



Universidad Internacional de la Rioja (UNIR)

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Inteligencia Artificial

Predicción de tráfico mediante
aprendizaje profundo y Trans-
formers

Trabajo Fin de Estudios

Presentado por: Jon Inazio Sánchez Martínez

Dirigido por: Omar Velázquez López

Ciudad: Bilbao

Fecha: 23 de julio 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 una arquitectura basada en transformers para predicción espaciotemporal del tráfico, 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, se 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, a transformer based architecture for spatiotemporal traffic prediction, 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

Agradecimientos

La culminación de este Trabajo Fin de Máster ha sido posible gracias al apoyo, comprensión y aliento de muchas personas a lo largo de este proceso. En primer lugar, quiero expresar mi más sincero agradecimiento a mi familia, pareja y amigos, por su constante apoyo, paciencia y comprensión durante todos estos meses de esfuerzo. También deseo agradecer a mis compañeros de trabajo, quienes han contribuido de manera directa e indirecta con sus consejos, motivación y ayuda práctica en diferentes fases del desarrollo de este proyecto.

Mi agradecimiento especial al director de este TFM, por su dedicación, orientación y acompañamiento a lo largo de todo el proceso, facilitando siempre el camino y guiando el enfoque de la investigación con rigor y profesionalidad.

A todas y a todos, mi más sincero agradecimiento.

Licencia



<https://creativecommons.org/licenses/by/4.0/>

Salvo que se indique expresamente lo contrario en cada repositorio o recurso, tanto el código fuente como los documentos, anexos y materiales generados en el contexto de este Trabajo Fin de Máster se distribuyen bajo las siguientes condiciones:

- **Código fuente:** publicado bajo licencia MIT. Se permite su uso, copia, modificación y distribución, con o sin modificaciones, siempre que se mantenga la atribución al autor original.
- **Documentación y memoria:** publicada bajo licencia Creative Commons Attribution 4.0 International (CC BY 4.0), que permite su uso, distribución y reproducción en cualquier medio, siempre que se cite adecuadamente la autoría original.

Para más detalles sobre las condiciones de uso, puede consultarse el archivo LICENSE.md incluido en cada uno de los repositorios públicos mencionados en el Anexo A de este documento.

© 2025 Jon Inazio Sánchez Martínez. Todos los derechos reservados, salvo indicación expresa.

Índice de contenidos

Resumen	i
Abstract	ii
Agradecimientos	iii
Licencia	iii
Índice de figuras	vi
Índice de tablas	viii
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	12
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
3.4 Fundamentos matemáticos del modelo Trafficformer	21
4 Construcción del dataset y arquitectura del sistema	24
4.1 Fuentes de datos y análisis de disponibilidad	24
4.2 Recolección, arquitectura y patrones de diseño empleados	32
4.3 Decisiones de construcción del dataset	34
4.4 Arquitectura y pipeline general del proyecto	43
5 Desarrollo experimental, comparación de modelos y resultados	45
5.1 Planteamiento experimental y justificación	45
5.2 Entorno de desarrollo y reproducibilidad	47
5.3 Preparación de datos y pipeline	51
5.4 Preparación y tratamiento del dataset	51
5.5 Descripción de los modelos	57

5.6 Diseño experimental y combinaciones	62
5.7 Proceso de entrenamiento y selección de mejores modelos	64
6 Evaluación, comparación de modelos y resultados	68
6.1 Resultados experimentales	68
6.2 Discusión y análisis crítico	76
6.3 Análisis avanzado de resultados por fuente de datos	79
7 Conclusiones y trabajo futuro	83
7.1 Síntesis global, verificación de la hipótesis y aportación al campo	83
7.2 Conclusiones principales por fuente de datos	84
7.3 Limitaciones y validez de los experimentos	86
7.4 Trabajo Futuro	87
Bibliografía	90
Anexo A – Direcciones a los repositorios del TFM	95
Anexo B – Descripción de sensores meteorológicos	96
Anexo C – Código fuente del generador de snapshots	98
Anexo D – Plantilla de infraestructura para AWS	104
Anexo E – Código fuente de la arquitectura Trafficformer	110
Anexo F – Listado detallado de combinaciones por modelo	122
Anexo G – Resultados detallados de los 120 experimentos	125
Anexo H – Análisis avanzado por fuente de datos	131

