Equilibrio general	Análisis macro o planificación

En la Tabla 11 se puede apreciar resumidamente las consideraciones descritas anteriormente.

Decisión tomada: el sistema desarrollado trabaja por defecto con ventanas de **30 minutos**. Esta elección permite capturar transiciones relevantes en la movilidad sin incrementar en exceso la complejidad del dataset. No obstante, la arquitectura del generador (`MobilitySnapshotGeneratorService`) permite modificar este parámetro fácilmente para experimentar con alternativas. Por ejemplo:

- Reducir a 15 minutos si se observan patrones planos o poca sensibilidad a eventos puntuales.
- Aumentar a 1 hora si el volumen de datos es demasiado elevado o si se prioriza una vista agregada del tráfico.

5 Desarrollo experimental, comparación de modelos y resultados

5.1 Planteamiento experimental y justificación

El presente capítulo recoge el desarrollo experimental llevado a cabo para validar la hipótesis principal del Trabajo Fin de Máster: que el uso de una arquitectura basada en *Transformers* con mecanismos de atención espacial, como *Trafficformer*, permite mejorar la capacidad predictiva respecto a modelos tradicionales como un perceptrón multicapa (*MLP*) en el contexto de predicción de aforos de tráfico.

Para ello, se ha diseñado una estrategia de evaluación comparativa entre modelos, aplicándolos sobre tres fuentes de datos independientes que recogen información de tráfico rodado en la provincia de Bizkaia. Cada fuente representa una infraestructura y cobertura distintas, lo que permite evaluar el comportamiento de los modelos bajo distintos escenarios y estructuras de datos.

Fuentes de datos

Las fuentes de datos empleadas se almacenan en la base de datos `mobility_db` de MongoDB, en la colección `mobility_snapshots`. Cada snapshot representa una observación de múltiples medidores de tráfico en una marca temporal concreta. Las fuentes son las siguientes:

- **source_id = 1:** Datos de la Dirección de Tráfico del Gobierno Vasco. Esta fuente ofrece una cobertura amplia a lo largo de la red viaria principal de la CAPV, incluyendo carreteras de alta capacidad.
- **source_id = 2:** Datos de la Diputación Foral de Bizkaia. Se trata de aforos situados principalmente en carreteras secundarias y forales.
- **source_id = 5:** Datos del Ayuntamiento de Bilbao. Esta fuente representa el entorno urbano por excelencia, con sensores instalados en intersecciones y calles principales de la ciudad.

La estructura de los datos ha sido estandarizada previamente a través de un pipeline de construcción de dataset explicado en capítulos anteriores. En todos los casos, los datos incluyen no

solo los conteos de vehículos sino también información meteorológica, atributos temporales (hora, día de la semana, etc.), y metadatos geográficos y semánticos de los sensores.

La Figura 6 muestra la ubicación de los sensores de la fuente `source_id = 1` (Gobierno Vasco). En muchos de estos puntos existen múltiples medidores, ya que se instala un sensor por carril en el mismo emplazamiento físico.

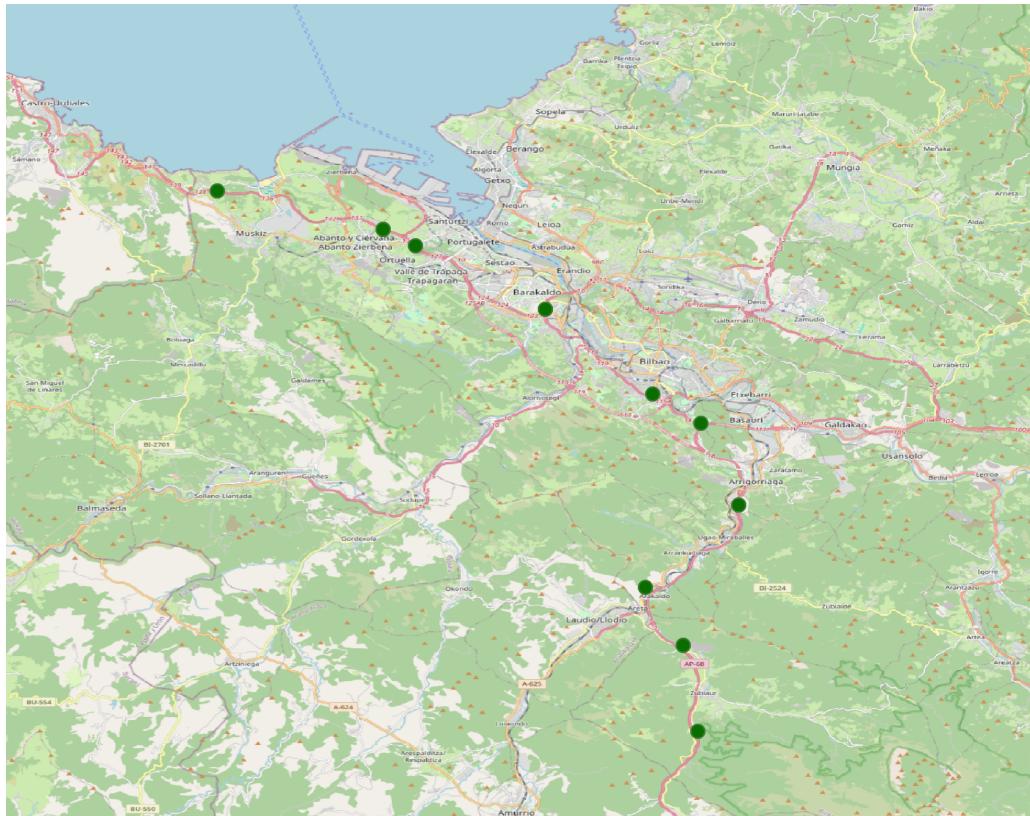


Figura 6: Ubicación de aforos de la fuente Gobierno Vasco (`source_id = 1`)

La Figura 7 recoge la distribución de medidores correspondiente a la fuente `source_id = 2` (Diputación Foral de Bizkaia). Esta fuente dispone de una extensa red de sensores desplegada a lo largo de la provincia, lo que proporciona una cobertura muy amplia y variada a nivel geográfico y funcional.

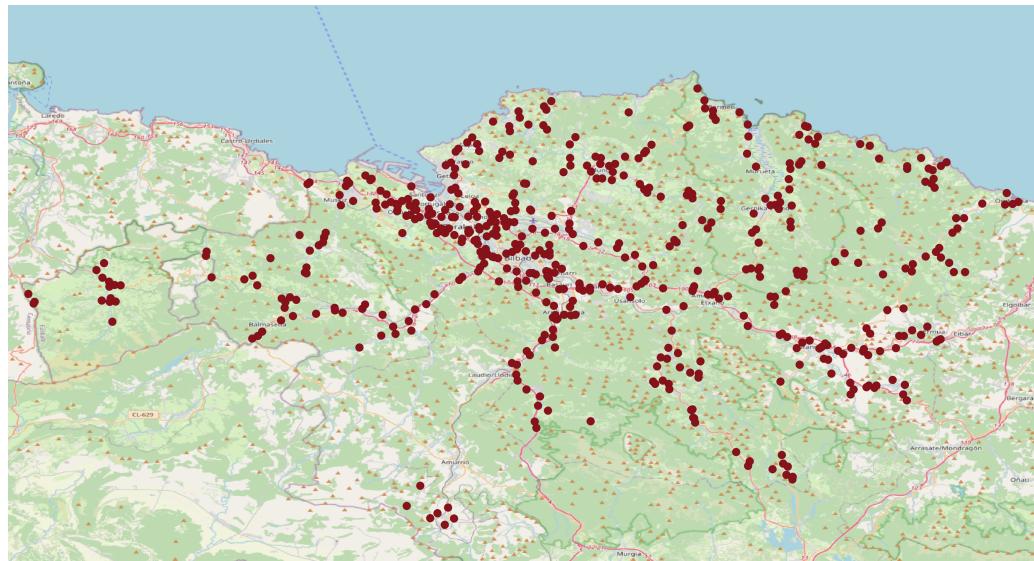


Figura 7: Ubicación de aforos de la fuente Diputación Foral de Bizkaia (source_id = 2)

Finalmente, la Figura 8 representa los sensores aportados por el source_id = 5 (Ayuntamiento de Bilbao), con presencia en los principales ejes viarios de la ciudad. Se conoce que existen más medidores instalados, pero el consistorio no ha habilitado su acceso público o no los tiene en funcionamiento, por razones que se desconocen.

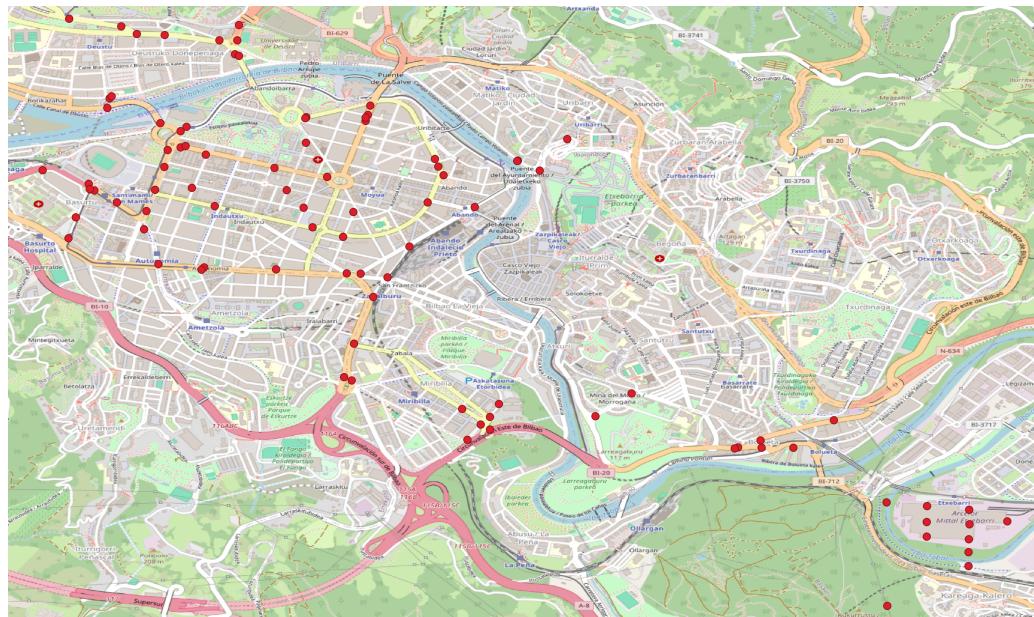


Figura 8: Ubicación de aforos de la fuente Ayuntamiento de Bilbao (source_id = 5)

Hipótesis y objetivos del experimento

El objetivo del experimento es doble:

1. **Evaluar el rendimiento comparativo** de dos modelos de predicción de tráfico: un modelo base (MLP) y un modelo avanzado (Trafficformer), bajo las mismas condiciones de entrenamiento y sobre las tres fuentes de datos.
2. **Analizar el impacto del diseño de arquitectura y la configuración de parámetros** sobre el rendimiento predictivo en escenarios heterogéneos, valorando especialmente la aportación del mecanismo de atención espacial.

La hipótesis de partida es que el modelo *Trafficformer*, al integrar mecanismos de atención multi-cabeza junto con máscaras espaciales basadas en OpenStreetMap, será capaz de capturar relaciones espaciales y temporales complejas entre sensores de tráfico, lo que se traducirá en mejores métricas de error respecto al MLP.

Antes de abordar las particularidades técnicas de cada arquitectura en apartados posteriores, en la Tabla 13 se presenta una comparación conceptual entre los dos modelos evaluados: un perceptrón multicapa clásico (MLP) y el modelo propuesto basado en atención (Trafficformer).

Tabla 13: Comparación conceptual entre los modelos MLP y Trafficformer

Característica	MLP (modelo base)	Trafficformer (modelo avanzado)
Tipo de arquitectura	Perceptrón multicapa clásico	Transformer con atención espacial enmascarada
Entrada esperada	Tensor vectorizado por muestra	Tensor estructurado por sensor y ventana temporal
Captura de relaciones espaciales	No	Sí, mediante mecanismos de atención enmascarada
Robustez ante ruido estructural	Limitada	Alta, gracias al diseño atencional con máscara espacial
Interpretabilidad	Alta (modelo simple)	Media (visualización de atención posible, pero más compleja)
Tiempo de entrenamiento	Bajo	Elevado
Requisitos computacionales	Moderados	Altos (uso de GPU y optimización de memoria)

Estrategia comparativa

La estrategia consiste en:

- Entrenar ambos modelos (MLP y Trafficformer) sobre cada `source_id`, utilizando combinaciones diversas de hiperparámetros.
- Evaluar los resultados sobre un conjunto de test independiente, tras aplicar *early stopping* sobre validación.
- Seleccionar el mejor modelo para cada combinación (`source_id`, modelo), basándose en las métricas MAE, RMSE y R^2 .

- Realizar comparativas cruzadas entre MLP y Trafficformer dentro de cada source, y globalmente.

Este enfoque permite no solo identificar el modelo óptimo por entorno de datos, sino también validar si la mejora obtenida por el modelo basado en atención es consistente y significativa. En apartados posteriores se detallarán las configuraciones evaluadas, el entorno de experimentación y los resultados obtenidos.

5.2 Entorno de desarrollo y reproducibilidad

Este apartado describe el conjunto de herramientas, tecnologías y decisiones que han permitido asegurar la trazabilidad, reproducibilidad y escalabilidad de los experimentos realizados en este trabajo. Se detallan tanto el entorno software empleado como las herramientas de seguimiento, exportación y despliegue previstas.

Entorno de desarrollo y gestión del proyecto

La experimentación ha sido desarrollada en un proyecto Python estructurado bajo el paquete `trafficformer-bizkaia`, gestionado mediante el gestor de dependencias `uv`. Este gestor ha demostrado ser una alternativa eficiente y moderna a sistemas tradicionales como `pip` o `poetry`, permitiendo la instalación aislada de entornos y una resolución rápida de paquetes.

El conjunto de dependencias se encuentra especificado en el fichero `pyproject.toml` de la Figura 9, y abarca bibliotecas clave para aprendizaje profundo (`torch`, `torchvision`), procesamiento de datos (`numpy`, `pandas`), manipulación geoespacial (`osmnx`, `pyrosm`) y conexión con bases de datos NoSQL (`pymongo`).

```

1  [project]
2    name = "trafficformer-bizkaia"
3    authors = [
4      {name = "Jon I", email = "jonin.sm@gmail.com"}
5    ]
6    version = "20250604"
7    description = "Short-term traffic prediction with Transformer models for Bizkaia road network"
8    readme = "README.md"
9    requires-python = ">=3.13"
10   dependencies = [
11     "python-dotenv==1.1.0",
12     "jupyter==1.1.1",
13     "numpy==2.2.6",
14     "pandas==2.2.3",
15     "pandas-stubs==2.2.3.250527",
16     "seaborn==0.13.2",
17     "matplotlib==3.10.3",
18     "scikit-learn==1.6.1",
19     "pymongo==4.13.1",
20     "torch>=2.7.1",
21     "torchaudio>=2.7.1",
22     "torchvision>=0.22.1",
23     "geopy==2.4.1",
24     "wandb==0.20.1",
25     "onnx==1.18.0",
26     "onnxscript==0.3.0",
27     "onnxruntime==1.22.0",
28     "folium==0.19.7",
29     "osmnx==2.0.4",
30     "pyrosm==0.6.2",
31     "tabulate==0.9.0"
32   ]
33
34   [tool.pytest.ini_options]
35   [torch]
36     # GPU
37     { index = "pytorch-cu128" },

```

Figura 9: Captura del fichero pyproject.toml

Monitorización y trazabilidad experimental con WANDB

La herramienta Weights & Biases (wandb) ha sido empleada como sistema de seguimiento y gestión de experimentos. Su integración ha permitido:

- Registrar cada ejecución con sus hiperparámetros, métricas y estructura del modelo.
- Visualizar en tiempo real curvas de entrenamiento, validación y test.
- Comparar rápidamente múltiples variantes experimentales.
- Recuperar el identificador del mejor modelo por combinación de parámetros.
- Exportar fácilmente los pesos y configuraciones más prometedoras.

Esta herramienta ha resultado esencial para asegurar la reproducibilidad de los experimentos y la trazabilidad científica, especialmente dado el elevado número de combinaciones exploradas (120 experimentos). En la Figura 10 se puede visualizar cuál es el aspecto visual de Wandb.

	Name	Tags	Created	Runtime	Sweep	batch_size	dropout	embed	epoch	ffnwdc	learning_rate	model	num_3	num_5	random	seq_le	adam_w
1	s011_mp_01	mp, salt	1w ago	1m 49s	-	32	-	0.2	-	100	-	Trafficformer	-	-	42	4	0.01
2	s011_mp_02	mp, salt	1w ago	1m 37s	-	64	-	0.2	-	100	-	Trafficformer	-	-	42	4	0.01
3	s011_mp_03	mp, salt	1w ago	1m 51s	-	32	-	0.2	-	100	-	Trafficformer	-	-	42	4	0.01
4	s011_mp_04	mp, salt	1w ago	1m 41s	-	64	-	0.2	-	100	-	Trafficformer	-	-	42	4	0.01
5	s011_mp_05	mp, salt	1w ago	1m 29s	-	32	-	0.2	-	100	-	Trafficformer	-	-	42	0	0.01
6	s011_mp_06	mp, salt	1w ago	1m 50s	-	64	-	0.2	-	100	-	Trafficformer	-	-	42	0	0.01
7	s011_mp_07	mp, salt	1w ago	3m 2s	-	32	-	0.2	-	100	-	Trafficformer	-	-	42	0	0.01
8	s011_mp_08	mp, salt	1w ago	2m 25s	-	64	-	0.2	-	100	-	Trafficformer	-	-	42	0	0.01
9	s011_mp_09	mp, salt	1w ago	3m 44s	-	32	9.5	0.2	64	120	256	Trafficformer	4	4	42	4	0.01
10	s011_mp_10	mp, salt	1w ago	6m 11s	-	32	9.5	0.2	64	120	512	Trafficformer	4	6	42	4	0.01
11	s011_mp_11	mp, salt	1w ago	7m 33s	-	32	9.5	0.2	128	120	256	Trafficformer	8	4	42	4	0.01
12	s011_mp_12	mp, salt	1w ago	3m 46s	-	32	9.5	0.2	128	120	512	Trafficformer	8	6	42	0	0.01
13	s011_mp_13	mp, salt	1w ago	3m 39s	-	64	9.5	0.2	64	120	256	Trafficformer	4	4	42	4	0.01
14	s011_mp_14	mp, salt	1w ago	7m 7s	-	64	9.5	0.2	64	120	512	Trafficformer	4	6	42	4	0.01
15	s011_mp_15	mp, salt	1w ago	2m 36s	-	64	9.5	0.2	128	120	256	Trafficformer	8	4	42	4	0.01
16	s011_mp_16	mp, salt	1w ago	5m 19s	-	64	9.5	0.2	128	120	512	Trafficformer	8	6	42	4	0.01
17	s011_mp_17	mp, salt	1w ago	3m 29s	-	32	9.5	0.2	64	120	256	Trafficformer	4	4	42	4	0.01
18	s011_mp_18	mp, salt	1w ago	5m	-	32	9.5	0.2	64	120	512	Trafficformer	4	6	42	4	0.01
19	s011_mp_19	mp, salt	1w ago	4m 23s	-	32	9.5	0.2	128	120	256	Trafficformer	8	4	42	4	0.01
20	s011_mp_20	mp, salt	1w ago	5m 49s	-	32	9.5	0.2	128	120	512	Trafficformer	8	6	42	4	0.01

Figura 10: Captura de Wandb

Exportación de modelos con ONNX

Para favorecer la portabilidad y posible despliegue en entornos de producción o evaluación externa, se ha integrado la exportación de modelos al formato ONNX (Open Neural Network Exchange).

Este estándar abierto permite:

- Guardar modelos entrenados en un formato independiente de PyTorch.
- Evaluar los modelos de forma rápida usando onnxruntime, sin necesidad de cargar el entorno completo de entrenamiento.
- Preparar la integración futura en dispositivos edge, servicios cloud u otras plataformas compatibles.

Los modelos óptimos extraídos (uno por pareja source/modelo) han sido exportados a este formato y almacenados de forma segura en un sistema local y remoto.

Preparación para entrenamiento en la nube

Aunque todos los experimentos del presente trabajo han sido ejecutados localmente, se ha diseñado y probado un entorno completo de entrenamiento en la nube sobre Amazon Web Services (AWS).

Mediante una plantilla en CloudFormation (fichero template.yaml), se automatiza el despliegue de:

- Una instancia g5.xlarge con GPU NVIDIA A10G (24 GB VRAM), compatible con CUDA 12.8.
- Un volumen persistente de 200 GB en /mnt/tfmdata.
- Una configuración automática de WireGuard, que conecta la instancia a una VPN personal y le permite acceder de forma segura a la base de datos MongoDB interna.
- Un entorno Python 3.13 preconfigurado en arranque, listo para ejecutar notebooks y scripts con PyTorch y aceleración GPU.

Esta infraestructura fue diseñada como alternativa escalable, con posibilidad de ser reutilizada en el futuro para entrenamientos de mayor duración o mayor carga de memoria.

5.2.0.1 Nota sobre seguridad:

toda la configuración se recupera desde un bucket S3 controlado mediante permisos específicos del rol IAM.

Con el objetivo de facilitar futuras ejecuciones en la nube, se ha preparado una plantilla de infraestructura como código utilizando AWS CloudFormation. Esta plantilla automatiza el despliegue de una instancia con GPU, volumen persistente y conexión segura mediante VPN a la base de datos local. Aunque no ha sido necesario su uso durante el desarrollo del presente trabajo, se ha documentado como valor añadido y se incluye en el Anexo C.

Almacenamiento auxiliar con MinIO

Como complemento al sistema principal de base de datos, se ha utilizado un servidor MinIO (compatibilidad S3) desplegado sobre un NAS local con TrueNAS Scale.

MinIO ha sido empleado para:

- Almacenar copias de seguridad de los modelos entrenados.
- Gestionar versiones de máscaras espaciales pesadas (por ejemplo, las generadas a partir de OpenStreetMap).
- Guardar mapas y estructuras temporales utilizadas durante el preprocesamiento y entrenamiento.

Esta solución ha permitido separar claramente el almacenamiento estructurado (MongoDB) del almacenamiento masivo de ficheros, facilitando tanto el backup como la reutilización eficiente de artefactos.

5.3 Preparación de datos y pipeline

La construcción del pipeline de datos ha sido un paso clave en este trabajo, permitiendo transformar observaciones crudas de tráfico en estructuras tensoriales aptas para el entrenamiento de modelos de aprendizaje profundo. Para ello, se ha implementado un sistema modular en Python, que distingue claramente entre las necesidades del modelo base (MLP) y las del modelo avanzado (Trafficformer), evitando así procesamientos innecesarios en cada caso.

Extracción desde base de datos

La información de entrada proviene de una base de datos MongoDB, concretamente de las colecciones `meters` (información estática de sensores) y `mobility_snapshots` (lecturas horarias del flujo de tráfico).

La clase `TrafficDataset` encapsula la lógica de extracción, conexión a MongoDB, limpieza y transformación. Esta clase implementa la interfaz de datasets de PyTorch, y permite configurar de forma flexible la longitud de ventana temporal, variables exógenas, y si se requiere o no aplicar estructura espacial. Esta última opción es clave para compartir la clase entre modelos como MLP (sin estructura espacial) y Trafficformer (con estructura espacial y máscara).

5.4 Preparación y tratamiento del dataset

La preparación de los datos es una de las fases más relevantes del proyecto, pues impacta directamente en la estabilidad del entrenamiento y la capacidad predictiva de los modelos. Esta responsabilidad recae sobre la clase `TrafficDataset`, que transforma los datos extraídos de MongoDB en estructuras de tensores de PyTorch aptas para su consumo por modelos de aprendizaje profundo.

Estructura temporal y ventana de entrada

Cada muestra de entrada se genera a partir de una **ventana temporal deslizante** de tamaño configurable, siendo el valor elegido durante los entrenamientos `SEQ_LEN = 24`. Esta ventana abarca las observaciones de las últimas 24 horas por sensor, lo que permite capturar patrones diarios recurrentes. Las muestras se organizan cronológicamente y se alinean por `meterId`, permitiendo la construcción de tensores tridimensionales de forma `[ventana, sensores, features]` en modelos estructurados como `Trafficformer`, o vectores planos en modelos como el `MLP`.

Tratamiento de variables y limpieza

Durante la carga y transformación de los datos, se aplican múltiples operaciones para garantizar su calidad y consistencia:

- **Eliminación de duplicados:** se eliminan entradas redundantes por `meterId` y marca temporal.
- **Imputación segura de NaNs:** los valores faltantes se tratan según el tipo de variable, evitando pérdidas de información o distorsiones estadísticas.
- **Tratamiento de variables numéricas:**
 - Se permite seleccionar entre múltiples estrategias de imputación: `mean`, `zero` o `ffill`.
 - Se normalizan mediante `StandardScaler` (opcionalmente `MinMaxScaler`), garantizando distribución centrada y varianza unitaria.

- **Tratamiento de variables booleanas:**

- Se convierten a valores float32 binarios (0.0 o 1.0).
- Los NaNs se imputan con valor 0.0.

- **Tratamiento de variables categóricas:**

- Se permite parametrizar como Ordinal o mediante OneHotEncoding.
- Para los entrenamientos realizados, se ha utilizado OneHotEncoder, ampliando el número de columnas por cada variable categórica.

Listado de variables utilizadas

A continuación se presenta la lista completa de variables utilizadas en el dataset, junto con su tipo y función:

- **sourceId:** categórica – origen de los datos (Gobierno Vasco, DFB, Bilbao).
- **meterId:** categórica – identificador del sensor de aforo.
- **startDateTime, endDateTime:** temporales – timestamps de inicio y fin de la medida.
- **weekday:** categórica – día de la semana (0-6).
- **hour, minute, end_hour:** numéricas – atributos horarios útiles para extraer patrones cíclicos.
- **totalVehicles:** numérica – cantidad de vehículos medidos, objetivo a predecir.
- **temperature, humidity, precipitation, pressure, solarRadiation, windSpeed:** numéricas – variables meteorológicas de entrada.
- **hasWeatherImpact, hasRoadwork, hasSafetyIssue, hasAccident, hasOtherIncident, hasEvent, hasCongestion:** booleanas – codifican la presencia de eventos o incidencias concurrentes.
- **latitude, longitude:** numéricas – ubicación del sensor, usadas solo en modelos estructurados.

Estas variables han sido seleccionadas por su relevancia en la predicción del flujo vehicular y su disponibilidad homogénea en todas las fuentes de datos integradas.

Persistencia y reutilización

Tras su preparación, los datos se serializan en disco en formato `.npz` (datos) y `.json` (metadatos de configuración). Esta estrategia ha permitido:

- Reutilizar conjuntos preparados en diferentes entrenamientos sin coste de regeneración.
- Validar consistencia entre entradas y configuración experimental.
- Integrar estos ficheros con entornos de entrenamiento remotos, como AWS, a través de MinIO.

Estructura de entrada por modelo

Para el modelo MLP, las muestras se representan como vectores planos: se concatena la información de todos los sensores y todas las variables dentro de una ventana temporal, sin conservar ninguna estructura espacial. Se omite completamente el uso de grafos o máscaras, priorizando la eficiencia y simplicidad.

Para el modelo Trafficformer, en cambio, se conserva una estructura tridimensional [ventana temporal, número de sensores, variables por sensor], permitiendo al modelo explotar tanto patrones temporales como relaciones espaciales mediante mecanismos de atención.

Esta diferenciación permite compartir código, reutilizando funciones comunes de preprocesado, pero manteniendo separadas las fases más costosas como la construcción de grafos o generación de máscaras.

Construcción del grafo de sensores

En el caso del modelo Trafficformer, se genera un grafo representando la red viaria de sensores. Cada nodo representa un `meter`, y las aristas se definen en función de su proximidad física o conectividad vial. Este grafo es generado desde el módulo `sensor_network_map.py`, utilizando distancias geográficas, heurísticas de agrupación y datos abiertos de OpenStreetMap.

A partir del grafo se puede derivar la matriz de adyacencia que servirá como base para aplicar enmascaramientos espaciales.

Máscaras espaciales (solo para Trafficformer)

Las máscaras espaciales permiten restringir el campo de atención del modelo, indicando qué sensores deben influenciarse entre sí durante el proceso de autoatención.

Se ha diseñado una arquitectura extensible de generación de máscaras, implementada en el paquete `mask`, con tres estrategias principales:

- `BinarySpatialMaskStrategy`: enmascaramiento binario según distancia máxima.
- `ContinuousSpatialMaskStrategy`: atenúa la atención según distancia continua.
- `OSMPathSpatialMaskStrategy`: estrategia principal utilizada, basada en conectividad vial real extraída de OpenStreetMap.

Estas máscaras se construyen en formato tensorial y se almacenan para su reutilización durante los entrenamientos posteriores.

Serialización y persistencia

Todos los tensores generados (entradas, salidas, máscaras) se serializan en disco utilizando formatos como `.npz` y `.json`. Estos ficheros se almacenan de forma organizada por combinación de modelo y fuente de datos, y se respaldan en MinIO para garantizar su disponibilidad en posteriores ejecuciones.

Esta estrategia de persistencia permite:

- Acelerar la experimentación al evitar preprocesado redundante.
- Asegurar reproducibilidad al asociar cada dataset a su configuración original.
- Compatibilidad con entornos remotos como AWS, donde se montan los datasets desde MinIO.

5.5 Descripción de los modelos

Durante el desarrollo de este trabajo se han implementado dos tipos de modelos de aprendizaje profundo para la predicción del flujo vehicular: un modelo base tipo perceptrón multicapa (MLP) y una arquitectura avanzada basada en Transformers denominada *Trafficformer*. Ambos modelos han sido implementados en PyTorch y diseñados para ajustarse al mismo pipeline de datos, permitiendo una comparación directa en términos de rendimiento y eficiencia.

Perceptrón Multicapa (MLP)

El modelo MLP, implementado en el fichero `mlp_enhanced.py`, actúa como modelo base para el conjunto de experimentos. Su arquitectura consiste en una serie de capas completamente conectadas (Linear), separadas por funciones de activación ReLU, normalización por lotes (BatchNorm1d) y capas de regularización Dropout.

Cada entrada al modelo consiste en un vector plano que concatena todas las variables relevantes para un conjunto de sensores a lo largo de una ventana temporal. No se incorpora ninguna estructura espacial, por lo que no hay noción de relaciones entre sensores.

El objetivo de este modelo es servir de referencia base o *baseline*, tanto en términos de complejidad como de rendimiento, permitiendo comparar sus resultados con arquitecturas más sofisticadas.

En la Figura 11 se presenta un esquema lógico de su flujo de capas.

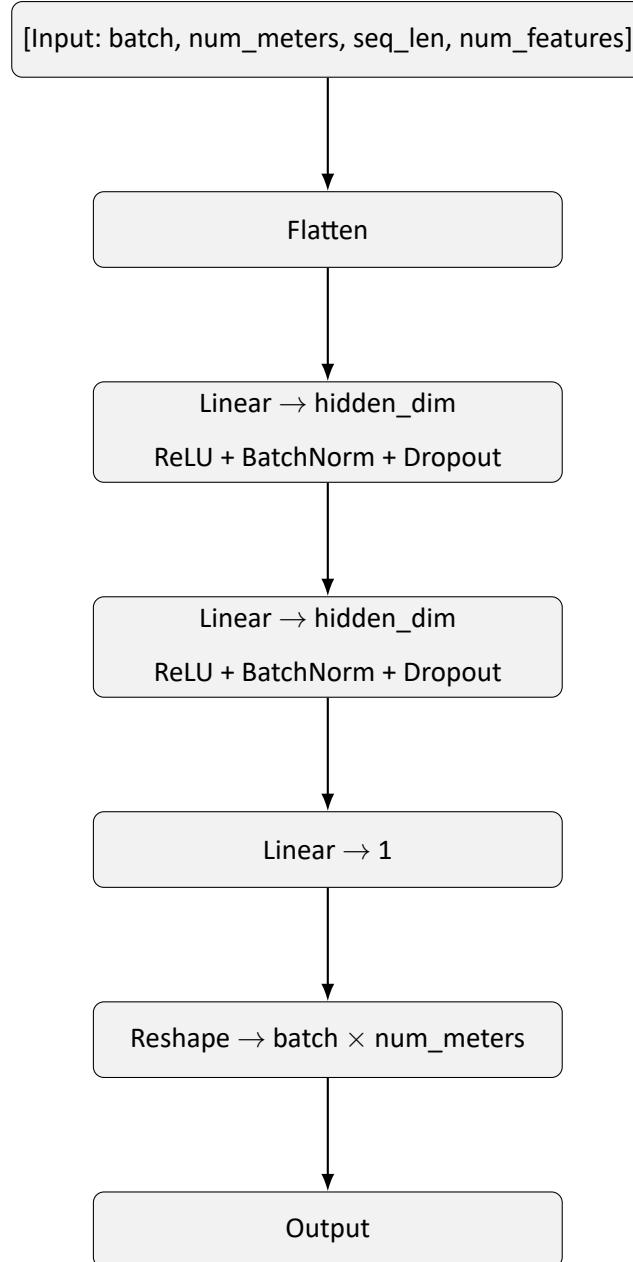


Figura 11: Esquema lógico del flujo de capas en el modelo MLP.

El comportamiento del modelo está condicionado por varios parámetros configurables en su constructor:

- `seq_len`: longitud de la ventana temporal. Determina el número de pasos hacia atrás que observa el modelo por sensor. Ejemplo: 8, para las últimas 4 horas.
- `num_features`: número de variables por paso temporal (meteorológicas, calendario, incidencias, etc.).
- `num_meters`: cantidad de sensores simultáneos a predecir en una sola muestra.

- `hidden_dim`: dimensión de las capas ocultas (**por defecto: 128**).
- `dropout`: probabilidad de desactivación aleatoria de neuronas durante el entrenamiento (**por defecto: 0.2**).

Estos parámetros permiten ajustar la capacidad del modelo a distintos tamaños de entrada y configuraciones experimentales, manteniendo una estructura modular y flexible.

Trafficformer

El modelo Trafficformer, implementado en el fichero `trafficformer.py` e incluido en el Anexo D, representa una evolución arquitectónica de los Transformers tradicionales adaptada al contexto específico de predicción de tráfico multivariable. A diferencia de las redes densas como el MLP, esta arquitectura se construye con el propósito de capturar de forma simultánea dependencias temporales a lo largo de una ventana de observación y relaciones espaciales entre sensores distribuidos por la red viaria.

Cada entrada al modelo es un tensor de cuatro dimensiones con forma `[batch, num_nodes, seq_len, num_features]`, donde cada nodo (o sensor) contiene una secuencia temporal de medidas multivariantes. A este tensor se le aplica una codificación posicional sinusoidal, que permite al modelo distinguir la posición relativa de cada instante de tiempo dentro de la secuencia, ya que carece de una estructura recurrente que imponga orden implícito.

Seguidamente, cada nodo procesa localmente su secuencia temporal a través de un extractor de características basado en un perceptrón multicapa (MLP), dotado de capas Linear, activations ReLU, normalización LayerNorm y Dropout. Esto da lugar a un embedding por nodo que encapsula su evolución temporal reciente.

Una vez obtenidos estos embeddings, se procede a su procesamiento mediante un bloque encoder compuesto por múltiples capas Transformer apiladas. Estas capas aplican mecanismos de atención multi-cabeza (MultiheadAttention) modificados con una máscara espacial. Esta máscara restringe selectivamente la atención entre pares de sensores en función de su proximidad geográfica o conectividad vial, impidiendo que el modelo derive patrones entre sensores inconexos o lejanos. Esta técnica es especialmente útil en entornos urbanos donde la correlación entre sensores es local y estructural. El uso de máscaras basadas en grafos viales —derivados, por ejemplo, de OpenStreetMap— permite guiar la atención del modelo hacia

relaciones físicamente plausibles.

Esta operación se ilustra esquemáticamente en la Figura 12, donde puede observarse cómo, para un nodo dado (por ejemplo, A), el mecanismo de atención restringe las relaciones posibles únicamente a aquellos nodos (como B o D) con los que mantiene una conexión válida según la máscara espacial predefinida. Los nodos no conectados —como C— quedan excluidos del proceso de atención mediante un enmascaramiento explícito. Este mecanismo refuerza la inductiva de la arquitectura, dotándola de una priorización estructural sobre la red vial que mejora la generalización y reduce la posibilidad de sobreajuste a correlaciones espurias.

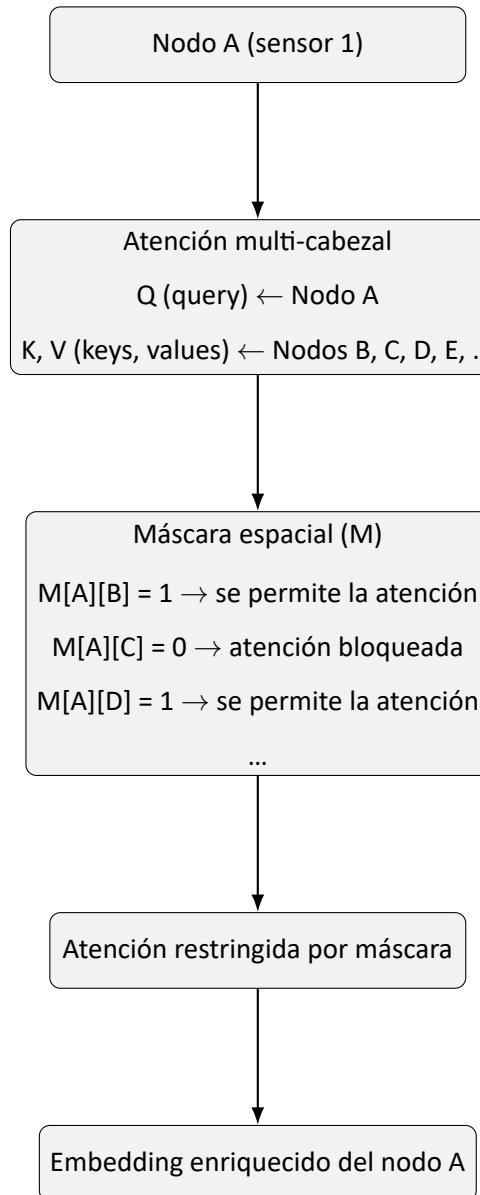


Figura 12: Aplicación de la máscara espacial en el mecanismo de atención de Trafficformer.

Tras el bloque de atención se encuentra una red FeedForward con normalización y conexiones

residuales, que complementa la capacidad de aprendizaje no lineal de la arquitectura. Finalmente, se aplica un bloque predictor por nodo, implementado como un MLP con estructura Linear → LayerNorm → ReLU → Linear, que emite la predicción del volumen de tráfico esperado en el próximo instante temporal.

El flujo completo de capas puede representarse de forma esquemática como sigue:

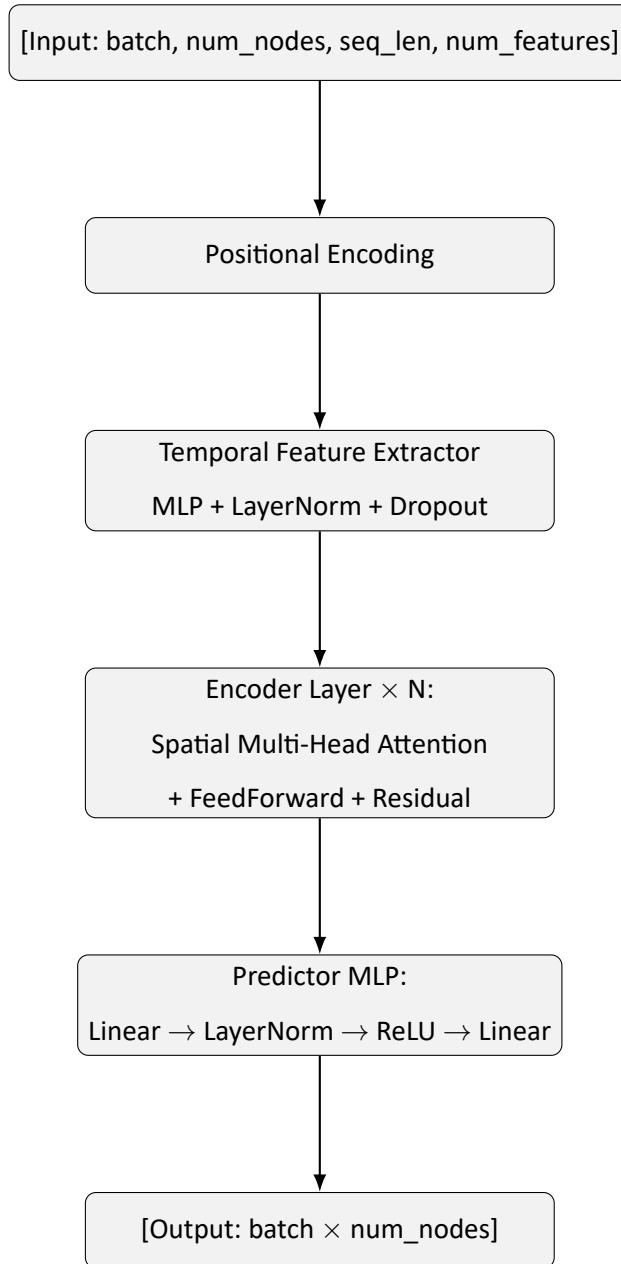


Figura 13: Esquema lógico del flujo de capas del modelo Trafficformer.

Cabe destacar que el modelo ha sido diseñado para ser altamente configurable, permitiendo ajustar su capacidad expresiva a distintos conjuntos de datos y requisitos computacionales.

Entre los principales hiperparámetros se encuentran:

- `seq_len`: número de pasos temporales observados por nodo.
- `num_features`: cantidad de variables por instante temporal.
- `embedding_dim`: dimensión del vector que representa cada nodo tras el extractor de características.
- `num_heads`: número de cabezales en el mecanismo de atención.
- `num_layers`: número de capas encoder apiladas.
- `ff_hidden_dim`: tamaño intermedio en la red feedforward de cada encoder.
- `dropout`: tasa de desactivación de unidades durante el entrenamiento.

Esta flexibilidad, unida a su capacidad para integrar estructura espacial, convierte a Trafficformer en una solución avanzada para la predicción de tráfico en entornos reales complejos, como el caso de estudio abordado en este trabajo.

5.6 Diseño experimental y combinaciones

Para evaluar de forma rigurosa el rendimiento de las distintas arquitecturas propuestas, se ha diseñado un conjunto exhaustivo de experimentos que cubre diversas combinaciones de modelos, hiperparámetros y fuentes de datos. El total de combinaciones generadas asciende a **120 experimentos**, cada uno de los cuales ha sido ejecutado de forma independiente y almacenado como notebook reproducible.

El diseño factorial considera dos modelos principales: MLP como base lineal y Trafficformer como modelo avanzado con atención espacial. A su vez, se han evaluado estas arquitecturas sobre tres conjuntos de datos procedentes de distintas fuentes institucionales, representadas por los identificadores `source_id = 1, 2, 5`.

Cada modelo ha sido entrenado utilizando un conjunto diverso de combinaciones de hiperparámetros. En la Tabla 15 se presentan los valores explorados para cada parámetro, clasificados por arquitectura.

Tabla 15: Configuraciones experimentales evaluadas

SID	M	SL	LR	BS	E/ES	NH	ED	NL/FF
1	MLP	4, 8	1e-3 / 5e-4	64 / 128	100 / patience=10	-	-	-
1	Trafficformer	4, 8	1e-3 / 5e-4	32 / 64	120 / patience=10	4 / 8	64 / 128	4/256 y 6/512
2	MLP	4, 8	1e-3 / 1e-4	64 / 128	100 / patience=10	-	-	-
2	Trafficformer	4, 8	1e-3 / 1e-4	32 / 64	120 / patience=10	4 / 8	64 / 128	4/256 y 6/512
5	MLP	4, 8	1e-3 / 5e-4	64 / 128	100 / patience=10	-	-	-
5	Trafficformer	4, 8	1e-3 / 5e-4	32 / 64	120 / patience=10	4 / 8	64 / 128	4/256 y 6/512

Leyenda de columnas: SID: sourceld.

M: Modelo.

SL: Secuencia temporal (seq_len).

LR: Learning Rate.

BS: Tamaño de batch.

E/ES: Épocas / Early Stopping (p: patience).

NH: Número de cabezas de atención (num_heads).

ED: Dimensión de embedding (embedding_dim).

NL/FF: Número de capas encoder / dimensión del feedforward (num_layers / ff_hidden_dim).

A continuación, se detallan las combinaciones de hiperparámetros evaluadas en el conjunto de experimentos realizados. Cada uno de estos parámetros ha sido cuidadosamente seleccionado en base a recomendaciones de la literatura científica y su aplicabilidad práctica en modelos de predicción multivariada de series temporales, como se describe seguidamente:

- **SEQ_LEN:** Los valores 4 y 8 permiten comparar el efecto de ventanas más cortas vs. más largas sobre la precisión y la capacidad de capturar patrones temporales. Es una práctica habitual en *forecasting* experimentar con varias ventanas para encontrar el equilibrio entre capacidad de predicción y sobreajuste.
- **Learning Rate:** 1e-3 es un valor estándar en *deep learning* y recomendado en el propio artículo de Trafficformer Chang et al. (2025), pero se elige 5e-4 para realizar el experimento. En el artículo también se cita el empleo de la estrategia ReduceLROnPlateau Ruder (2017), estrategia que permite reducir el learning rate hasta un umbral en función de la selección de una métrica. Pese a que dicha estrategia está disponible en Pytorch PyTorch (2025) se decidió no emplearla por reducir la complejidad del experimento.

- **Batch Size:** Se experimenta con 64 y 128 para MLP, y con 32 y 64 para Trafficformer. Un *batch size* menor en Trafficformer permite reducir el uso de memoria y mejora la generalización en modelos más profundos.
- **Epochs / Early Stopping:** Se ha limitado el entrenamiento a 100–120 épocas, con un mecanismo de *early stopping* con paciencia de 10 épocas. Esta configuración evita el sobreentrenamiento sin necesidad de vigilancia manual.
- **NUM_HEADS y EMBEDDING_DIM:** Las combinaciones evaluadas (4/64 y 8/128) reflejan el equilibrio entre capacidad representacional y coste computacional. Un mayor número de cabezas puede capturar relaciones más complejas entre sensores.
- **NUM_LAYERS / FF_HIDDEN_DIM:** Se han probado configuraciones de 4 capas con 256 dimensiones ocultas y 6 capas con 512. Estos valores están alineados con los recomendados en el artículo original y en benchmarks del estado del arte.

Cada experimento fue registrado y monitorizado utilizando la plataforma *Weights & Biases*, lo que permitió realizar un análisis sistemático posterior y facilitar la selección de los mejores modelos por fuente de datos y arquitectura. Para facilitar la reproducibilidad y la trazabilidad de los resultados, todos los notebooks están numerados de forma consistente e incluyen visualizaciones, métricas y configuración exacta de entrenamiento.

En el Anexo E se pueden consultar las diferentes combinaciones experimentales propuestas.

5.7 Proceso de entrenamiento y selección de mejores modelos

El proceso de entrenamiento de los modelos se ha realizado bajo un entorno controlado y reproducible, siguiendo las mejores prácticas en experimentación con redes neuronales. Para ello, se han desarrollado cuadernos Jupyter donde se define de manera explícita cada paso del ciclo experimental: carga de datos, configuración del entorno, definición del modelo, entrenamiento, evaluación y registro de resultados.