Índice de figuras

1	Red viaria de la CAPV	2
2	Arquitectura general del modelo Trafficformer Chang et al. (2025)	11
3	Diagrama de arquitectura y pipeline general del proyecto.	19
4	Ubicación de aforos de la fuente Gobierno Vasco (source_id = 1)	26
5	Ubicación de aforos de la fuente Diputación Foral de Bizkaia (source_id = 2) . .	26
6	Ubicación de aforos de la fuente Ayuntamiento de Bilbao (source_id = 5) . . .	27
7	Ejemplo de error a la hora de consultar el API de Euskalmet.	27
8	Modelo de clases en UML de las entidades principales del sistema.	28
9	Diagrama de secuencia de integración y persistencia en MobilitySnapshot. .	39
10	Distribución espacial de sensores para Gobierno Vasco (sourceld 1).	55
11	Distribución espacial de sensores para Diputación Foral de Bizkaia (sourceld 2).	55
12	Distribución espacial de sensores para Ayuntamiento de Bilbao (sourceld 5). .	56
13	Esquema lógico del flujo de capas en el modelo MLP.	58
14	Aplicación de la máscara espacial en el mecanismo de atención de Trafficformer.	60
15	Esquema lógico del flujo de capas del modelo Trafficformer.	61
16	Curvas de evolución de pérdida para entrenamiento y validación, con indicación de la mejor época.	66
17	Curvas de entrenamiento para el modelo MLP con datos del Gobierno Vasco (sourceld 1).	70
18	Curvas de entrenamiento para el modelo Trafficformer con datos del Gobierno Vasco (sourceld 1).	71
19	Curvas de entrenamiento para el modelo MLP con datos de la Diputación Foral de Bizkaia (sourceld 2).	72
20	Curvas de entrenamiento para el modelo Trafficformer con datos de la Diputación Foral de Bizkaia (sourceld 2).	73
21	Curvas de entrenamiento para el modelo MLP con datos del Ayuntamiento de Bilbao (sourceld 5).	74
22	Curvas de entrenamiento para el modelo Trafficformer con datos del Ayuntamiento de Bilbao (sourceld 5).	75
23	Gráfico de dispersión (predicción vs. valores reales) para Sourceld 1.	131
24	Histograma del error absoluto (residual) para Sourceld 1.	132
25	Boxplot del error absoluto por sensor para todos los sensores del Sourceld 1. .	132
26	Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 1.	132

27	Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 1.	133
28	Mapa espacial de errores (MAE) para sensores del Sourceld 1.	134
29	Gráfico de dispersión (predicción vs. valores reales) para Sourceld 2.	135
30	Histograma del error absoluto (residual) para Sourceld 2.	135
31	Boxplot del error absoluto por sensor para todos los sensores del Sourceld 2.	136
32	Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 2.	136
33	Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 2.	137
34	Mapa espacial de errores (MAE) para sensores del Sourceld 2.	138
35	Gráfico de dispersión (predicción vs. valores reales) para Sourceld 5.	139
36	Histograma del error absoluto (residual) para Sourceld 5.	139
37	Boxplot del error absoluto por sensor para todos los sensores del Sourceld 5.	140
38	Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 5.	140
39	Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 5.	141
40	Mapa espacial de errores (MAE) para sensores del Sourceld 5.	142

Índice de tablas

1	Comparativa entre arquitecturas en tareas de predicción del tráfico.	13
2	Notación de los principales símbolos utilizados en la formulación de Trafficformer.	23
3	Cobertura de datos por fuente y tipo.	25
4	Frecuencia de incidencias por tipo original	35
5	Agrupación final de incidencias para el modelo predictivo	36
6	Selección de sensores meteorológicos y su relevancia para la predicción de tráfico	37
7	Comparativa entre intervalos temporales posibles para el resampling	42
8	Comparación conceptual entre los modelos MLP y Trafficformer	46
9	Configuraciones experimentales evaluadas	63
10	Resultados de los mejores modelos por combinación sourceId–arquitectura .	68
11	Comparativa de resultados de Trafficformer respecto al estado del arte.	81
12	Sensores Euskalmet empleados en la construcción del dataset.	96
13	Combinaciones evaluadas para el modelo MLP (por cada sourceId)	122
14	Combinaciones evaluadas para el modelo Trafficformer (por cada sourceId)	123
15	Métricas principales de los 120 experimentos realizados (validación y test). . .	125

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.