5.7.0.1 Configuración y reproducibilidad.

El entorno de trabajo se inicializa con la carga de variables de entorno desde ficheros .env, estableciendo parámetros como la conexión a la base de datos MongoDB, claves de autenticación para *Weights & Biases* (Wandb), y valores por defecto de hiperparámetros como batch

size, learning rate, dropout o hidden dimensions. Se fija una semilla aleatoria global para asegurar la reproducibilidad entre ejecuciones, controlando el estado de generación de números aleatorios tanto de NumPy, Python como de PyTorch (CPU y GPU).

5.7.0.2 Carga del dataset y preparación.

Se emplea una clase personalizada `TrafficDataset`, diseñada para conectarse directamente a la base de datos y construir un conjunto de muestras a partir de una ventana temporal configurable (`SEQ_LEN`). El conjunto de datos incluye tanto variables numéricas escaladas (`StandardScaler`), como categóricas codificadas (`OneHotEncoder`) y booleanas convertidas a float con valores binarios. Las variables faltantes se imputan de forma segura mediante media (`mean`) para variables continuas y con cero para booleanos. La división de datos se realiza de forma estratificada y reproducible en tres subconjuntos: entrenamiento (70 %), validación (20 %) y test (10 %).

5.7.0.3 Máscara espacial y visualización.

Para los modelos tipo Transformer, se genera una máscara espacial binaria que codifica qué sensores de tráfico pueden atenderse entre sí, basada en criterios geográficos extraídos desde OpenStreetMap (vía GraphML). Esta máscara se emplea durante el mecanismo de atención multi-cabezal. Se valida la consistencia entre los sensores del dataset y la máscara, y se genera una visualización interactiva de la red de sensores y sus enlaces espaciales, posteriormente registrada como artefacto en Wandb.

5.7.0.4 Inicialización del modelo y optimización.

El modelo se instancia según el tipo elegido (`TrafficMLPEnhanced` o `Trafficformer`), utilizando parámetros definidos al inicio del experimento. El criterio de pérdida utilizado es el error cuadrático medio (`MSELoss`), y el optimizador elegido es `AdamW`, con tasa de aprendizaje inicial y `weight decay` ajustables. El modelo y el criterio son registrados en Wandb, junto con los hiperparámetros y la configuración del experimento.

5.7.0.5 Entrenamiento supervisado y *early stopping*.

El bucle de entrenamiento se ejecuta durante un máximo de 100–120 épocas, con activación de un mecanismo de *early stopping* basado en la pérdida de validación con una paciencia de

10 épocas. Por cada época, se computan las métricas MSE, MAE, RMSE, MAPE y R^2 para los tres subconjuntos (train, val, test). Además, se monitoriza el uso de memoria RAM y GPU, el tiempo de inferencia por muestra y la duración total por época.

En cada iteración se guardan:

- Un *checkpoint* del estado actual del modelo, optimizador y estados aleatorios.
- La historia de métricas en un fichero CSV.
- El mejor modelo según pérdida de validación.

5.7.0.6 Evaluación y seguimiento.

Al finalizar el entrenamiento, se guarda el modelo final, se registra el tiempo total del proceso y se finaliza la sesión de Wandb. Además, se generan gráficas para cada métrica de evaluación, comparando la evolución entre entrenamiento y validación. La mejor época (mínima pérdida de validación) se resalta en cada gráfica. Estas curvas permiten detectar patrones de sobreentrenamiento, estancamiento o mejoras progresivas en el aprendizaje.

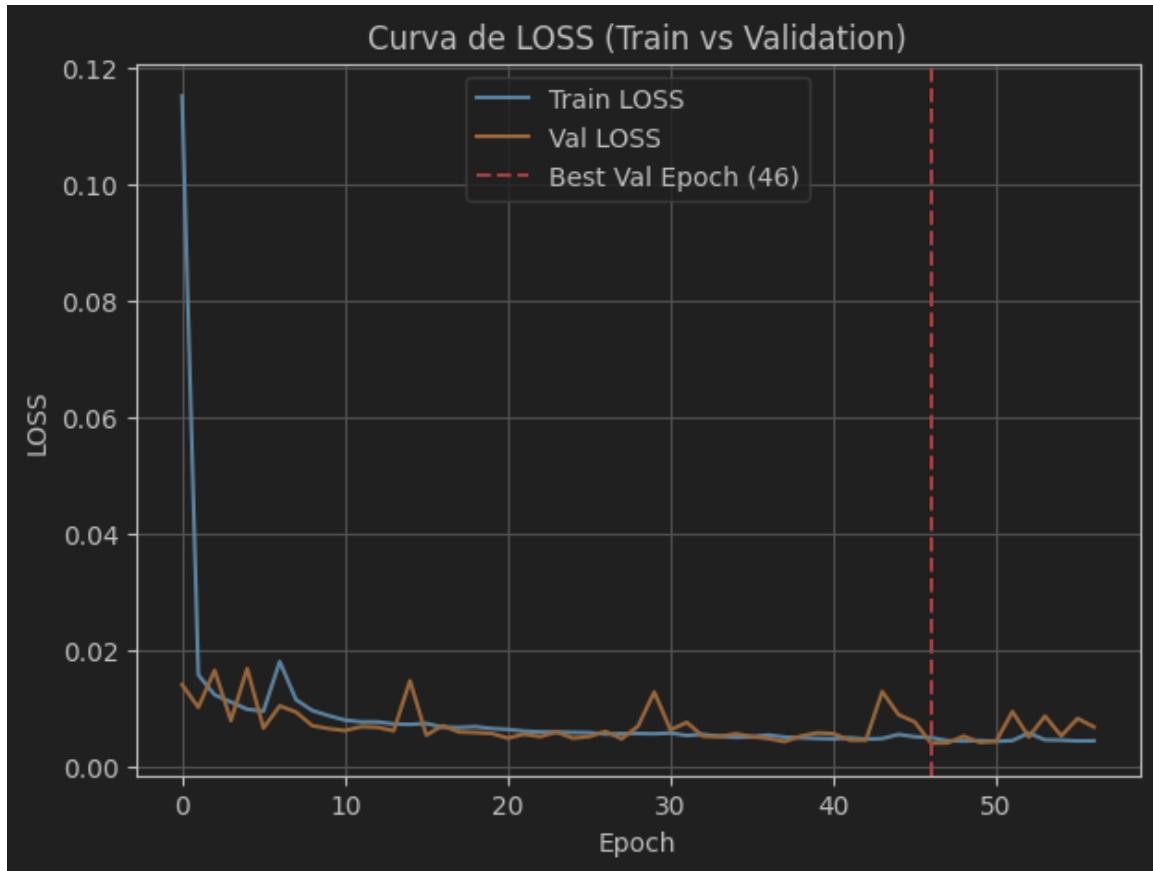


Figura 14: Curvas de evolución de pérdida para entrenamiento y validación, con indicación de la mejor época.

5.7.0.7 Registro de resultados y comparación.

Cada experimento queda documentado y reproducible mediante artefactos almacenados en el sistema de ficheros y sincronizados con la plataforma *Weights & Biases*. Esto incluye mapas de sensores, configuración exacta, métricas por época, curvas de entrenamiento y modelos entrenados. El historial completo permite comparar el rendimiento de diferentes combinaciones de hiperparámetros, tanto a nivel visual como mediante agregación tabular.

6 Evaluación, comparación de modelos y resultados

6.1 Resultados experimentales

Para cada una de las tres fuentes de datos disponibles (`sourceId = 1, 2 y 5`), se ha seleccionado el mejor modelo entrenado para cada una de las dos arquitecturas evaluadas: `MLP` y `Trafficformer`. La selección se ha realizado tomando como criterio el valor mínimo de la métrica de pérdida sobre el conjunto de validación (`val_loss`) entre todas las combinaciones de hiperparámetros evaluadas durante el proceso experimental.

En el Anexo F se puede consultar la tabla completa de todos los experimentos llevados a cabo junto con los resultados de todas las métricas obtenidas.

En la tabla 17 se muestran los modelos seleccionados junto con sus respectivas métricas sobre el conjunto de test, así como los valores clave de los hiperparámetros empleados.

Tabla 17: Resultados de los mejores modelos por combinación `sourceId`-arquitectura

SID	M	EP	VTL	VTMAE	VTRMSE	VTMAPE	VTR2
1	mlp	67	0.000	25.327/25.466	50.904/38.489	1.826/1.853	0.743/0.759
1	trafficformer	30	0.143	19.300/19.125	44.298/30.784	1.718/1.745	0.797/0.822
2	mlp	30	0.000	30.967/31.017	38.144/38.177	1.75e14/1.82e14	0.868/0.866
2	trafficformer	96	0.002	5.787/5.832	15.356/15.046	1.31e3/1.51e3	0.983/0.982
5	mlp	39	0.000	216.265/213.850	345.766/343.603	4.85e5/4.83e5	0.523/-1.797
5	trafficformer	29	0.153	176.356/175.975	308.041/306.134	1.74e5/1.74e5	0.703/0.640

Leyenda de columnas:

SID: `sourceId`.

M: Modelo.

EP: Epoch óptima.

VTL: Validation & Test Loss.

VTMAE: Validation & Test MAE.

VTRMSE: Validation & Test RMSE.

VTMAPE: Validation & Test MAPE.

VTR2: Validation & Test R^2 .

Los valores muy grandes se muestran en notación científica (a.eb significa $a \times 10^b$). Decimales reducidos para mejorar la presentación.

A continuación se muestran, para cada una de las combinaciones óptimas de fuente de datos (`sourceId`) y arquitectura (MLP y `Trafficformer`), las curvas de evolución de las principales métricas durante el entrenamiento. Estas gráficas permiten visualizar el comportamiento del proceso de aprendizaje en términos de error (`loss`), precisión (MAE, MAPE, MSE, RMSE), y capacidad explicativa (R^2), tanto en entrenamiento como en validación. De este modo, se facilita la identificación de fenómenos de sobreajuste, convergencia y diferencias entre arquitecturas y datasets.

Cada figura agrupa, en formato compacto, las seis métricas principales para cada modelo, mostrando la evolución época a época.

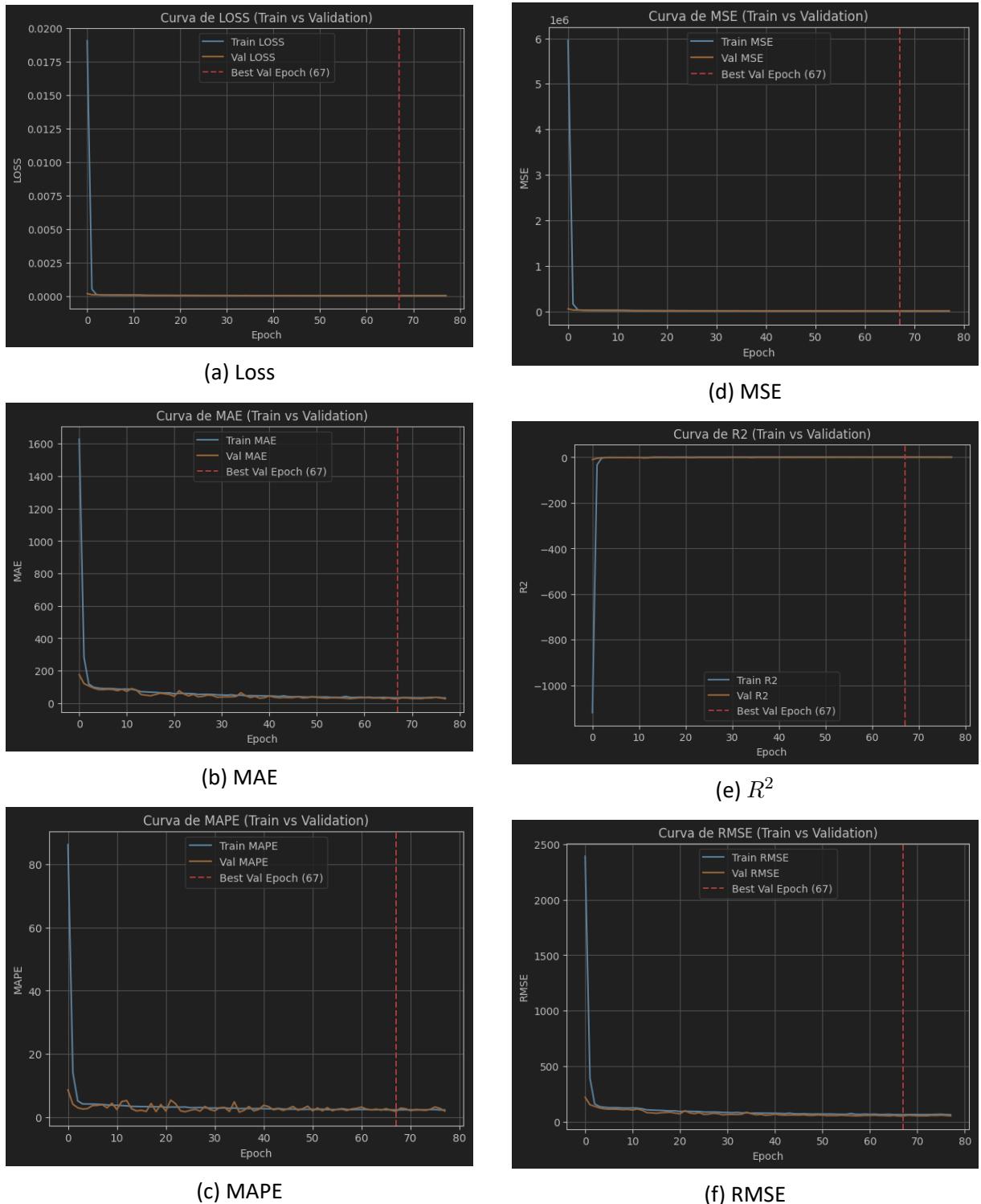


Figura 15: Curvas de entrenamiento para el modelo MLP con sourceld 1.

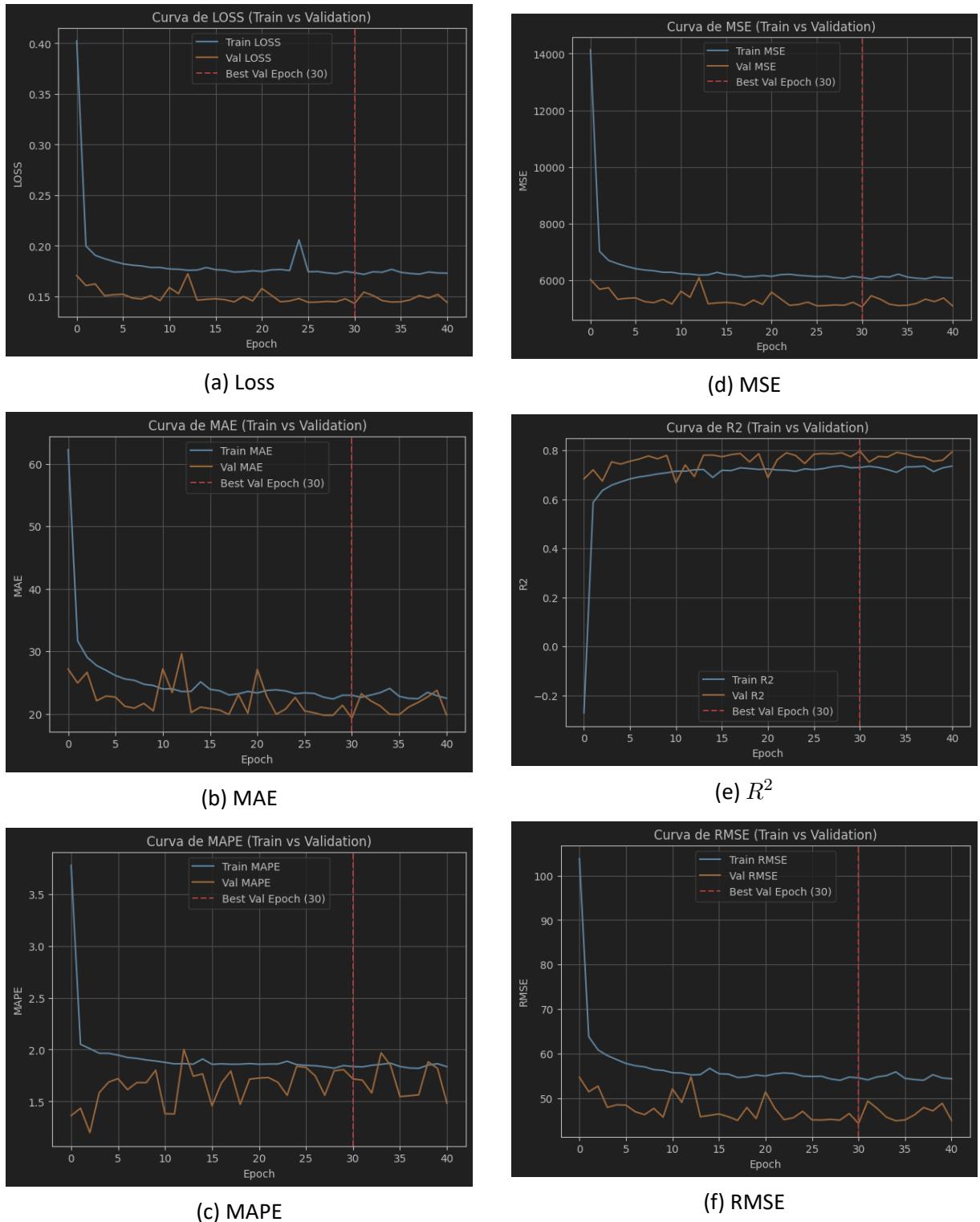


Figura 16: Curvas de entrenamiento para el modelo Trafficformer con sourceld 1.

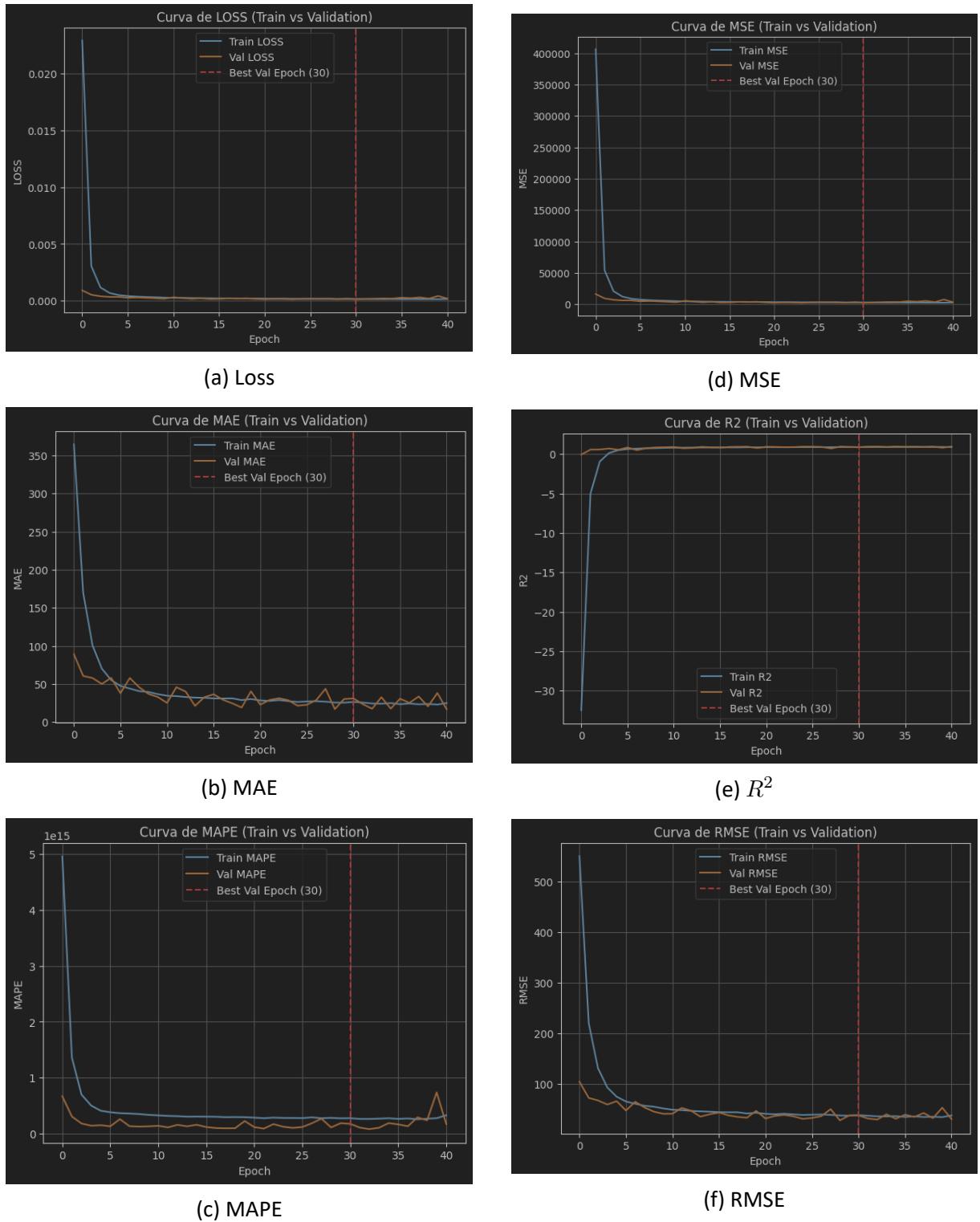


Figura 17: Curvas de entrenamiento para el modelo MLP con sourceId 2.

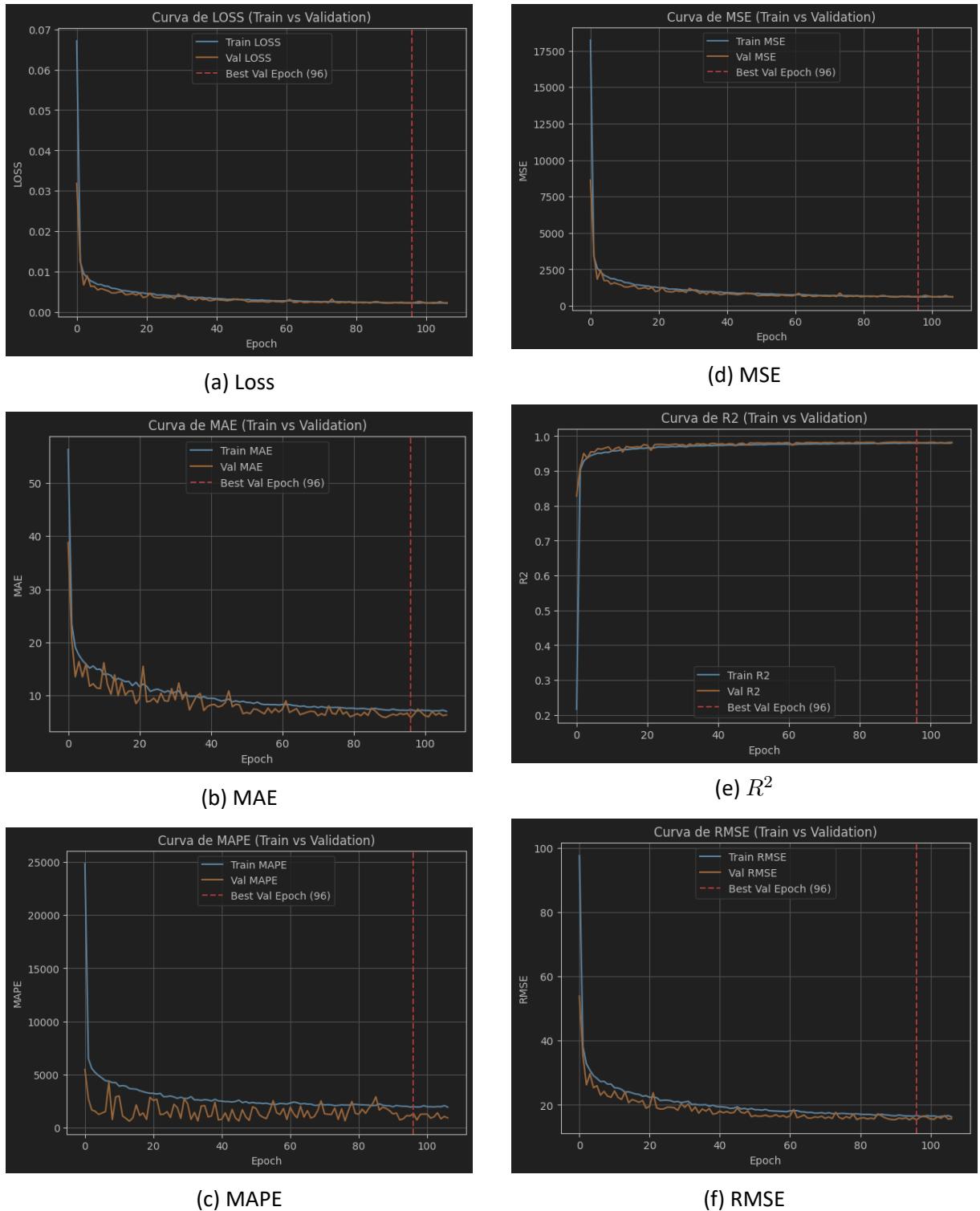


Figura 18: Curvas de entrenamiento para el modelo Trafficformer con sourceld 2.

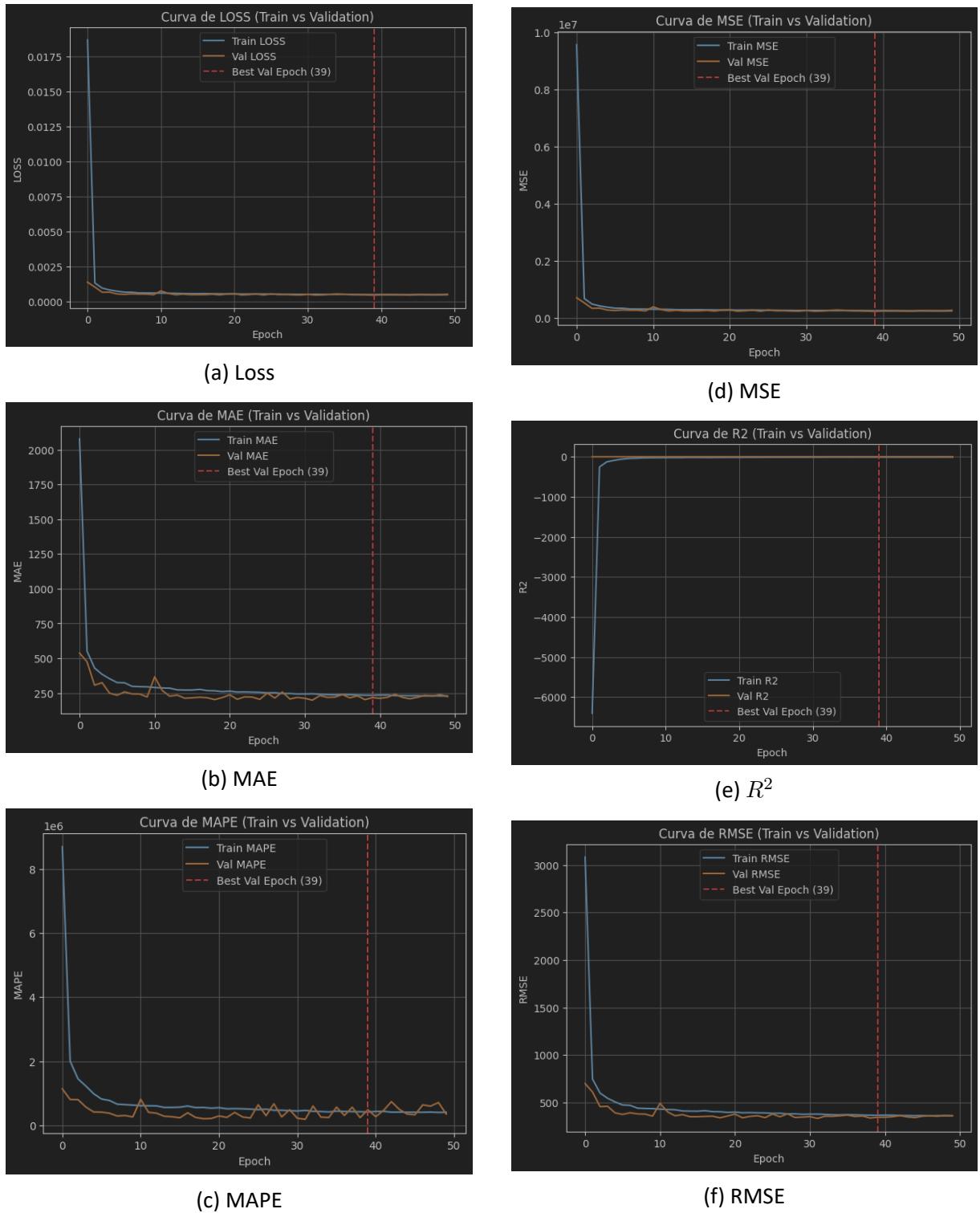


Figura 19: Curvas de entrenamiento para el modelo MLP con sourceId 5.

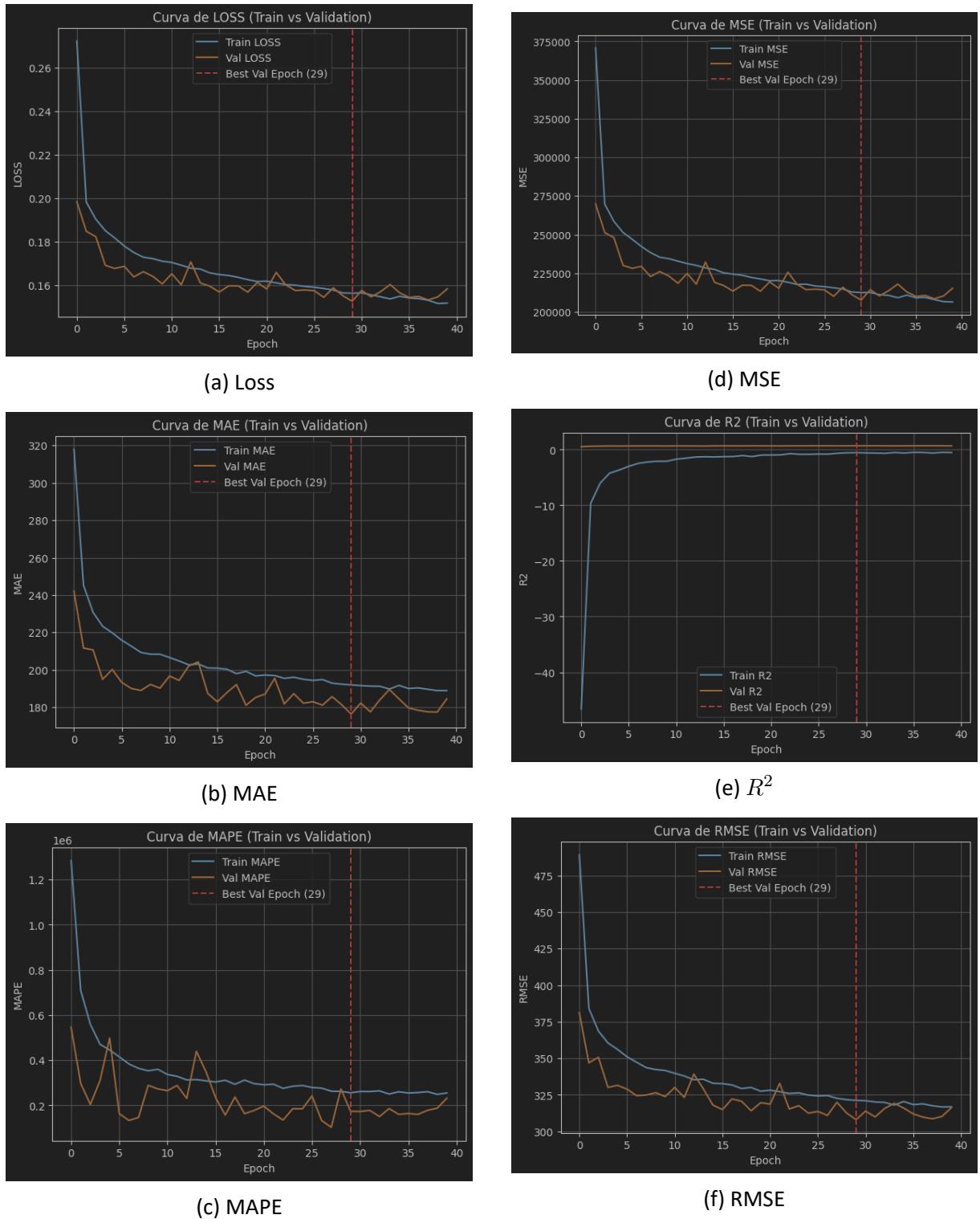


Figura 20: Curvas de entrenamiento para el modelo Trafficformer con sourceld 5.

Todas las curvas y métricas detalladas para los 120 experimentos están disponibles en la plataforma *Weights & Biases*.

6.2 Discusión y análisis crítico

En esta sección se realiza una evaluación crítica y comparativa de los resultados obtenidos por las dos arquitecturas propuestas (MLP y Trafficformer), para cada una de las tres fuentes de datos analizadas (sourcelds 1, 2 y 5). Se emplean las métricas reportadas en la tabla 17, junto con las gráficas de entrenamiento para cada modelo, para ofrecer un análisis detallado sobre la superioridad y limitaciones observadas.

Comparativa en sourceld 1

La comparativa entre modelos MLP (figura 15) y Trafficformer (figura 16) muestra claramente la ventaja del modelo Trafficformer. Esta arquitectura obtiene un valor significativamente menor en las métricas de pérdida (loss), MAE, RMSE y MAPE, así como un valor superior en R^2 . En particular, Trafficformer logra reducir el MAE desde 25.466 hasta 19.125 y el RMSE desde 38.489 hasta 30.784, mostrando una mejor capacidad para capturar las complejidades espaciales y temporales de los datos de tráfico.

El análisis visual de las curvas de entrenamiento evidencia una convergencia más rápida y estable del modelo Trafficformer, alcanzando el punto óptimo en la época 30, notablemente antes que el modelo MLP (época 67). Este fenómeno puede atribuirse a su capacidad de atención espacial y aprendizaje secuencial, permitiendo explotar eficazmente las relaciones entre sensores cercanos.

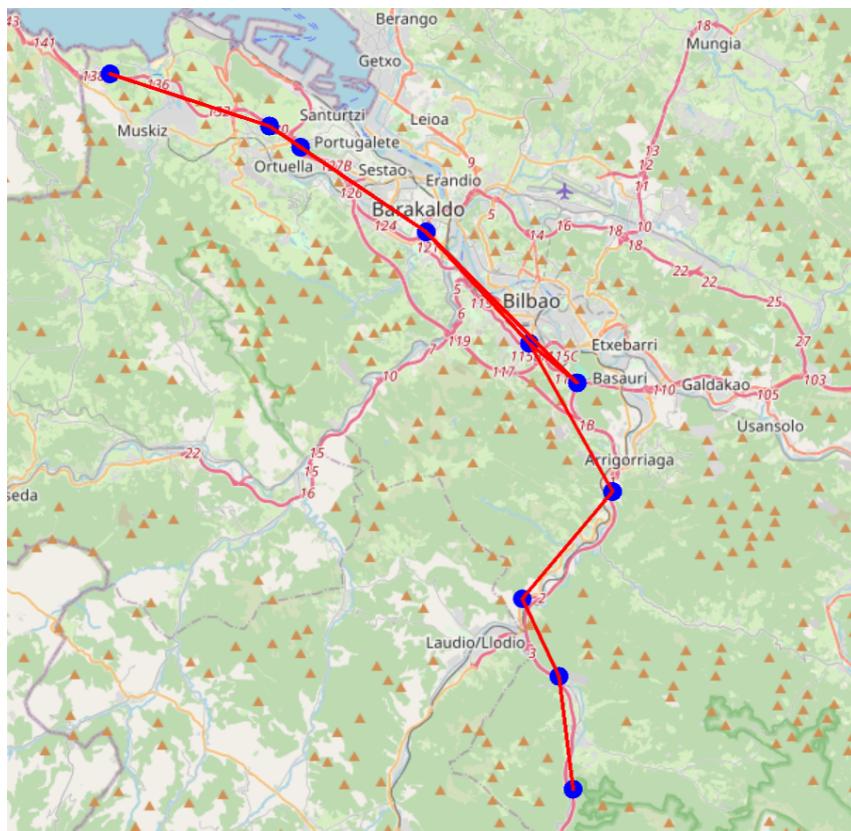


Figura 21: Distribución espacial de sensores para Sourceld 1

La disposición de los sensores del Sourceld 1, agrupados en puntos estratégicos específicos a lo largo de vías principales como se ve en la figura 21, permite al modelo Trafficformer aprovechar mejor las relaciones espaciales locales, favoreciendo su desempeño frente al modelo MLP.

Comparativa en sourceld 2

La superioridad del modelo Trafficformer respecto al modelo MLP se acentúa aún más en los datos del sourceld 2. Como se observa en las figuras 17 y 18, Trafficformer reduce considerablemente el MAE desde 31.017 hasta 5.832 y el RMSE desde 38.177 hasta 15.046. El R^2 mejora sustancialmente de 0.866 (MLP) hasta 0.982 (Trafficformer), indicando una capacidad notablemente superior para explicar la variabilidad en los datos.

Las curvas de entrenamiento muestran una convergencia estable y continua del modelo Trafficformer hasta la época 96, reflejando una adecuada selección de hiperparámetros que permitió una optimización profunda. El modelo MLP, en cambio, converge rápidamente en la época 30, mostrando potencialmente limitaciones en su capacidad para aprovechar plenamente el volumen y complejidad de los datos disponibles.

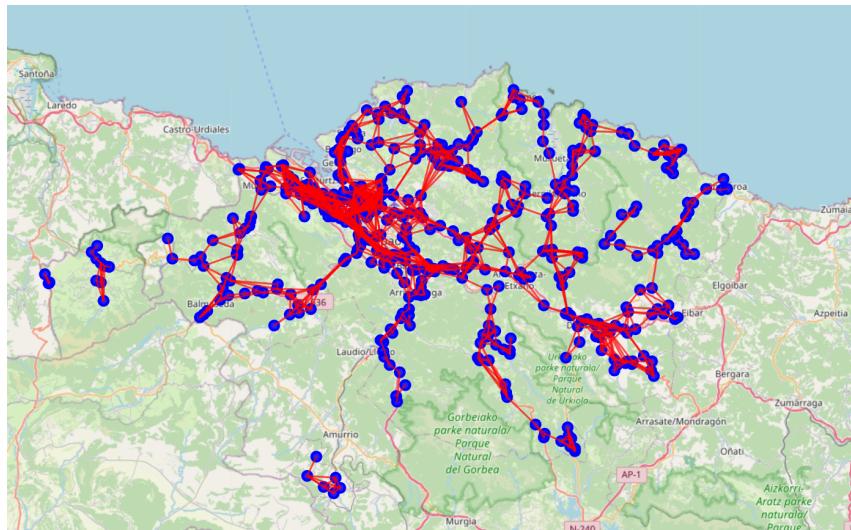


Figura 22: Distribución espacial de sensores para Sourceld 2

La amplia distribución geográfica de los sensores para Sourceld 2, como se aprecia en la figura 22, permite al modelo Trafficformer capturar relaciones espaciales complejas, aspecto que el modelo MLP no puede explotar debido a su limitada capacidad para modelar dependencias espaciales.

Comparativa en sourceld 5

Para la fuente sourceld 5, el modelo Trafficformer sigue mostrando una clara ventaja frente al modelo MLP. Según las figuras 19 y 20, se observa una mejora en todas las métricas principales. El MAE disminuye desde 213.850 hasta 175.975, y el RMSE desde 343.603 hasta 306.134. El valor R^2 presenta una mejora significativa, desde un negativo e inadecuado -1.797 hasta un aceptable 0.640 para Trafficformer.

Las curvas de entrenamiento revelan que Trafficformer converge rápidamente (época 29), mostrando que el mecanismo de atención espacial resulta particularmente efectivo en escenarios urbanos densos como el representado en esta fuente de datos.

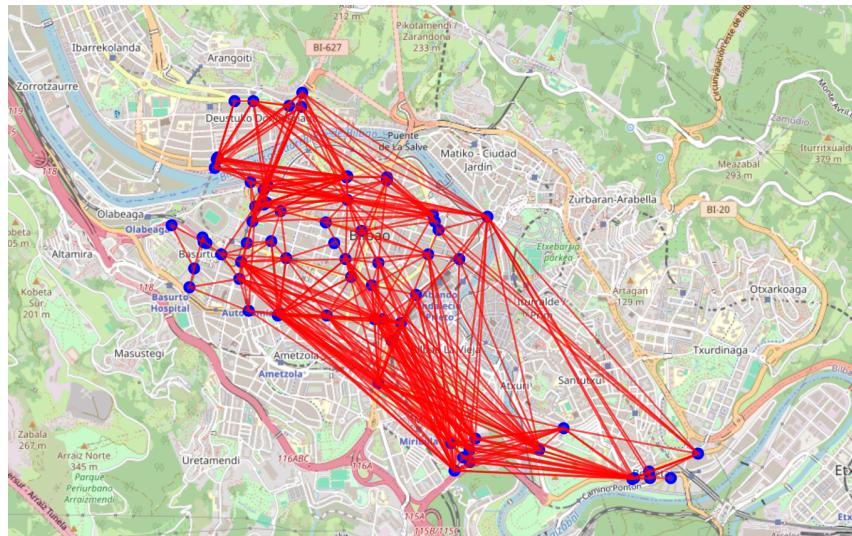


Figura 23: Distribución espacial de sensores para SourceID 5

La concentración urbana y la alta densidad espacial del SourceID 5, apreciable en la figura 23, generan complejas interacciones entre sensores cercanos. Esta configuración beneficia notablemente al modelo Trafficformer frente al MLP, que carece de mecanismos específicos para gestionar esta complejidad espacial.

Análisis comparativo de fuentes de datos

Las características particulares de cada fuente de datos han ejercido una influencia importante en el desempeño de los modelos.

6.2.0.1 SourceID 1

: Este conjunto posee un total de 386 sensores (meters), aunque estos se encuentran ubicados en unos pocos puntos estratégicos distribuidos linealmente a lo largo de vías principales, como se observa en la figura 21. Dado que cada sensor representa un carril individual, existe una alta densidad espacial localizada en ciertas áreas críticas, pero una baja distribución geográfica. El dataset utilizado para el entrenamiento tiene una dimensionalidad de entrada de 14,828 ventanas temporales, con una longitud de secuencia (`seq_len`) de 8 (ventanas de 4 horas). El tamaño del conjunto de entrenamiento es de 10,379, validación 2,980, y test 1,469. Esta configuración temporal amplia permite al modelo captar patrones dinámicos extendidos en el tiempo, siendo favorable para el mecanismo de atención espacial del modelo Trafficformer.

6.2.0.2 Sourceld 2

: La fuente de datos 2 presenta una distribución geográfica amplia y diversa, cubriendo exhaustivamente la red viaria de Bizkaia (figura 22). Inicialmente se consideraron 520 sensores, aunque tras el filtrado por ausencia de datos, el número efectivo de sensores fue de 462. Esto implica una alta complejidad en términos de relaciones espaciales y diversidad de patrones de tráfico. El conjunto de entrenamiento está compuesto por 12,292 ventanas temporales, mientras que validación y test contienen 3,530 y 1,739 ventanas respectivamente. Para esta fuente se ha empleado una ventana temporal más reducida ($\text{seq_len} = 4$, equivalentes a ventanas de 2 horas), debido a la mayor densidad y complejidad espacial. La máscara espacial generada revela numerosas conexiones entre sensores distantes, lo que beneficia significativamente al modelo Trafficformer gracias a su capacidad de capturar relaciones espaciales complejas mediante atención multi-cabezal.

6.2.0.3 Sourceld 5

: El tercer conjunto se caracteriza por una cobertura urbana densa, centrado en la ciudad de Bilbao (figura 23). Aunque inicialmente existían 92 sensores, el filtrado redujo el número a 70. La distribución espacial en un entorno urbano tan concentrado genera relaciones espaciales complejas y altamente correlacionadas. El dataset dispone de 17,330 ventanas temporales, dividido en entrenamiento (12,131 ventanas), validación (3,483 ventanas) y test (1,716 ventanas). También se empleó una ventana de 2 horas ($\text{seq_len} = 4$), facilitando al modelo capturar dinámicas urbanas rápidas. La máscara espacial evidencia la compleja red de interacciones espaciales entre sensores urbanos cercanos, circunstancia que nuevamente favorece al modelo Trafficformer debido a su capacidad innata para modelar dependencias espaciales y temporales simultáneamente.

Este análisis de fuentes de datos subraya la importancia del diseño experimental y la selección de ventanas temporales adecuadas a las particularidades de cada conjunto, reflejándose directamente en el rendimiento relativo de los modelos evaluados.

Síntesis y reflexión general

La superioridad generalizada del modelo Trafficformer sobre MLP en las tres fuentes de datos analizadas se explica principalmente por la capacidad de capturar dependencias espaciales

y temporales mediante el mecanismo de atención multi-cabezal. Esta capacidad es especialmente valiosa en contextos urbanos complejos (sourceId 5) y en redes de sensores extensas (sourceId 2), donde las relaciones entre los puntos de medida tienen gran relevancia.

La elección de hiperparámetros, como el tamaño del batch, learning rate, número de cabezas de atención, y dimensión de embedding, ha demostrado ser crítica en la optimización de Trafficformer. Los resultados muestran que valores intermedios o altos para `num_heads` y `embedding_dim` mejoran significativamente el rendimiento, validando lo observado en estudios previos del estado del arte (Chang et al., 2025).

Finalmente, el análisis pone de manifiesto que el modelo MLP presenta limitaciones inherentes a su arquitectura más simple, especialmente en contextos con fuertes correlaciones espaciales o temporales. El modelo Trafficformer, al incluir mecanismos avanzados de atención, proporciona mayor robustez y capacidad predictiva, alineándose con las tendencias recientes en la literatura científica sobre modelos Transformer aplicados a predicción del tráfico.

6.3 Análisis avanzado de resultados por fuente de datos

Además del análisis comparativo entre las arquitecturas y fuentes de datos realizado previamente, se ha llevado a cabo un análisis avanzado, complementario y exhaustivo, con el objetivo de identificar patrones específicos y posibles limitaciones de los modelos entrenados. Este análisis incluye gráficos específicos tales como la representación del error absoluto frente a los valores reales, series temporales comparativas para sensores individuales, mapas de errores y gráficos de distribución del error por sensor.

A continuación, se presentan las conclusiones más relevantes derivadas del análisis avanzado realizado para cada una de las fuentes de datos (sourceId). El detalle completo de las gráficas mencionadas está disponible en el Anexo G.

6.3.0.1 SourceId 1

: El análisis avanzado sobre la fuente de datos 1 muestra una adecuada capacidad predictiva global del modelo Trafficformer. El gráfico de dispersión (scatter plot) revela una fuerte correlación entre los valores predichos y los reales, con la mayoría de puntos cercanos a la diagonal. Sin embargo, el histograma de errores refleja la presencia de errores significativos en ciertas predicciones puntuales, lo que sugiere áreas específicas donde el modelo podría mejorar. El

mapa de errores confirma visualmente que la mayoría de errores altos se concentran en puntos geográficos específicos (probablemente zonas críticas de tráfico o intersecciones complejas), lo que podría ser causado por factores no capturados completamente por el modelo.

6.3.0.2 Sourceld 2

: En el caso del sourceld 2, se observa un excelente rendimiento del modelo Trafficformer, especialmente notable en el gráfico de dispersión y el histograma de errores, que muestran una alta concentración alrededor del valor real y errores generalmente bajos. Sin embargo, la distribución espacial del error representada en el mapa destaca algunas ubicaciones específicas con errores ligeramente mayores, posiblemente relacionadas con zonas periféricas o sensores más aislados, donde la densidad de datos disponibles para el entrenamiento fue inferior. Las series temporales para los mejores y peores sensores confirman que los mayores errores ocurren en contextos específicos, probablemente vinculados a eventos atípicos o patrones de tráfico inusuales.

6.3.0.3 Sourceld 5

: Finalmente, la fuente de datos 5, que cubre una zona urbana densa (Bilbao), presenta una complejidad intrínseca mayor. Esto se evidencia en el gráfico de dispersión, donde se observa una mayor dispersión de los puntos respecto a la diagonal ideal, indicando predicciones menos precisas para determinados sensores urbanos. El histograma del error muestra una distribución más amplia, sugiriendo heterogeneidad en la capacidad predictiva del modelo según zonas específicas. La distribución espacial de los errores confirma que áreas urbanas densamente pobladas y congestionadas muestran consistentemente errores más elevados, un aspecto esperado dada la complejidad del tráfico urbano.

Este análisis avanzado permite concluir que, aunque el modelo Trafficformer muestra en general buenos resultados en todas las fuentes de datos, existe un margen significativo de mejora en escenarios específicos y complejos, tales como intersecciones críticas y áreas urbanas densas. Además, pone de manifiesto la importancia del análisis visual detallado para identificar oportunidades específicas de optimización en futuras iteraciones del modelo.

Las gráficas completas utilizadas para este análisis avanzado se encuentran detalladamente en el Anexo G.

6.4 Limitaciones y validez de los experimentos

Aunque los resultados obtenidos a lo largo de este trabajo demuestran una alta eficacia en la predicción del tráfico mediante las arquitecturas propuestas, es fundamental reconocer ciertas limitaciones inherentes a la investigación realizada, que deben considerarse al interpretar los resultados obtenidos y planificar futuras líneas de trabajo.

En primer lugar, existe una **limitación asociada al tamaño y representatividad de las muestras**. Aunque se han empleado datasets significativos con numerosos sensores distribuidos espacialmente, ciertas fuentes de datos (como el caso del sourceId 1) presentan una distribución espacial concentrada en pocos puntos estratégicos. Esto podría implicar una representación parcial del comportamiento general del tráfico en áreas menos monitorizadas o rutas secundarias.

En segundo lugar, los experimentos se han visto afectados por ciertas **restricciones técnicas y de hardware**. La necesidad de entrenar múltiples modelos complejos como Trafficformer, con elevados requerimientos computacionales, ha limitado la exploración exhaustiva del espacio de hiperparámetros, especialmente en lo relativo al número de capas, cabezas de atención o tamaño de embedding. Esta limitación técnica puede haber impedido obtener resultados aún más óptimos.

Otra limitación importante reside en la **posible presencia de sesgos en los datos originales**. Los datasets provienen de fuentes públicas (Gobierno Vasco, Diputación Foral de Bizkaia y Ayuntamiento de Bilbao), por lo que es posible que contengan sesgos derivados de errores instrumentales, pérdida de datos en algunos sensores o inconsistencias temporales en la recopilación de información. Estos sesgos pueden afectar la precisión y generalización de los modelos desarrollados.

Adicionalmente, el uso exclusivo de **variables numéricas y categóricas predefinidas** limita la capacidad del modelo para captar factores externos relevantes, como eventos específicos no registrados, cambios estacionales detallados o efectos socioeconómicos más amplios, que podrían mejorar la precisión y robustez del modelo.

Por último, la **validez externa y generalización** de los resultados está condicionada al contexto específico de la provincia de Bizkaia. Aunque la metodología aplicada es escalable y transferible a otros contextos urbanos, es necesario realizar experimentos adicionales para confirmar que

los modelos desarrollados mantienen su rendimiento en otras regiones con diferentes características geográficas, culturales y económicas.

Estas limitaciones deben ser tomadas como puntos de partida para futuras investigaciones, enfocadas hacia la mejora de los modelos y la inclusión de nuevas fuentes de datos y técnicas que puedan aumentar la robustez y generalización del sistema propuesto.

7 Conclusiones y trabajo futuro

Este capítulo sintetiza los hallazgos principales derivados del desarrollo y evaluación de los modelos propuestos en este trabajo fin de máster. Se presentan primero las conclusiones fundamentales del estudio, seguidas por recomendaciones y propuestas específicas para futuras investigaciones.

7.1 Conclusiones

El desarrollo de este trabajo ha permitido abordar con éxito el objetivo principal planteado inicialmente: diseñar, implementar y evaluar modelos de predicción del tráfico vehicular mediante técnicas avanzadas de aprendizaje profundo, destacando especialmente la arquitectura Trafficformer basada en Transformers con atención espacial.

La comparación entre MLP y Trafficformer ha demostrado la clara superioridad del segundo en los tres contextos evaluados (sourceids 1, 2 y 5). Trafficformer logró resultados significativamente mejores en métricas clave como MAE, RMSE y R^2 , validando así su capacidad para explotar eficazmente dependencias espaciales y temporales, especialmente en escenarios complejos con alta densidad y distribución espacial.

La metodología de experimentación exhaustiva permitió identificar combinaciones óptimas de hiperparámetros (tamaño de ventana temporal, número de cabezas de atención, tamaño de embedding, entre otros), destacando la importancia crítica del ajuste adecuado de estos parámetros para obtener un rendimiento óptimo del modelo.

Asimismo, la utilización de la plataforma *Weights & Biases* ha resultado fundamental para asegurar la reproducibilidad, transparencia y análisis riguroso de los experimentos, proporcionando un seguimiento detallado y sistemático del proceso de entrenamiento y evaluación.

No obstante, el trabajo ha revelado diversas limitaciones, principalmente relacionadas con la representatividad espacial de las fuentes de datos, restricciones técnicas de hardware, posibles sesgos en los datos y limitaciones en la generalización externa. Estas limitaciones representan oportunidades claras para futuros desarrollos y mejoras.

7.2 Trabajo Futuro

De cara a futuras investigaciones, se identifican diversas líneas de trabajo que permitirían mejorar aún más los resultados obtenidos:

- **Ampliación de datos y fuentes adicionales:** Incorporar nuevos conjuntos de datos provenientes de otras regiones o ciudades, así como otras variables exógenas no consideradas, para mejorar la generalización y robustez del modelo.
- **Optimización y escalado del modelo:** Realizar experimentos con infraestructuras computacionales más potentes (por ejemplo, GPU de alto rendimiento o clústeres en la nube), lo que permitiría profundizar en la exploración de hiperparámetros y probar arquitecturas más complejas. Para ello, en el proyecto actual se provee de una implementación lista para usar en Amazon Web Services.
- **Técnicas avanzadas de tratamiento de datos:** Evaluar el impacto de estrategias más avanzadas de imputación de datos, tratamiento de valores atípicos, y técnicas de aprendizaje auto-supervisado que podrían mejorar la calidad y representatividad de los datasets empleados.
- **Evaluación en tiempo real y despliegue operativo:** Desarrollar un sistema integrado capaz de realizar predicciones en tiempo real, desplegado en un entorno operativo real, permitiendo validar su utilidad práctica y detectar oportunidades adicionales de mejora.
- **Interpretabilidad y explicabilidad:** Implementar técnicas adicionales que aumenten la interpretabilidad de los modelos desarrollados, facilitando la comprensión y justificación de las decisiones tomadas por el sistema.
- **Integración de otros modelos avanzados:** Explorar modelos complementarios como Graph Neural Networks (GNNs) o modelos híbridos que podrían aprovechar aún más las estructuras espaciales complejas presentes en los datos de tráfico.

En conclusión, este TFM establece una sólida base metodológica y técnica para futuros trabajos en predicción de tráfico, y destaca la capacidad de los modelos basados en Transformers, particularmente Trafficformer, para abordar eficazmente desafíos de alta complejidad espacial y temporal.

Acrónimos y Abreviaturas

API Interfaz de programación de aplicaciones. Conjunto de funciones y definiciones que permiten la comunicación entre sistemas de software.

ARIMA AutoRegressive Integrated Moving Average.

CAPV Comunidad Autónoma del País Vasco.

CNN Convolutional Neural Networks o Redes Neuronales Convolucionales.

DL Deep Learning o Aprendizaje Profundo.

GAT Graph Attention Networks.

GCN Graph Convolutional Networks.

GGNN Gated Graph Neural Networks.

GNN Graph Neural Networks.

GPU Graphics Processing Unit.

GRU Gated Recurrent Units.

ITS Sistemas Inteligentes de Transporte.

KNN K-Nearest Neighbors.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error o Error Medio Absoluto.

MAPE Mean Absolute Percentage Error o Error Porcentual Absoluto Medio.

MLP Multi Layer Perceptron o Perceptrones Multi Capa.

MRE Mean Relative Error o Error Medio Relativo.

RAM Random Access Memory.

RF Random Forests.

RMSE Root Mean Square Error o Error Cuadrático Medio.

RNN Recurrent Neural Networks o Redes Neuronales Recurrentes.

SARIMA Seasonal AutoRegressive Integrated Moving Average.

SSD Solid State Drive.

SVM Support Vector Machines.

SVR Support Vector Regression.

Bibliografía

- Aditya, F., Nasution, S., & Virgono, A. (2020, 10). Traffic flow prediction using sumo application with k-nearest neighbor (knn) method. *International Journal of Integrated Engineering*, 12. Retrieved from https://www.researchgate.net/publication/346622922_Traffic_Flow_Prediction_using_SUMO_Application_with_K-Nearest_Neighbor_KNN_Method doi: 10.30880/ijie.2020.12.07.011
- Chang, A., Ji, Y., & Bie, Y. (2025). Transformer-based short-term traffic forecasting model considering traffic spatiotemporal correlation. *Frontiers in Neurorobotics, Volume 19 - 2025*. Retrieved from <https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2025.1527908> doi: 10.3389/fnbot.2025.1527908
- Chen, X., Wang, J., & Xie, K. (2021). Trafficstream: A streaming traffic flow forecasting framework based on graph neural networks and continual learning. Retrieved from <https://arxiv.org/abs/2106.06273>
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014, October). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1724–1734). Doha, Qatar: Association for Computational Linguistics. Retrieved from <https://aclanthology.org/D14-1179/> doi: 10.3115/v1/D14-1179
- DataCamp. (2023). *Pytorch vs tensorflow vs keras: Key differences*. Retrieved 2025-05-23, from <https://www.datacamp.com/tutorial/pytorch-vs-tensorflow-vs-keras> (Accedido el 23 de mayo de 2025)
- Eusko Jaurlaritza / Gobierno Vasco. (2024). *Estaciones meteorológicas. lecturas recogidas en 2024.* <https://opendata.euskadi.eus/catalogo/-/estaciones-meteorologicas-lecturas-recogidas-en-2024/>. ([Conjunto de datos]. Open Data Euskadi.)
- Eusko Jaurlaritza / Gobierno Vasco. (s.f.). *Abreviaturas de los tipos de mediciones de sensores meteorológicos.* <https://opendata.euskadi.eus//webopd00-dataset/es/>

contenidos/ds_meteorologicos/met_stations_ds_2009/es_dataset/adjuntos/abbreviaturas.txt. ([Archivo de texto]. Open Data Euskadi.)

Gobierno Vasco. (2024). *Portal de open data euskadi*. Retrieved 2025-05-23, from <https://opendata.euskadi.eus/> (Accedido el 23 de mayo de 2025)

Gobierno Vasco, Eusko Jaurlaritza. (2020). *Tercer plan general de carreteras del país vasco 2017-2028*. Retrieved 2020-05-25, from <https://www.euskadi.eus/tercer-plan-general-de-carreteras-del-pais-vasco-2017-2028/web01-a2bideko/es/> (Accedido el 18 de abril de 2025)

Gobierno Vasco, Eusko Jaurlaritza. (2025a). *Apis de open data euskadi*. Retrieved 2025-04-18, from <https://opendata.euskadi.eus/apis/-/apis-open-data/> (Accedido el 18 de abril de 2025)

Gobierno Vasco, Eusko Jaurlaritza. (2025b). *Open data euskadi: Api traffic*. Retrieved 2025-04-18, from <https://opendata.euskadi.eus/api-traffic/> (Accedido el 18 de abril de 2025)

Gobierno Vasco, Eusko Jaurlaritza. (2025c). *Open data euskadi: Euskalmet api*. Retrieved 2025-04-18, from <https://opendata.euskadi.eus/api-euskalmet/-/api-de-euskalmet/> (Accedido el 18 de abril de 2025)

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504–507. Retrieved from <https://www.science.org/doi/10.1126/science.1127647> doi: 10.1126/science.1127647

Hochreiter, S., & Schmidhuber, J. (1997, 11). Long short-term memory. *Neural Computation*, 9(8), 1735-1780. Retrieved from <https://doi.org/10.1162/neco.1997.9.8.1735> doi: 10.1162/neco.1997.9.8.1735

Katambire, V. N., Musabe, R., Uwitonze, A., & Mukanyiligira, D. (2023). Forecasting the traffic flow by using arima and lstm models: Case of muhimba junction. *Forecasting*, 5(4), 616–628. Retrieved from <https://www.mdpi.com/2571-9394/5/4/34> doi: 10.3390/forecast5040034

- Khan, R. H., Miah, J., Arafat, S. M. Y., Syeed, M. M. M., & Ca, D. M. (2023). Improving traffic density forecasting in intelligent transportation systems using gated graph neural networks. Retrieved from <https://arxiv.org/abs/2310.17729>
- Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. Retrieved from <https://arxiv.org/abs/1609.02907>
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (neurips)* (Vol. 25). Retrieved from https://papers.nips.cc/paper_2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
- Kumar, S. V., & Vanajakshi, L. (2015). Short-term traffic flow prediction using seasonal arima model with limited input data. *European Transport Research Review*, 7(3), 21. Retrieved from <https://doi.org/10.1007/s12544-015-0170-8> doi: 10.1007/s12544-015-0170-8
- Liu, Y., Liu, Z., Liu, J., Zhang, X., & Tang, M. (2020). Congestion time prediction model based on multiple regression analysis and survival analysis. *PLOS ONE*, 15(7), e0235660. Retrieved from <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0235660> doi: 10.1371/journal.pone.0235660
- Liu, Y., Song, Y., Zhang, Y., & Liao, Z. (2022). Wt-2dcnn: A convolutional neural network traffic flow prediction model based on wavelet reconstruction. *Physica A: Statistical Mechanics and its Applications*, 603, 127817. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0378437122005349> doi: <https://doi.org/10.1016/j.physa.2022.127817>
- Ma, C., Zhao, Y., Dai, G., Xu, X., & Wong, S.-C. (2023). A novel stfsa-cnn-gru hybrid model for short-term traffic speed prediction. *IEEE Transactions on Intelligent Transportation Systems*, 24(4), 3728-3737. doi: 10.1109/TITS.2021.3117835
- McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 115–133. Retrieved from <https://link.springer.com/article/10.1007/BF02478259> doi: 10.1007/BF02478259

- Minsky, M., & Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. MIT Press. Retrieved from <https://direct.mit.edu/books/monograph/3132/>
PerceptronsAn-Introduction-to-Computational
- Momin, K. A., Barua, S., Jamil, M. S., & Hamim, O. F. (2023). Short duration traffic flow prediction using kalman filtering. In *6th international conference on civil engineering for sustainable development (iccesd 2022)*. AIP Publishing. Retrieved from <http://dx.doi.org/10.1063/5.0129721> doi: 10.1063/5.0129721
- Omar, M., Yakub, F., Abdullah, S. S., Abd Rahim, M. S., Zuhairi, A. H., & Govindan, N. (2024). One-step vs horizon-step training strategies for multi-step traffic flow forecasting with direct particle swarm optimization grid search support vector regression and long short-term memory. *Expert Systems with Applications*, 252, 124154. doi: 10.1016/j.eswa.2024.124154
- PyTorch. (2025). *ReduceLronplateau*. Retrieved from https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408. Retrieved from <https://psycnet.apa.org/doi/10.1037/h0042519> doi: 10.1037/h0042519
- Ruder, S. (2017). *An overview of gradient descent optimization algorithms*. Retrieved from <https://arxiv.org/abs/1609.04747>
- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1), 61-80. doi: 10.1109/TNN.2008.2005605
- UnfoldAI. (2024). *Keras vs pytorch — which dl framework to choose in 2024?* Retrieved 2025-05-23, from <https://unfoldai.com/keras-vs-pytorch-in-2024> (Accedido el 23 de mayo de 2025)
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. Retrieved from <https://arxiv.org/abs/1706.03762> (Publicado en 2017, última revisión (v7): 2023) doi: 10.48550/arXiv.1706.03762

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. Retrieved from <https://arxiv.org/abs/1710.10903>

Wikipedia, la enciclopedia libre. (2025). *Problema de desvanecimiento de gradiente – wikipedia, la enciclopedia libre*. Retrieved 2025-04-18, from https://es.wikipedia.org/wiki/Problema_de_desvanecimiento_de_gradiente (Accedido el 18 de abril de 2025)

Wu, J. (2024, 08). A study on short-term traffic flow prediction based on random forest regression. *Highlights in Science, Engineering and Technology*, 107, 323-331. Retrieved from https://www.researchgate.net/publication/383208801_A_Study_on_Short-Term_Traffic_Flow_Prediction_Based_on_Random_Forest_Regression doi: 10.54097/tv6vfy08

YANG, D., LI, S., PENG, Z., WANG, P., WANG, J., & YANG, H. (2019). Mf-cnn: Traffic flow prediction using convolutional neural network and multi-features fusion. *IEICE Transactions on Information and Systems*, E102.D(8), 1526-1536. doi: 10.1587/transinf.2018EDP7330

Zhao, Z., Chen, W., Wu, X., Chen, P. C. Y., & Liu, J. (2017). Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2), 68-75. Retrieved from <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-its.2016.0208> doi: <https://doi.org/10.1049/iet-its.2016.0208>

Anexo A – Descripción de sensores meteorológicos

En este anexo se presenta un listado exhaustivo de los sensores meteorológicos utilizados en la construcción del dataset. Cada sensor cuenta con un identificador único, una abreviatura técnica y una descripción textual de la variable que mide. Estos sensores provienen de estaciones automáticas distribuidas en la CAPV, y sus mediciones son fundamentales para enriquecer los MobilitySnapshot con información climática contextual.

La codificación y nomenclatura de los sensores meteorológicos responde al estándar empleado por Euskalmet, el cual está documentado en su catálogo de abreviaturas técnicas de tipo de sensor Eusko Jaurlaritza / Gobierno Vasco (s.f.).

ID	Nombre	Descripción
12	Dir.Med	Dirección media del viento en grados (º)
14	Vel.Max	Racha máxima del viento horizontal en km/h
16	Sig.Vel	Sigma de la velocidad del viento en km/h
17	Sig.Dir	Sigma de la dirección del viento en grados (º)
18	Cub.Vto	Velocidad cúbica media del viento en Dm/s ³
21	Tem.Aire	Temperatura del aire en ºC
31	Humedad	Humedad relativa del aire en %
40	Precip.	Precipitación acumulada en mm o l/m ²
50	Presión	Presión atmosférica en milibares (mb)
60	Nivel 1	Nivel de lámina de agua en metros (m)
61	Nivel 2	Nivel de lámina de agua en metros (m)
70	Irradia.	Irradiancia solar global en w/m ²
90	Tem.Agua	Temperatura del agua en ºC
91	Oxígeno	Oxígeno disuelto en ppm
92	pH	pH del agua
93	Conduct.	Conductividad del agua en µS
94	Amonio	Amonio en mg/l
95	Turbidez	Turbidez del agua en NTU
96	Redox.	Potencial redox del agua en mV

ID	Nombre	Descripción
97	Mat.Org.	Materia orgánica medida como demanda de oxígeno
11	VelMed	Velocidad media del viento en km/h
22	Tem.Sue	Temperatura del suelo en °C
B0	Visibili	Visibilidad en metros
B1	Nivelm 1	Nivel del mar (sensor 1) en metros
B2	Nivelm 2	Nivel del mar (sensor 2) en metros
B3	Ola Med	Altura media de ola en metros
B4	Ola Max	Altura máxima de ola en metros
B5	Ola Sig	Altura significante de ola en metros
B6	Ola Per	Periodo de oleaje en segundos
B7	Pre MSP	Presión hidrostática MSP en hPa
B8	Pre LSP	Presión hidrostática LSP en hPa
B9	Vel Cor	Velocidad de corriente en cm/s
BA	Dir Cor	Dirección de la corriente en grados (º)
BB	Tem Mar	Temperatura del agua del mar en °C
BC	Tem Ter	Temperatura termistor en °C
BD	Ola Sig2	Altura significante de ola (variante) en metros
72	Rad.Refl	Radiación solar reflejada en w/m ²
73	Rad.UV	Radiación UV en J/m ² h

Anexo B – Código fuente del generador de snapshots

A continuación se presenta el código fuente completo de la clase `MobilitySnapshotGeneratorService`, responsable de generar las instancias de `MobilitySnapshot` mediante la integración de datos de tráfico, meteorología e incidencias.

Listing 5: Clase `MobilitySnapshotGeneratorService`

```
package es.joninx.tfm.dc.service

import es.joninx.tfm.dc.builder.dataset.MobilitySnapshotBuilder
import es.joninx.tfm.dc.config.Cfg
import es.joninx.tfm.dc.repository.meteo.XmlRepository
import es.joninx.tfm.dc.repository.meteo.StationRepository
import es.joninx.tfm.dc.repository.traffic.FlowRepository
import es.joninx.tfm.dc.repository.traffic.IncidenceRepository
import es.joninx.tfm.dc.repository.traffic.MeterRepository
import es.joninx.tfm.dc.util.DateTimeUtils
import org.apache.logging.log4j.LogManager
import org.apache.logging.log4j.Logger
import org.springframework.stereotype.Service
import java.time.Duration
import java.time.LocalDateTime

@Service
class MobilitySnapshotGeneratorService(
    private val cfg: Cfg,
    private val snapshotBuilder: MobilitySnapshotBuilder,
    private val snapshotPersistenceService: MobilitySnapshotPersistenceService,
    private val flowRepository: FlowRepository,
    private val meterRepository: MeterRepository,
    private val incidenceRepository: IncidenceRepository,
    private val stationRepository: StationRepository,
```

```
private val meteoReadingsRepository: ReadingXmlRepository,  
) {  
  
    /**  
     * Mapa cache: meterId → stationId  
     */  
  
    private val meterToStationCache = mutableMapOf<String, String>()  
  
    fun getNearestStationId(meterId: String, latitude: Double, longitude:  
        Double): String? {  
        // ¿Ya lo tenemos cacheado?  
        meterToStationCache[meterId]?.let {  
            log.debug("Cache HIT: meterId=$meterId → stationId=$it")  
            return it  
        }  
  
        // Si no, buscamos la estación más cercana  
        val nearestStation = stationRepository.findNearestStation(  
            longitude = longitude,  
            latitude = latitude,  
            maxDistanceMeters = cfg.algorithm.maxDistanceToStation  
        ).blockFirst() ?: return null  
  
        meterToStationCache[meterId] = nearestStation.stationId  
        log.debug("Cache MISS: meterId=$meterId →  
            stationId=${nearestStation.stationId}")  
        return nearestStation.stationId  
    }  
  
    fun generateSnapshots(  
        sourceIds: List<String>,  
        startDate: LocalDateTime,  
        endDate: LocalDateTime,  
        batchSize: Int = 500
```

```
) {  
  
    log.debug("Comenzando generación de MobilitySnapshots para  
    sourceIds=${sourceIds.joinToString(",")}, fechas entre  
    '${DateTimeUtils.format(startDate)}' y  
    '${DateTimeUtils.format(endDate)}'")  
  
    val meters = meterRepository.findAllBySourceIdIn(sourceIds)  
        .collectMap { it.meterId }  
        .block() ?: emptyMap()  
  
    val intervals = generateIntervals(startDate, endDate,  
        cfg.algorithm.timeWindowDuration)  
    var totalSnapshots = 0  
  
    intervals.forEachIndexed { idx, (intervalStart, intervalEnd) ->  
        val flows =  
            flowRepository.findAllBySourceIdInAndDateTimeBetweenQuery(  
                sourceIds, intervalStart, intervalEnd  
            ).collectList().block() ?: emptyList()  
  
        val groupedByMeter = flows.groupBy { it.meterId }  
  
        val snapshots = groupedByMeter.mapNotNull { (meterId, flowList) ->  
            val meter = meters[meterId]  
            if (meter != null && flowList.isNotEmpty()) {  
                val totalVehiclesSum = flowList.sumOf {  
                    it.totalVehicles.toIntOrNull() ?: 0 }  
  
                // Obtener incidencias para el intervalo  
                val latitude = meter.latitude  
                val longitude = meter.longitude  
  
                // Busca incidencias cercanas y activas  
                val incidences = incidenceRepository
```

```
.findIncidencesNearAndActive(  
    longitude, latitude, cfg.algorithm.maxDistanceToIncidences,  
    intervalStart, intervalEnd  
)  
.collectList()  
.block() ?: emptyList()  
  
// Meteo  
  
val nearestStationId = getNearestStationId(  
    meterId = meterId,  
    latitude = latitude,  
    longitude = longitude  
)  
  
val meteoReadings = if (nearestStationId != null) {  
    meteoReadingsRepository.findReadingsByStationIdAndDateTimeBetween(  
        stationId = nearestStationId,  
        windowStart = intervalStart,  
        windowEnd = intervalEnd  
    ).collectList().block() ?: emptyList()  
} else emptyList()  
  
log.debug("Intervalo '${DateTimeUtils.format(intervalStart)}' →  
    '${DateTimeUtils.format(intervalEnd)}' | meterId=$meterId |  
    numFlows=${flowList.size} |  
    totalVehiclesSum=$totalVehiclesSum |  
    totalIncidences=${incidences.size}  
    |meteoReadings=${meteoReadings.size}")  
  
snapshotBuilder.fromGroupedFlows(  
    flows = flowList,  
    meter = meter,  
    windowStartTime = intervalStart,  
    windowEndTime = intervalEnd,  
    totalVehicles = totalVehiclesSum,  
    incidences = incidences,
```

```

meteoReadings = meteoReadings,
)
} else {
    if (meter == null) {
        log.warn("MeterId=$meterId no encontrado en meterMap para
ventana ${DateTimeUtils.format(intervalStart)} →
${DateTimeUtils.format(intervalEnd)}")
    } else {
        log.warn("No hay flows para meterId=$meterId en ventana
${DateTimeUtils.format(intervalStart)} →
${DateTimeUtils.format(intervalEnd)}")
    }
    null
}
}

// Guardar por lotes
snapshots.chunked(batchSize).forEach { batch ->
    snapshotPersistenceService.saveBatch(batch).block()
    log.debug("Batch guardado para intervalo $intervalStart →
$intervalEnd (${batch.size} snapshots)")
}
totalSnapshots += snapshots.size

if ((idx + 1) % 24 == 0) { // Cada 12 horas
    log.debug("Progreso: ${idx + 1} de ${intervals.size} intervalos
procesados, $totalSnapshots snapshots generados.")
}
}

log.debug("Generación de MobilitySnapshots finalizada. Total
snapshots: $totalSnapshots")
}

```

```
// Utilidad para generar ventanas de 30 minutos (la misma que antes)
fun generateIntervals(start: LocalDateTime, end: LocalDateTime, step:
    Duration): List<Pair<LocalDateTime, LocalDateTime>> {
    val intervals = mutableListOf<Pair<LocalDateTime, LocalDateTime>>()
    var current = start
    while (current.isBefore(end)) {
        val next = current.plus(step)
        intervals.add(Pair(current, if (next.isBefore(end)) next else end))
        current = next
    }
    return intervals
}

companion object {
    val log: Logger = LogManager.getLogger(this::class.java)
}
```

Anexo C – Plantilla de infraestructura para AWS

Con el objetivo de permitir la escalabilidad y entrenamiento de los modelos desarrollados en la nube, se ha preparado una plantilla de infraestructura como código en formato YAML, siguiendo el estándar AWS CloudFormation.

Esta plantilla permite desplegar de forma automática una instancia con aceleración por GPU, almacenamiento persistente y conexión segura a la base de datos local mediante VPN. Aunque no ha sido necesario su uso durante el desarrollo del presente trabajo, se considera un componente valioso para futuras ejecuciones de alto rendimiento o despliegues remotos. La plantilla está pensada para ser lanzada directamente desde la consola de AWS o mediante herramientas como AWS SAM o AWS CLI.

Listing 6: Plantilla CloudFormation utilizada para el despliegue de entorno de entrenamiento

AWSTemplateFormatVersion: '2010-09-09'

Description: EC2 g5. xlarge Ubuntu 24.04 DLAMI GPU con volumen persistente y
WireGuard

Parameters:

KeyName:

Type: AWS::EC2::KeyPair::KeyName

Default: joninx

WireguardConfBucket:

Type: String

Default: tfm-jon- trafficformer

WireguardConfKey:

Type: String

Default: wireguard/joninx .conf

Resources:

VPC:

Type: AWS::EC2::VPC

Properties:

CidrBlock: 10.100.0.0/16

Tags: [{Key: Name, Value: tfm}]

InternetGateway: {Type: AWS::EC2::InternetGateway}

AttachGateway:

Type: AWS::EC2::VPCGatewayAttachment

Properties:

VpcId: !Ref VPC

InternetGatewayId: !Ref InternetGateway

PublicSubnet:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref VPC

CidrBlock: 10.100.1.0/24

MapPublicIpOnLaunch: true

AvailabilityZone : !Select [0, !GetAZs '']

Tags: [{Key: Name, Value: tfm}]

RouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref VPC

Tags: [{Key: Name, Value: tfm}]

PublicRoute:

Type: AWS::EC2::Route

DependsOn: AttachGateway

Properties:

RouteTableId: !Ref RouteTable

DestinationCidrBlock : 0.0.0.0/0

GatewayId: !Ref InternetGateway

RouteTableAssoc:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

SubnetId: !Ref PublicSubnet

RouteTableId: !Ref RouteTable

SecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: SSH access

VpcId: !Ref VPC

SecurityGroupIngress:

- **IpProtocol:** tcp

FromPort: 22

ToPort: 22

CidrIp: 0.0.0.0/0

Tags: [{Key: Name, Value: tfm}]

EC2InstanceProfile :

Type: AWS::IAM::InstanceProfile

Properties :

Roles: [! Ref EC2S3AccessRole]

EC2S3AccessRole:

Type: AWS::IAM::Role

Properties :

AssumeRolePolicyDocument:

Version: "2012-10-17"

Statement:

- **Effect :** Allow

Principal : { Service: ec2.amazonaws.com}

Action: sts:AssumeRole

Policies :

- **PolicyName:** S3WGAccess

PolicyDocument:

Version: "2012-10-17"

Statement:

- **Effect :** Allow

Action: s3:GetObject

Resource: !Sub arn:aws:s3:::\${WireguardConfBucket}/\${WireguardConfKey}

EC2DataVolume:

Type: AWS::EC2::Volume

Properties :

AvailabilityZone : !Select [0, !GetAZs '']

Size: 200

VolumeType: gp3

Encrypted: true

Tags: [{Key: Name, Value: tfm}]

EC2Instance:

Type: AWS::EC2::Instance

Properties :

InstanceType: g5.2xlarge

KeyName: !Ref KeyName

SubnetId: !Ref PublicSubnet

SecurityGroupIds: [! Ref SecurityGroup]

IamInstanceProfile: !Ref EC2InstanceProfile

BlockDeviceMappings:

- **DeviceName:** /dev/sda1

Ebs:

VolumeSize: 60

VolumeType: gp3

DeleteOnTermination: true

ImageId: !Sub

```
"{{resolve:ssm:/aws/service/deeplearning/ami/x86_64/base-oss-nvidia-driver}}
```

Tags: [{Key: Name, Value: tfm}]

UserData:

```
Fn::Base64: !Sub |  
#!/bin/bash  
set -eux  
apt-get update -y  
apt-get upgrade -y  
# Instala Python 3.13 desde deadsnakes PPA (o desde source si  
necesario)  
if ! python3.13 --version 2>/dev/null; then  
    apt-get install -y software-properties-common  
    add-apt-repository ppa:deadsnakes/ppa -y  
    apt-get update -y  
    apt-get install -y python3.13 python3.13-venv python3.13-distutils  
    fi  
# Configura python3 para que apunte a 3.13 por defecto  
update-alternatives -- install /usr/bin/python3 python3  
/usr/bin/python3.13 2  
update-alternatives --set python3 /usr/bin/python3.13  
# Instala pip y venv para 3.13 si no están  
curl -sS https://bootstrap.pypa.io/get-pip.py | python3.13  
  
# Python 3.13 alternativo  
update-alternatives -- install /usr/bin/python3 python3  
/usr/bin/python3.13 2  
  
# Instala wireguard y awscli  
apt-get install -y wireguard awscli  
  
# WireGuard config
```

```

aws s3 cp s3://${WireguardConfBucket}/${WireguardConfKey}
    /etc/wireguard/wg0.conf
chmod 600 /etc/wireguard/wg0.conf
systemctl enable wg-quick@wg0 && systemctl start wg-quick@wg0

# Attach, format & mount data volume
mkfs.ext4 -F /dev/xvdb || true
mkdir -p /mnt/tfmdata
mount /dev/xvdb /mnt/tfmdata
echo '/dev/xvdb /mnt/tfmdata ext4 defaults,nofail 0 2' >>/etc/fstab
chown ubuntu:ubuntu /mnt/tfmdata

```

AttachDataVolume:**Type:** AWS::EC2::VolumeAttachment**Properties :****Device:** /dev/xvdb**InstanceId:** !Ref EC2Instance**VolumId:** !Ref EC2DataVolume**Outputs:****PublicIP :****Description:** "EC2 Public IP"**Value:** !GetAtt EC2Instance. PublicIp

Anexo D – Código fuente de la arquitectura Trafficformer

A continuación se presenta el código fuente completo de la arquitectura Trafficformer, implementada en Python y utilizada como modelo principal en el presente trabajo. El código está debidamente documentado mediante docstrings, y ha sido diseñado con una estructura modular que facilita su reutilización y extensión.

```
import math

import torch
import torch.nn as nn


class TemporalPositionalEncoding(nn.Module):
    """
    Codificación posicional sinusoidal para secuencias temporales.

    Esta clase implementa una codificación posicional basada en funciones seno y
    coseno,
    siguiendo la propuesta original del paper "Attention is All You Need". Se aplica
    directamente
    sobre cada paso temporal de la secuencia de entrada para preservar el orden en
    modelos
    sin recurrencia.

    La codificación se suma a las features originales antes de ser procesadas por
    capas densas,
    permitiendo al modelo diferenciar posiciones temporales dentro de la ventana.

    Attributes:
        seq_len (int): Longitud de la secuencia temporal.
        num_features (int): Número de variables por paso temporal.
        pos_encoding (torch.Tensor): Tensor con la codificación posicional
            precomputada.

    Methods:
        forward(x): Aplica la codificación posicional a un tensor de entrada.
    """

```

```
def __init__(self, seq_len, num_features):
    """
    Inicializa el codificador posicional.

    Args:
        seq_len (int): Longitud de la secuencia temporal (número de pasos).
        num_features (int): Número de variables por paso temporal (dimensión del
            embedding por paso).

    """
    super().__init__()
    self.seq_len = seq_len
    self.num_features = num_features
    self.pos_encoding = self._build_positional_encoding(seq_len, num_features) # (seq_len, num_features)

def _build_positional_encoding(self, seq_len, d_model):
    """
    Genera la matriz de codificación posicional usando funciones seno y coseno.

    Args:
        seq_len (int): Longitud de la secuencia temporal.
        d_model (int): Dimensión del vector de entrada por paso temporal.

    Returns:
        torch.Tensor: Matriz de codificación de tamaño [seq_len, d_model].
    """
    pe = torch.zeros(seq_len, d_model)
    position = torch.arange(0, seq_len, dtype=torch.float).unsqueeze(1)
    div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))
    pe[:, 0::2] = torch.sin(position * div_term[: (d_model + 1) // 2])
    pe[:, 1::2] = torch.cos(position * div_term[:d_model // 2])
    return pe # (seq_len, d_model)

def forward(self, x):
    """
    Suma la codificación posicional al tensor de entrada.

    Args:

```

```
x (torch.Tensor): Tensor de entrada de tamaño [batch_size, num_meters,  
seq_len, num_features].
```

Returns:

```
torch.Tensor: Tensor codificado con posición, del mismo tamaño que la  
entrada.
```

"""

```
pe = self.pos_encoding.to(x.device) # (seq_len, num_features)  
return x + pe # broadcasting en eje temporal
```

```
class TrafficTemporalFeatureExtractor(nn.Module):
```

"""

Extractor de características temporales para secuencias por sensor.

Esta clase transforma la secuencia temporal de cada sensor (nodo) en un vector de embedding fijo, incorporando codificación posicional y un perceptrón multicapa profundo. Está diseñada para alimentar un bloque de atención espacial como el utilizado en Trafficformer.

Arquitectura:

- Codificación posicional sinusoidal.
- MLP con las siguientes características:
 - Tres capas lineales con normalización LayerNorm tras cada una.
 - Dropout para regularización y evitar sobreajuste.
 - Activación ReLU para capturar no linealidades.
 - Todos los hiperparámetros principales son configurables.

Attributes:

```
seq_len (int): Longitud de la ventana temporal histórica.  
num_features (int): Número de variables por paso temporal.  
input_dim (int): Dimensión total de entrada (seq_len * num_features).  
pos_encoder (TemporalPositionalEncoding): Módulo de codificación posicional.  
layers (nn.ModuleList): Capas densas de la MLP.  
norms (nn.ModuleList): Normalizaciones LayerNorm.  
dropouts (nn.ModuleList): Capas de regularización por dropout.
```

"""

```
def __init__(  
    self,  
    seq_len: int,  
    num_features: int,  
    embedding_dim: int,  
    hidden_dims=(128, 128),  
    dropout=0.2  
):  
    super().__init__()  
    self.seq_len = seq_len  
    self.num_features = num_features  
    self.input_dim = seq_len * num_features  
  
    # Codificación posicional temporal  
    self.pos_encoder = TemporalPositionalEncoding(seq_len, num_features)  
  
    # Red MLP para mapear a embedding  
    dims = [self.input_dim] + list(hidden_dims) + [embedding_dim]  
    self.layers = nn.ModuleList()  
    self.norms = nn.ModuleList()  
    self.dropouts = nn.ModuleList()  
  
    for i in range(len(dims) - 1):  
        self.layers.append(nn.Linear(dims[i], dims[i + 1]))  
        self.norms.append(nn.LayerNorm(dims[i + 1]))  
        self.dropouts.append(nn.Dropout(dropout))  
  
def forward(self, x):  
    """  
    Aplica el extractor temporal nodo a nodo y devuelve los embeddings temporales.  
  
    Args:  
        x (torch.Tensor): Tensor de entrada de shape [batch, num_meters, seq_len, num_features].  
  
    Returns:  
        torch.Tensor: Tensor de shape [batch, num_meters, embedding_dim], donde  
        embedding_dim es la dimensión de la representación temporal de cada  
        nodo.
```

```
"""
x = self.pos_encoder(x) # Codificación posicional
batch_size, num_meters, T, F = x.shape
x = x.view(batch_size * num_meters, T * F)

for i in range(len(self.layers)):
    x = self.layers[i](x)
    x = self.norms[i](x)
    x = torch.relu(x)
    if i < len(self.layers) - 1: # No Dropout after last layer
        x = self.dropouts[i](x)

x = x.view(batch_size, num_meters, -1) # [batch, num_meters, embedding_dim]
return x

class SpatialMultiHeadAttention(nn.Module):
"""
Multi-Head Attention espacial con soporte para máscaras personalizadas.

Esta clase adapta el mecanismo estándar de atención multi-cabeza para incorporar
una máscara espacial arbitraria que restringe la atención solo a los nodos
conectados físicamente.

"""

def __init__(self, embed_dim, num_heads):
    """
    Inicializa el bloque de atención multi-cabeza espacial.

    Args:
        embed_dim (int): Dimensión del embedding de entrada/salida.
        num_heads (int): Número de cabezas de atención.

    """
    super().__init__()
    self.attn = nn.MultiheadAttention(embed_dim, num_heads, batch_first=True)

    def forward(self, x, spatial_mask=None):
        """
        Aplica atención multi-cabeza sobre los nodos, restringida por la máscara
        
```

espacial.

Args:

```
x (torch.Tensor): Tensor de shape [batch, num_nodes, embed_dim].  
spatial_mask (torch.Tensor or None): Matriz [num_nodes, num_nodes] booleana  
o binaria, donde 1 indica conexión.
```

Returns:

```
torch.Tensor: Tensor de salida tras atención multi-cabeza, de shape [batch,  
num_nodes, embed_dim].
```

"""

```
# x: [batch, num_nodes, embed_dim]  
# spatial_mask: [num_nodes, num_nodes] -> 1: conectado, 0: no conectado  
(float/bool)  
attn_mask = None  
if spatial_mask is not None:  
    # PyTorch espera mask de shape [num_nodes, num_nodes], True = MASCARA, es  
    # decir, IGNORA esos valores.  
    attn_mask = ~spatial_mask.bool() # Invierte la máscara: 1->0 y 0->1
```

```
out, _ = self.attn(x, x, x, attn_mask=attn_mask)  
return out
```

```
class TrafficformerEncoderLayer(nn.Module):
```

"""

Capa básica de encoder tipo Transformer para interacción espacial.

Cada capa realiza:

- Atención multi-cabeza con máscara espacial (residual + normalización).
- Feedforward no lineal (residual + normalización).
- Dropout tras cada bloque.

"""

```
def __init__(self, embed_dim, num_heads, ff_hidden_dim=None, dropout=0.1):
```

"""

Inicializa una capa de encoder Trafficformer.

Args:

```
embed_dim (int): Dimensión del embedding de entrada/salida.  
num_heads (int): Número de cabezas de atención.  
ff_hidden_dim (int, opcional): Tamaño de la capa oculta en el feedforward  
(por defecto: 2*embed_dim).  
dropout (float): Proporción de dropout tras atención y feedforward.  
"""  
super().__init__()  
self.attn = SpatialMultiHeadAttention(embed_dim, num_heads)  
self.norm1 = nn.LayerNorm(embed_dim)  
self.ff = nn.Sequential(  
    nn.Linear(embed_dim, ff_hidden_dim or embed_dim * 2),  
    nn.ReLU(),  
    nn.Linear(ff_hidden_dim or embed_dim * 2, embed_dim),  
)  
self.norm2 = nn.LayerNorm(embed_dim)  
self.dropout = nn.Dropout(dropout)  
  
def forward(self, x, spatial_mask=None):  
    """  
    Aplica atención multi-cabeza y feedforward con conexiones residuales.  
  
    Args:  
        x (torch.Tensor): Tensor de entrada [batch, num_nodes, embed_dim].  
        spatial_mask (torch.Tensor or None): Matriz de adyacencia espacial opcional.  
  
    Returns:  
        torch.Tensor: Tensor procesado [batch, num_nodes, embed_dim].  
    """  
    # Multi-head attention (residual + norm)  
    attn_out = self.attn(x, spatial_mask=spatial_mask)  
    x = self.norm1(x + self.dropout(attn_out))  
    # Feedforward (residual + norm)  
    ff_out = self.ff(x)  
    x = self.norm2(x + self.dropout(ff_out))  
    return x  
  
class TrafficformerEncoder(nn.Module):  
    """
```

Módulo encoder compuesto por varias capas Transformer apiladas para modelar la interacción espaciotemporal.

Este bloque es responsable de propagar y refinar la información espacial entre nodos, permitiendo capturar dependencias complejas entre ellos.

"""

```
def __init__(self, num_layers, embed_dim, num_heads, ff_hidden_dim=None,  
            dropout=0.1):
```

"""

Inicializa el encoder apilando varias capas Transformer.

Args:

num_layers (int): Número de capas encoder a apilar.

embed_dim (int): Dimensión del embedding de entrada/salida.

num_heads (int): Número de cabezas de atención en cada capa.

ff_hidden_dim (int, opcional): Tamaño oculto en el bloque feedforward.

dropout (float): Proporción de dropout.

"""

```
super().__init__()
```

```
self.layers = nn.ModuleList([
```

```
TrafficformerEncoderLayer(embed_dim, num_heads, ff_hidden_dim, dropout)
```

```
for _ in range(num_layers)
```

```
])
```

```
def forward(self, x, spatial_mask=None):
```

"""

Propaga la información a través de las capas encoder.

Args:

x (torch.Tensor): Tensor de entrada [batch, num_nodes, embed_dim].

spatial_mask (torch.Tensor or None): Matriz de adyacencia espacial opcional.

Returns:

```
torch.Tensor: Tensor de salida tras todas las capas encoder [batch,  
num_nodes, embed_dim].
```

"""

```
for layer in self.layers:
```

```
x = layer(x, spatial_mask=spatial_mask)
return x # [batch, num_nodes, embed_dim]

class SpeedPredictorMLP(nn.Module):
    """
    MLP para predecir el flujo o velocidad final a partir del embedding
    espaciotemporal de cada nodo.

    Este bloque implementa un perceptrón simple con normalización y activación ReLU.
    """

    def __init__(self, embed_dim, hidden_dim=128):
        """
        Inicializa el predictor final.

        Args:
            embed_dim (int): Dimensión del embedding de entrada.
            hidden_dim (int, opcional): Dimensión de la capa oculta (por defecto 128).
        """
        super().__init__()

        self.linear1 = nn.Linear(embed_dim, hidden_dim)
        self.norm1 = nn.LayerNorm(hidden_dim)
        self.relu = nn.ReLU()

        self.linear2 = nn.Linear(hidden_dim, 1) # salida escalar por nodo

    def forward(self, x):
        """
        Genera la predicción final para cada nodo.

        Args:
            x (torch.Tensor): Tensor de entrada [batch, num_nodes, embed_dim].
        Returns:
            torch.Tensor: Tensor de salida [batch, num_nodes], predicción por nodo.
        """
        # x: [batch, num_nodes, embed_dim]
        x = self.linear1(x)
        x = self.norm1(x)
```

```
x = self.relu(x)
x = self.linear2(x)
return x.squeeze(-1) # [batch, num_nodes]

class Trafficformer(nn.Module):
"""
Implementación completa del modelo Trafficformer para predicción de tráfico basada
en Transformers.
```

Pipeline de procesamiento:

1. Extracción de embedding temporal por nodo (MLP mejorado).
2. Interacción espaciotemporal mediante encoder Transformer (máscara espacial opcional).
3. Predicción de velocidad o flujo por sensor mediante MLP final.

Este modelo es totalmente configurable en número de capas, dimensiones de embedding y arquitectura interna. Puede adaptarse fácilmente a tareas de predicción de tráfico con distintos sensores y ventanas temporales.

"""

```
def __init__(self,
            seq_len,
            num_features,
            embedding_dim,
            num_heads,
            num_layers,
            ff_hidden_dim=None,
            dropout=0.1,
            ):
    """
    Inicializa Trafficformer.
```

Args:

seq_len (int): Longitud de la ventana temporal histórica.
num_features (int): Número de variables por paso temporal.
embedding_dim (int): Dimensión del embedding intermedio.
num_heads (int): Número de cabezas de atención en cada capa encoder.
num_layers (int): Número de capas Transformer en el encoder.

```
    ff_hidden_dim (int, opcional): Tamaño oculto en la parte feedforward de
        cada encoder.

    dropout (float): Proporción de dropout.

    """
    super().__init__()
    self.temporal_extractor = TrafficTemporalFeatureExtractor(
        seq_len=seq_len,
        num_features=num_features,
        embedding_dim=embedding_dim,
        hidden_dims=(embedding_dim, embedding_dim), # ejemplo configurable
        dropout=dropout
    )
    self.encoder = TrafficformerEncoder(
        num_layers=num_layers,
        embed_dim=embedding_dim,
        num_heads=num_heads,
        ff_hidden_dim=ff_hidden_dim,
        dropout=dropout
    )
    self.predictor = SpeedPredictorMLP(embed_dim=embedding_dim,
                                        hidden_dim=embedding_dim)

    def forward(self, x, spatial_mask=None):
        """
        Realiza la pasada completa del modelo: extracción temporal, interacción
        espacial y predicción.

    Args:
        x (torch.Tensor): Tensor de entrada [batch, num_nodes, seq_len,
                           num_features].
        spatial_mask (torch.Tensor or None): Matriz de adyacencia espacial opcional
                                             [num_nodes, num_nodes].
    Returns:
        torch.Tensor: Tensor de predicción final [batch, num_nodes], estimaciones
                     para cada sensor/nodo.

        """
        # x: [batch, num_nodes, seq_len, num_features]
        temp_embed = self.temporal_extractor(x) # [batch, num_nodes, embedding_dim]
```

```
st_embed = self.encoder(temp_embed, spatial_mask) # [batch, num_nodes,  
embedding_dim]  
out = self.predictor(st_embed) # [batch, num_nodes]  
return out
```

Anexo E – Listado detallado de combinaciones por modelo

Este anexo recoge todas las combinaciones de hiperparámetros evaluadas durante los experimentos de entrenamiento, tanto para el modelo base MLP como para el modelo avanzado Trafficformer. Se han definido y ejecutado un total de 120 configuraciones distintas, distribuidas de forma equitativa entre tres fuentes de datos (sourceId 1, 2 y 5). Cada configuración representa una combinación específica de parámetros como la longitud de la secuencia de entrada, tasa de aprendizaje, tamaño del batch, número de capas, dimensiones del embedding, entre otros.

MLP (por sourceld)

En el caso del modelo MLP, se han explorado ocho combinaciones por cada sourceId, con variaciones en tres hiperparámetros clave: la longitud de la ventana temporal (seq_len), la tasa de aprendizaje (learning_rate) y el tamaño del batch (batch_size). Esto se ve en la Tabla 20.

Tabla 20: Combinaciones evaluadas para el modelo MLP (por cada sourceId)

NN	seq_len	learning_rate	batch_size
01	4	0.001	32
02	4	0.001	64
03	4	0.0005	32
04	4	0.0005	64
05	8	0.001	32
06	8	0.001	64
07	8	0.0005	32
08	8	0.0005	64

Trafficformer (por sourceld)

En el caso del modelo Trafficformer, se han diseñado 32 combinaciones por cada sourceId, contemplando una variedad mucho mayor de hiperparámetros. Las variables analizadas inclu-

yen además de las anteriores, el número de cabezas de atención (`num_heads`), la dimensión del embedding (`embedding_dim`), el número de capas (`num_layers`) y la dimensión oculta del bloque feedforward (`ff_hidden_dim`). Estas combinaciones permiten evaluar el impacto de cada configuración sobre la capacidad de generalización y aprendizaje del modelo. Esto se ve en la Tabla 22.

Tabla 22: Combinaciones evaluadas para el modelo Trafficformer (por cada sourceId)

NN	seq_len	learning_rate	batch_size	num_heads	embedding_dim	num_layers	hidden_dim	
01	4	0.001	32	4	64	4	256	
02	4	0.001	32	4	64	6	512	
03	4	0.001	32	8	128	4	256	
04	4	0.001	32	8	128	6	512	
05	4	0.001	64	4	64	4	256	
06	4	0.001	64	4	64	6	512	
07	4	0.001	64	8	128	4	256	
08	4	0.001	64	8	128	6	512	
09	4	0.0005	32	4	64	4	256	
10	4	0.0005	32	4	64	6	512	
11	4	0.0005	32	8	128	4	256	
12	4	0.0005	32	8	128	6	512	
13	4	0.0005	64	4	64	4	256	
14	4	0.0005	64	4	64	6	512	
15	4	0.0005	64	8	128	4	256	
16	4	0.0005	64	8	128	6	512	
17	8	0.001	32	4	64	4	256	
18	8	0.001	32	4	64	6	512	
19	8	0.001	32	8	128	4	256	
20	8	0.001	32	8	128	6	512	
21	8	0.001	64	4	64	4	256	
22	8	0.001	64	4	64	6	512	
23	8	0.001	64	8	128	4	256	
24	8	0.001	64	8	128	6	512	
25	8	0.0005	32	4	64	4	256	
26	8	0.0005	32	4	64	6	512	
27	8	0.0005	32	8	128	4	256	
28	8	0.0005	32	8	128	6	512	
29	8	0.0005	64	4	64	4	256	
30	8	0.0005	64	4	64	6	512	
31	8	0.0005	64	8	128	4	256	120
32	8	0.0005	64	8	128	6	512	

Anexo F – Resultados detallados de los 120 experimentos

Este anexo recoge el resultado completo de los 120 experimentos de entrenamiento realizados, abarcando diferentes modelos, fuentes de datos y combinaciones de hiperparámetros. Se incluyen, para cada experimento, los valores principales de las métricas obtenidas en validación y test: loss, mae, rmse, mape y r2, además de la epoch de mejor resultado.

La tabla siguiente muestra la información relevante de cada experimento, agrupando por fuente de datos (sourceId) y modelo (model).

Tabla 24: Métricas principales de los 120 experimentos realizados (validación y test).

ID	Src	Modelo	Ep	Val Loss	Val MAE	Val RMSE	Val MAPE	Val R2	Test Loss	Test MAE	Test RMSE	Test	Test	Test R2
1	1	MLP	21	0.002531	0.037882	0.045857	21.077439	0.964011	0.002628	0.038794	0.046621	21.545246	0.96253	
2	1	MLP	40	0.002647	0.038681	0.046977	21.463618	0.962054	0.002772	0.039857	0.047914	22.135354	0.96035	
3	1	MLP	25	0.002478	0.037261	0.045537	20.822926	0.964684	0.002567	0.038133	0.046172	21.283347	0.96335	
4	1	MLP	38	0.002452	0.037038	0.045358	20.710834	0.965002	0.002542	0.037919	0.045987	21.163511	0.96368	
5	1	MLP	32	0.002467	0.037130	0.045422	20.756244	0.964849	0.002555	0.037989	0.046049	21.184368	0.96358	
6	1	MLP	15	0.002509	0.037486	0.045644	20.920890	0.964370	0.002596	0.038565	0.046443	21.390472	0.96297	
7	1	MLP	33	0.002493	0.037332	0.045515	20.815909	0.964623	0.002584	0.038420	0.046330	21.345234	0.96314	
8	1	MLP	39	0.002462	0.037053	0.045375	20.709663	0.964982	0.002554	0.037997	0.046056	21.186427	0.96357	
9	1	Trafficformer	0.001816	0.031694	0.041054	19.023583	0.971024	0.001919	0.032790	0.041963	19.617563	0.96969		
10	1	Trafficformer	0.001847	0.031983	0.041263	19.124199	0.970653	0.001963	0.033101	0.042177	19.754355	0.96927		
11	1	Trafficformer	0.001800	0.031577	0.040959	18.952897	0.971147	0.001899	0.032687	0.041866	19.556610	0.96981		
12	1	Trafficformer	0.001873	0.032225	0.041436	19.176864	0.970508	0.001992	0.033184	0.042251	19.801366	0.96911		
13	1	Trafficformer	0.001870	0.032189	0.041414	19.152456	0.970540	0.001983	0.033121	0.042200	19.782216	0.96917		
14	1	Trafficformer	0.001871	0.032204	0.041429	19.165013	0.970527	0.001984	0.033136	0.042214	19.791249	0.96915		
15	1	Trafficformer	0.001794	0.031526	0.040918	18.936470	0.971190	0.001892	0.032638	0.041825	19.535484	0.96985		
16	1	Trafficformer	0.001848	0.031994	0.041273	19.130842	0.970641	0.001966	0.033112	0.042188	19.762462	0.96925		
17	1	Trafficformer	0.001779	0.031402	0.040814	18.858172	0.971335	0.001876	0.032527	0.041733	19.480682	0.96997		
18	1	Trafficformer	0.001862	0.032129	0.041382	19.128180	0.970583	0.001974	0.033073	0.042156	19.774651	0.96920		
19	1	Trafficformer	0.001869	0.032185	0.041409	19.156487	0.970544	0.001983	0.033126	0.042208	19.780865	0.96916		
20	1	Trafficformer	0.001810	0.031646	0.041007	18.978653	0.971079	0.001911	0.032741	0.041919	19.603344	0.96976		
21	2	MLP	20	0.002531	0.037882	0.045857	21.077439	0.964011	0.002628	0.038794	0.046621	21.545246	0.96253	
22	2	MLP	40	0.002647	0.038681	0.046977	21.463618	0.962054	0.002772	0.039857	0.047914	22.135354	0.96035	
23	2	MLP	25	0.002478	0.037261	0.045537	20.822926	0.964684	0.002567	0.038133	0.046172	21.283347	0.96335	
24	2	MLP	38	0.002452	0.037038	0.045358	20.710834	0.965002	0.002542	0.037919	0.045987	21.163511	0.96368	
25	2	MLP	32	0.002467	0.037130	0.045422	20.756244	0.964849	0.002555	0.037989	0.046049	21.184368	0.96358	
26	2	MLP	15	0.002509	0.037486	0.045644	20.920890	0.964370	0.002596	0.038565	0.046443	21.390472	0.96297	
27	2	MLP	33	0.002493	0.037332	0.045515	20.815909	0.964623	0.002584	0.038420	0.046330	21.345234	0.96314	
28	2	MLP	39	0.002462	0.037053	0.045375	20.709663	0.964982	0.002554	0.037997	0.046056	21.186427	0.96357	
29	2	Trafficformer	0.001816	0.031694	0.041054	19.023583	0.971024	0.001919	0.032790	0.041963	19.617563	0.96969		

Continúa en la siguiente página

Tabla 24 – continuación de la página anterior

ID	Src	Modelo	Ep	Val Loss	Val MAE	Val RMSE	Val	Val R2	Test Loss	Test MAE	Test	Test	Test R2
							MAPE			RMSE	MAPE		
116	5	Trafficformer	69	0.002216	0.035144	0.047057	20.633137	0.967294	0.002342	0.036330	0.048102	21.277395	0.96579
117	5	Trafficformer	70	0.002175	0.034851	0.046804	20.458176	0.967777	0.002288	0.035968	0.047753	21.058676	0.96633
118	5	Trafficformer	71	0.002173	0.034831	0.046793	20.449985	0.967802	0.002285	0.035950	0.047741	21.049797	0.96635
119	5	Trafficformer	72	0.002170	0.034817	0.046782	20.442479	0.967820	0.002283	0.035937	0.047729	21.041914	0.96637
120	5	Trafficformer	73	0.002187	0.034954	0.046887	20.500823	0.967654	0.002302	0.036074	0.047838	21.103521	0.96621

Anexo G – Análisis avanzado por fuente de datos

Este anexo contiene las gráficas avanzadas complementarias para el análisis exhaustivo realizado sobre los mejores modelos entrenados con cada fuente de datos (sourceId 1, 2 y 5). Estas gráficas permiten identificar visualmente patrones específicos, errores sistemáticos, posibles sesgos y áreas potenciales de mejora para futuros desarrollos del modelo.

SOURCEID 1

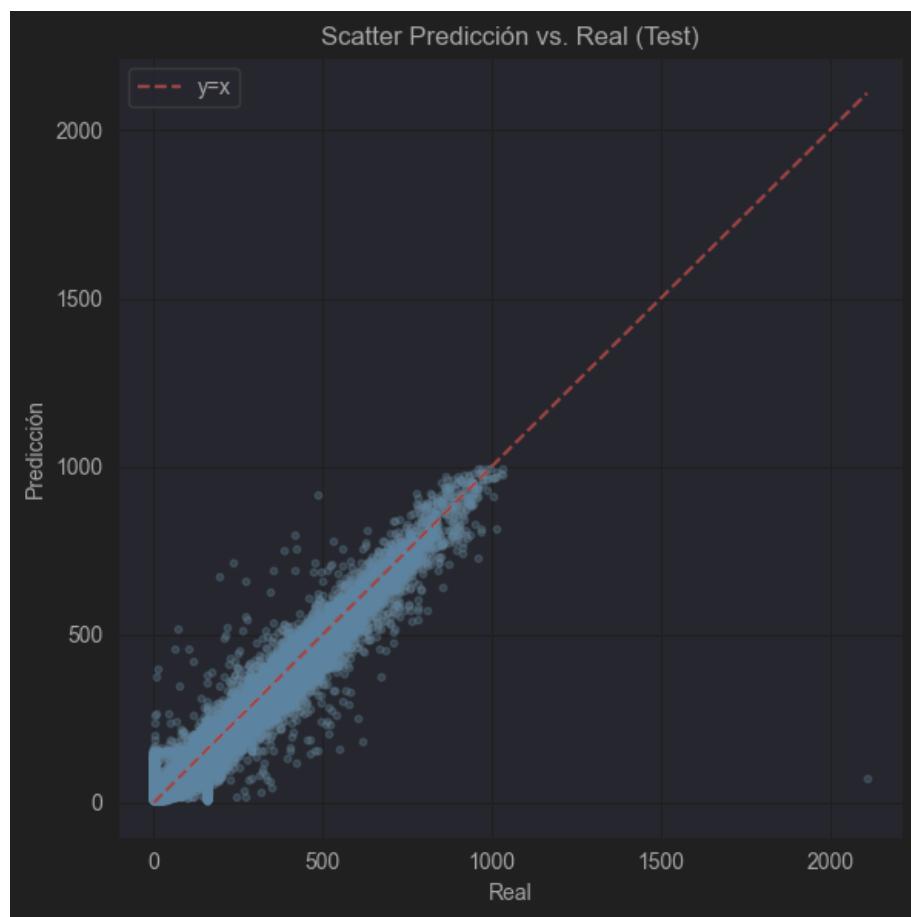


Figura 24: Gráfico de dispersión (predicción vs. valores reales) para SOURCEID 1.

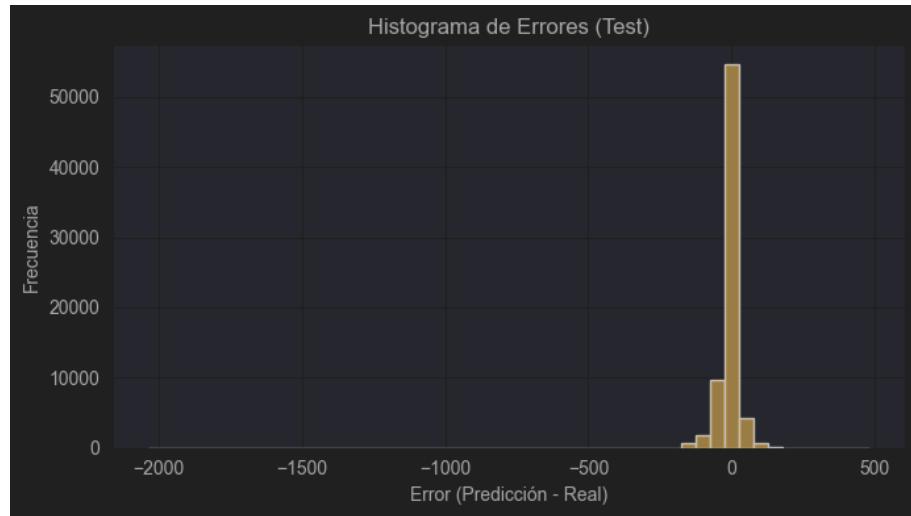


Figura 25: Histograma del error absoluto (residual) para Sourceld 1.

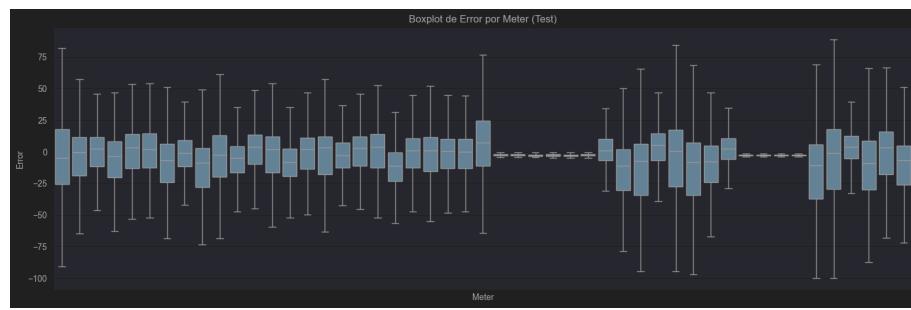


Figura 26: Boxplot del error absoluto por sensor para todos los sensores del Sourceld 1.

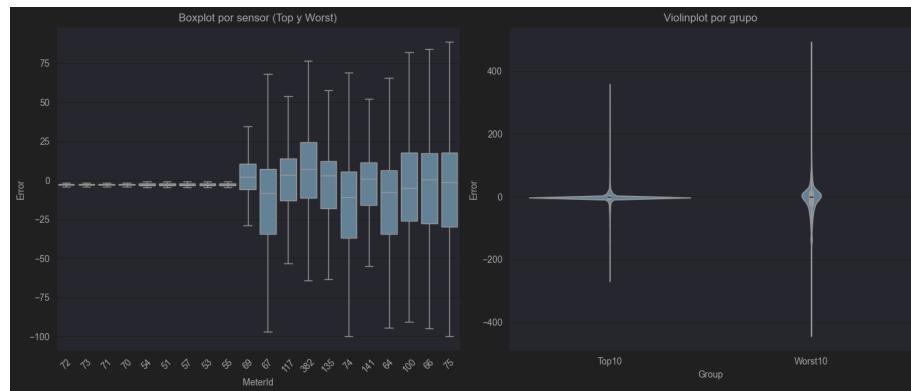


Figura 27: Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 1.

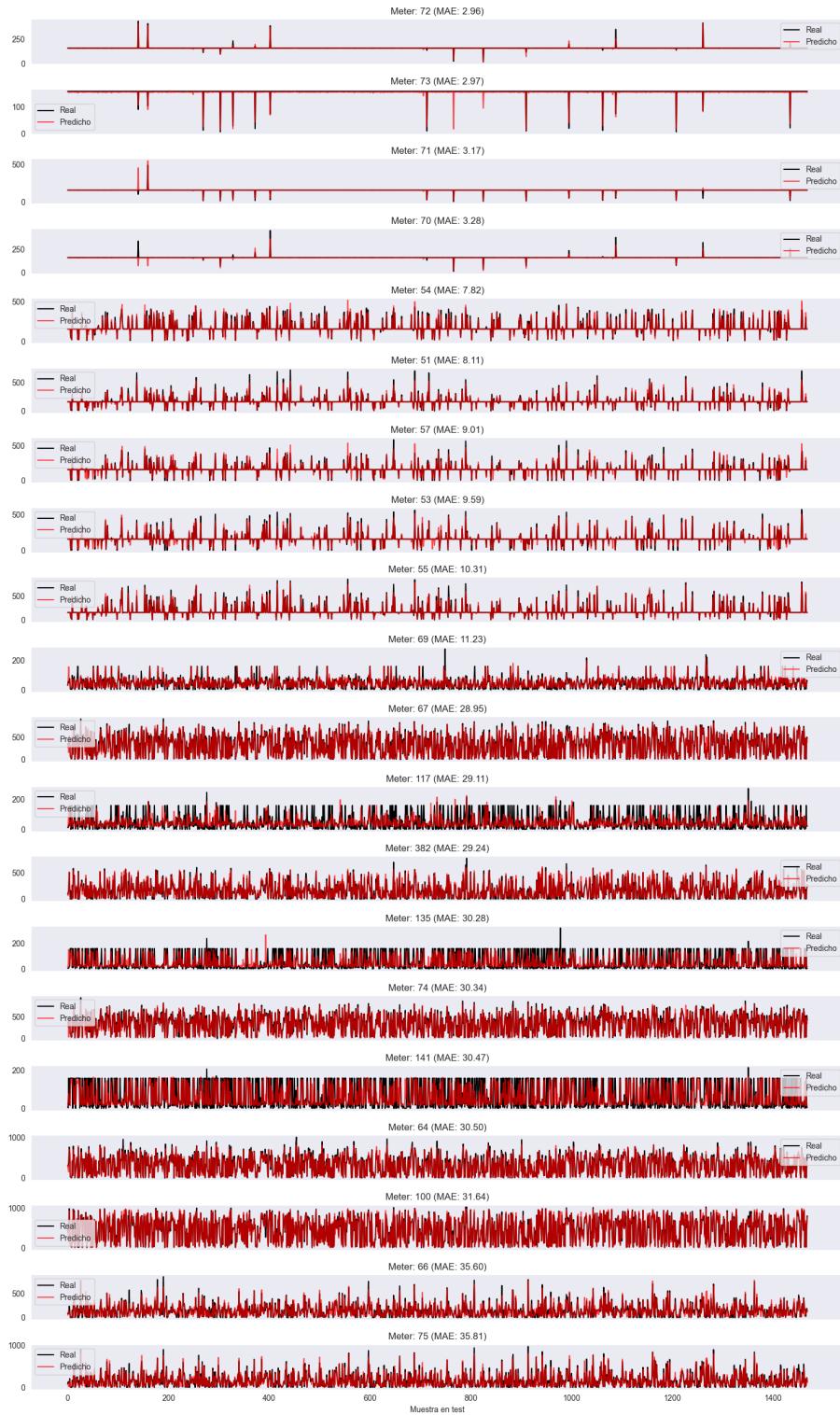


Figura 28: Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 1.

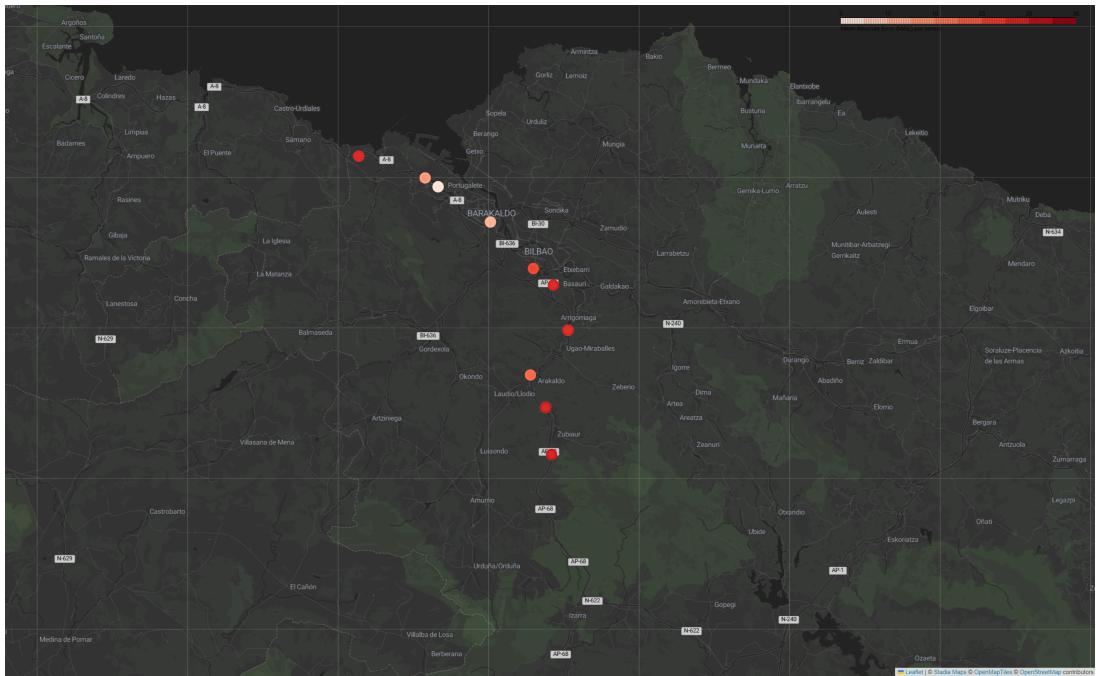


Figura 29: Mapa espacial de errores (MAE) para sensores del SourceID 1.

Sourceld 2

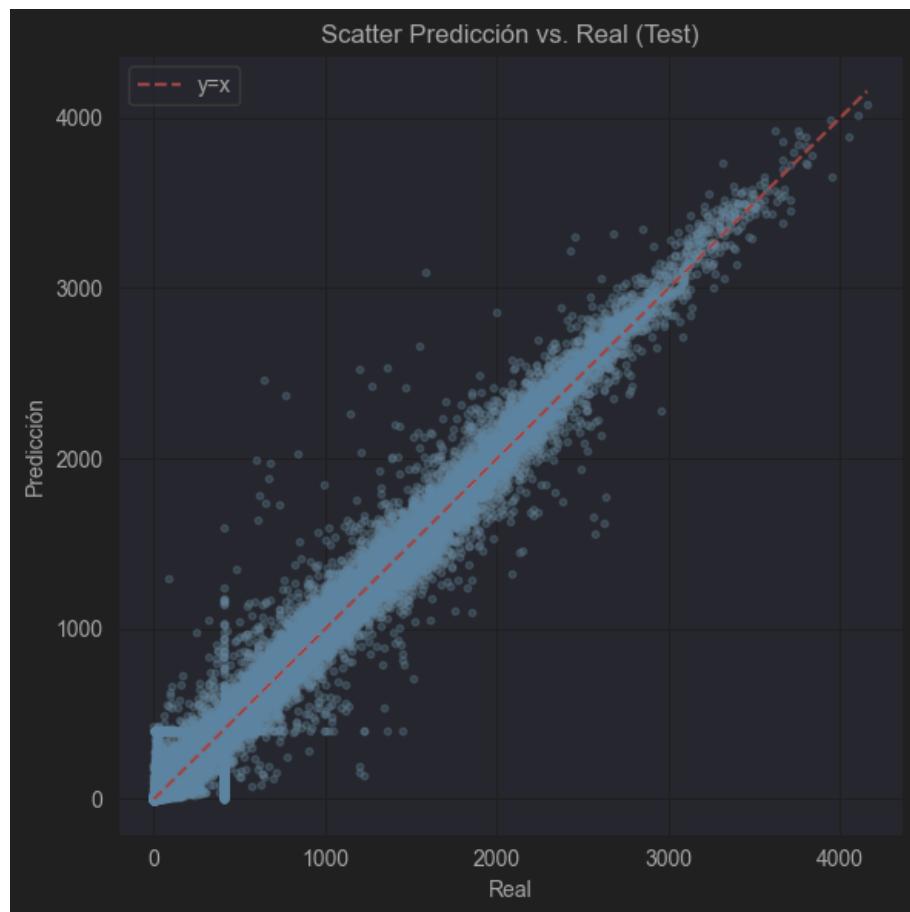


Figura 30: Gráfico de dispersión (predicción vs. valores reales) para Sourceld 2.

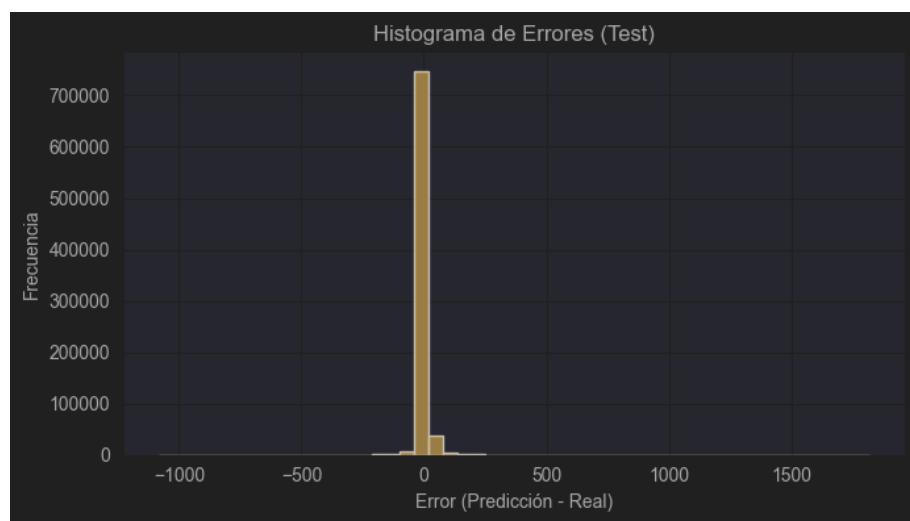


Figura 31: Histograma del error absoluto (residual) para Sourceld 2.

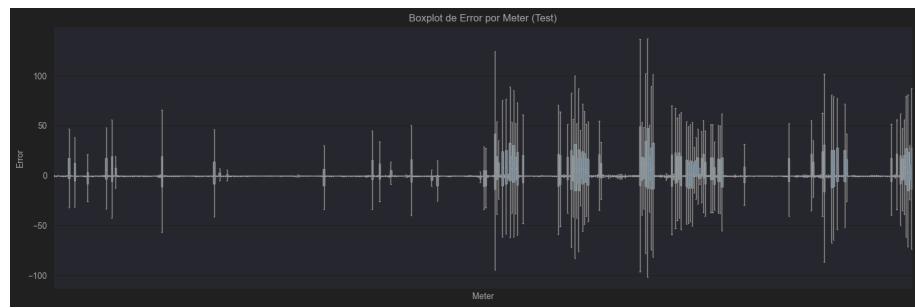


Figura 32: Boxplot del error absoluto por sensor para todos los sensores del Sourceld 2.

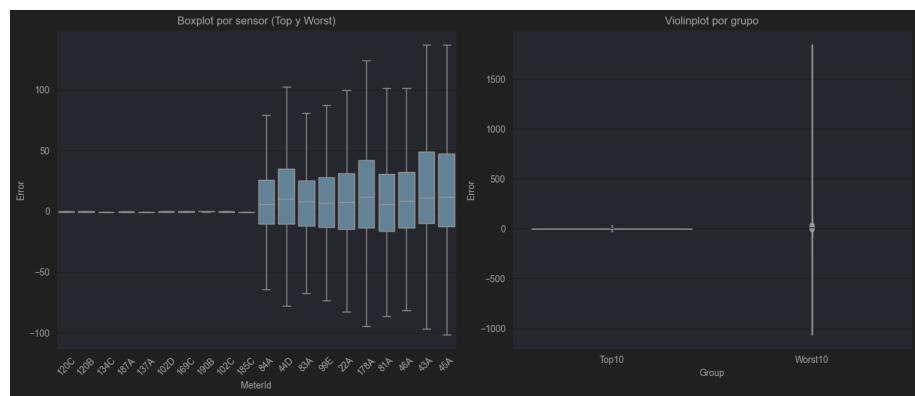


Figura 33: Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 2.

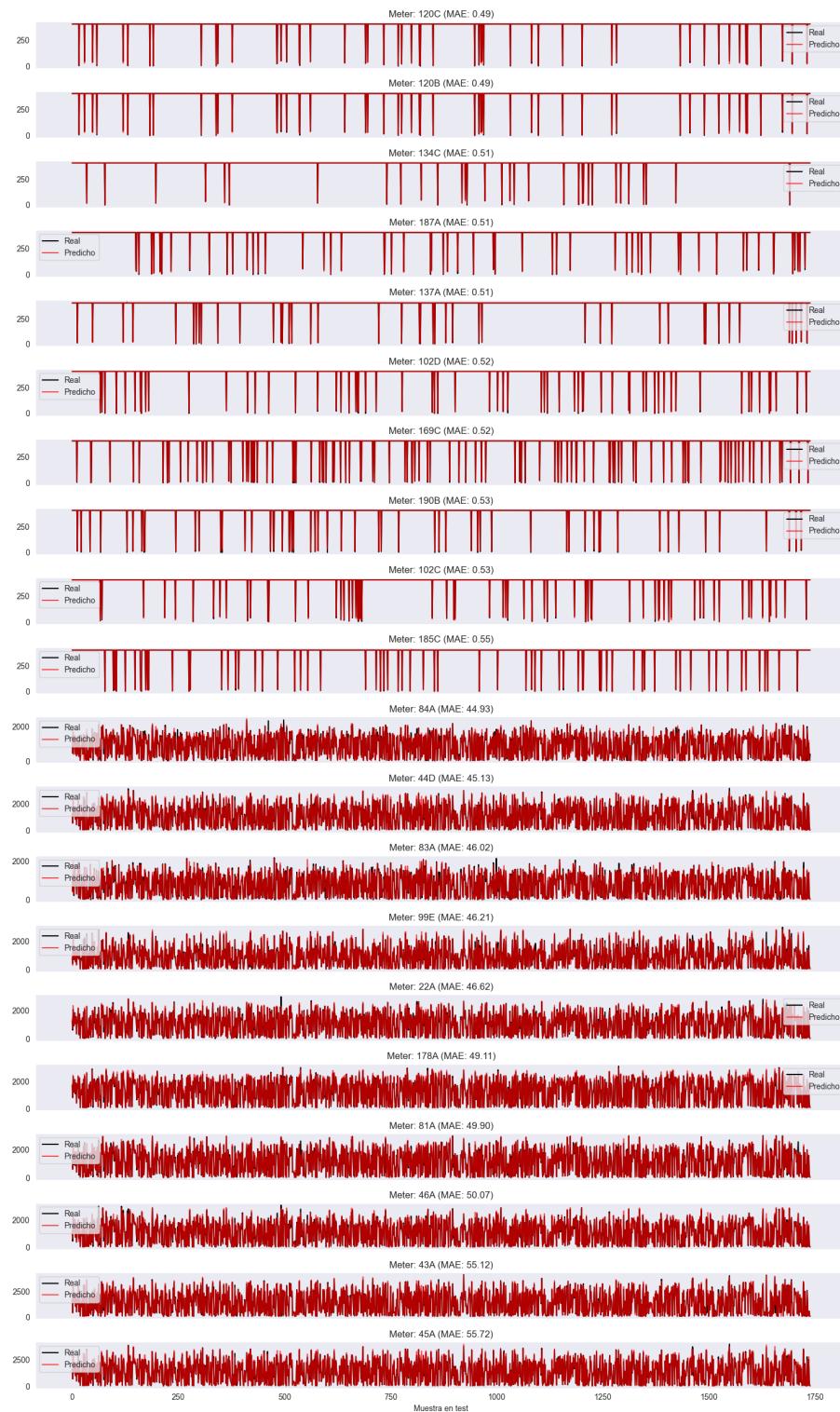


Figura 34: Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 2.

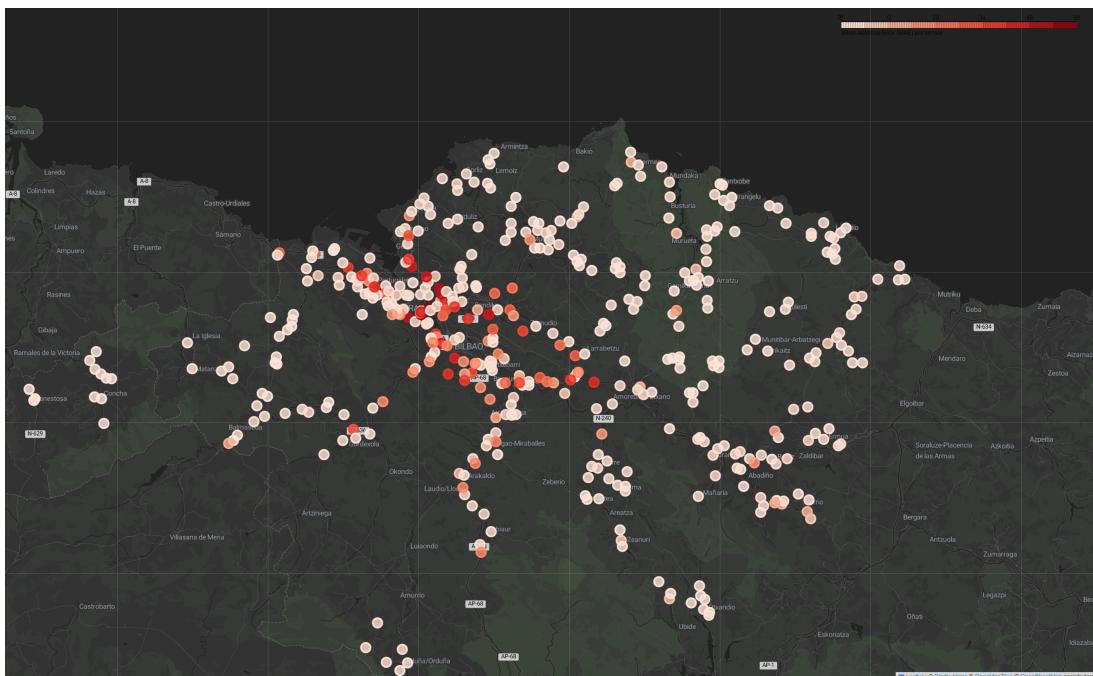


Figura 35: Mapa espacial de errores (MAE) para sensores del Sourceld 2.

Sourceld 5

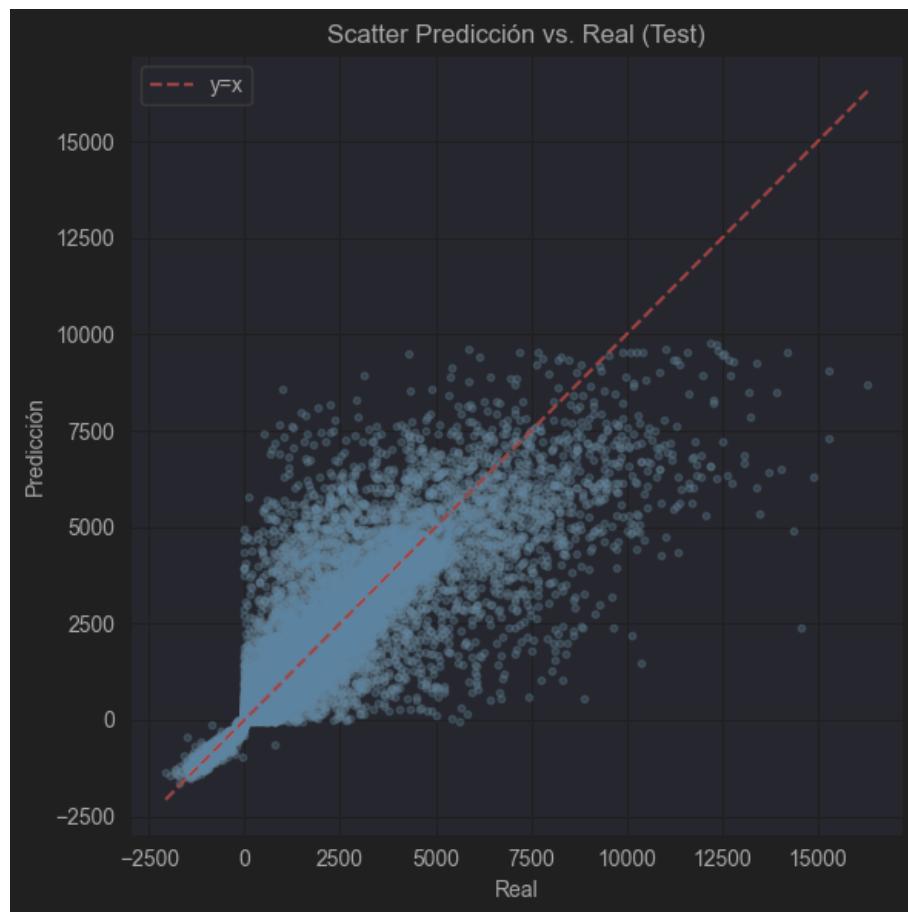


Figura 36: Gráfico de dispersión (predicción vs. valores reales) para Sourceld 5.

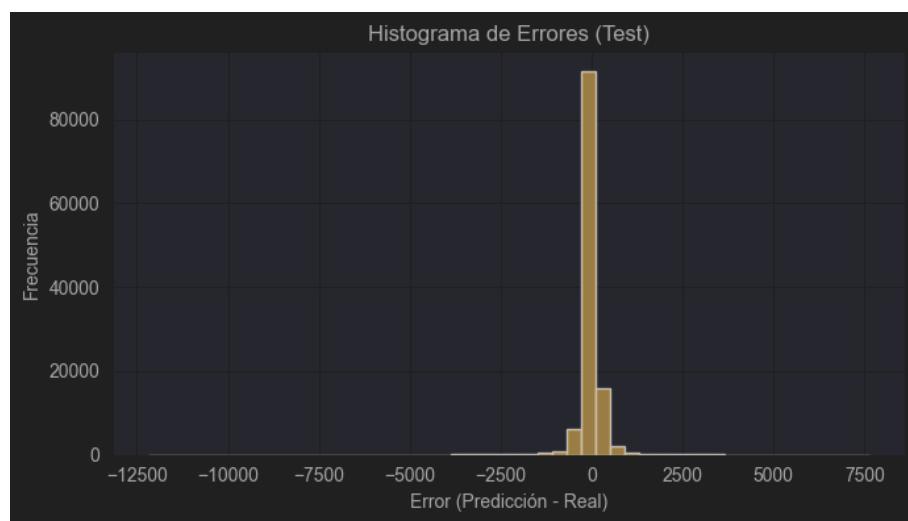


Figura 37: Histograma del error absoluto (residual) para Sourceld 5.

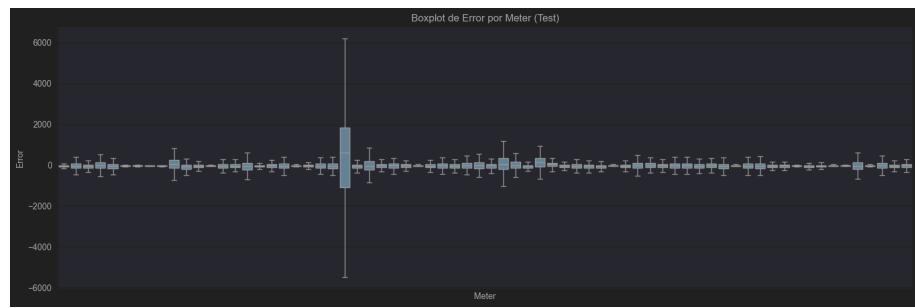


Figura 38: Boxplot del error absoluto por sensor para todos los sensores del Sourceld 5.



Figura 39: Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 5.

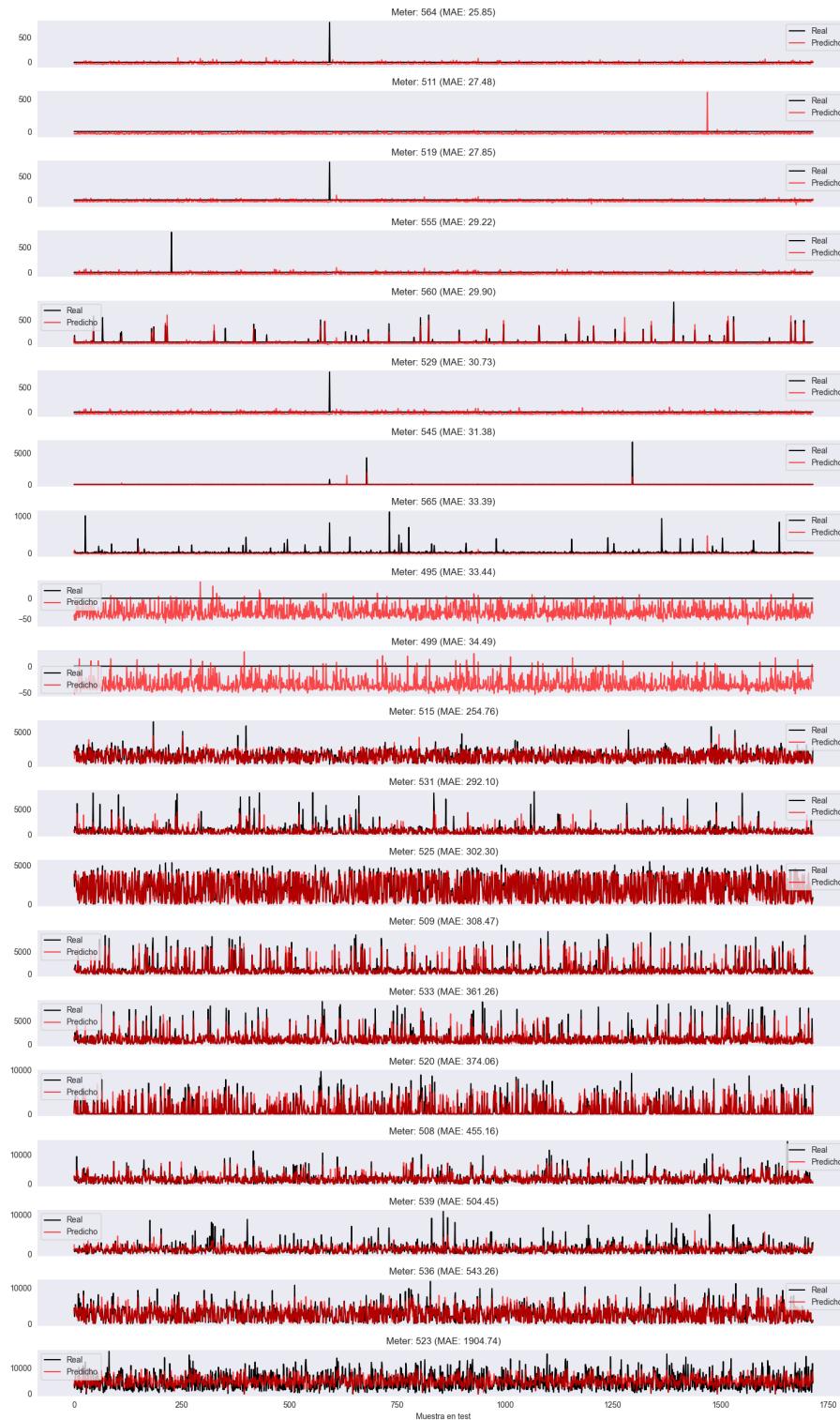


Figura 40: Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 5.

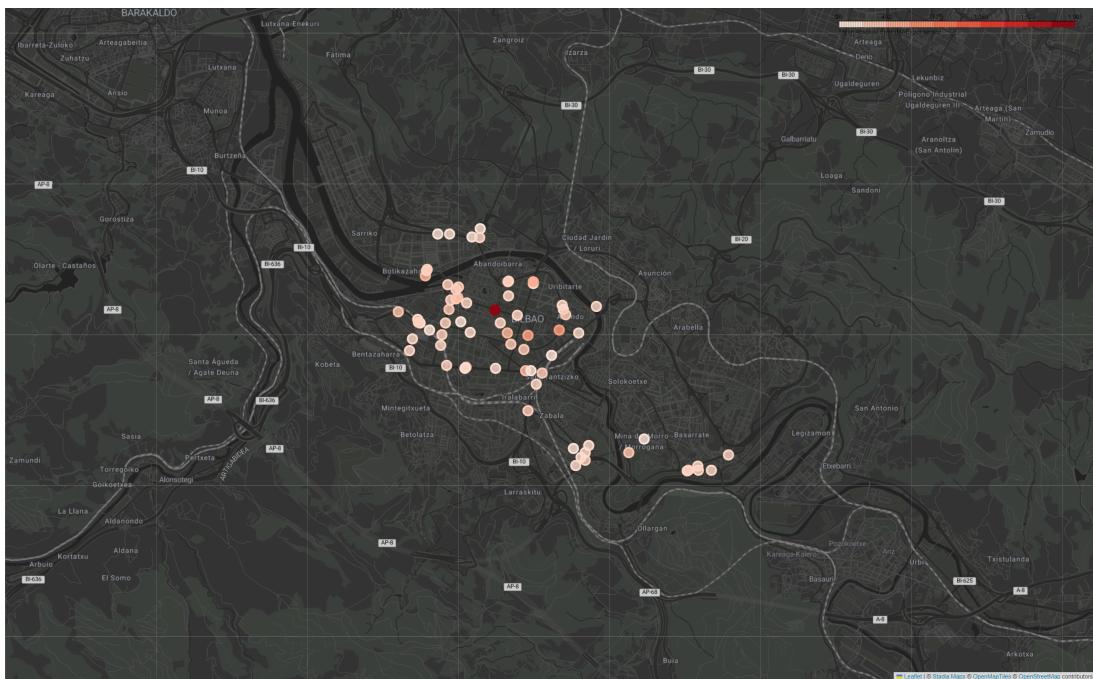


Figura 41: Mapa espacial de errores (MAE) para sensores del SourceID 5.