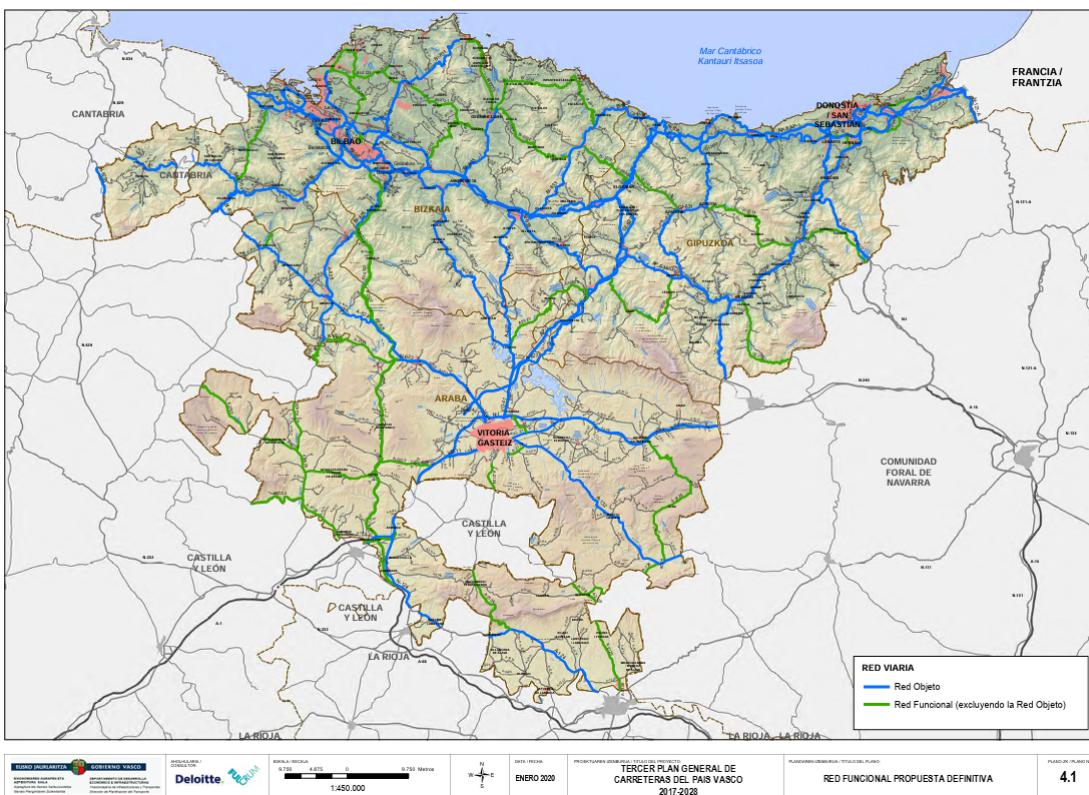
En este escenario, la predicción del tráfico emerge como una herramienta clave para anticiparse a situaciones de congestión, disruptores u otros eventos que puedan comprometer la fluidez de la circulación. La inteligencia Artificial y, en particular, del Aprendizaje Profundo, ha impulsado el desarrollo 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.

Figura 1: Red viaria de la CAPV



Fuente: Gobierno Vasco, Eusko Jaurlaritza (2020). *Tercer Plan General de Carreteras del País Vasco 2017-2028*.

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, 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

tura transformer. La elección de este enfoque se fundamenta en los resultados recientes de la literatura, donde modelos como Trafficformer han demostrado una notable capacidad para capturar dependencias espaciotemporales complejas en datos de tráfico y obtener mejores resultados que otras arquitecturas profundas tradicionales, gracias al mecanismo de atención y la integración de información topológica de la red viaria Chang et al. (2025). Esta evidencia convierte a los Transformers en una opción idónea para abordar la predicción de tráfico con alta precisión en entornos urbanos complejos.

El modelo se alimentará de datos abiertos sobre tráfico y meteorología, publicados por organismos 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; el capítulo 4 muestra la construcción del dataset y presenta la arquitectura; el capítulo 5 describe el software desarrollado; en el capítulo 6 se muestra la evaluación y discuten los resultados obtenidos; acabando con el capítulo 7 donde se detallan las conclusiones y el trabajo futuro.

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

En la literatura se han desarrollado diversas técnicas para la predicción del tráfico, desde modelos estadísticos clásicos hasta enfoques basados en aprendizaje profundo. Entre las principales categorías se encuentran:

- **Modelos estadísticos lineales:** como AutoRegressive Integrated Moving Average, Kalman Filter o regresiones lineales, empleados tradicionalmente en la predicción de series temporales de tráfico Kumar & Vanajakshi (2015); Katambire et al. (2023); Momin et al. (2023); Liu et al. (2020).
- **Modelos basados en Support Vector Regression:** Support Vector Regression, Random Forests, K-Nearest Neighbors, que han mostrado eficacia en contextos con datos limitados y estructuras menos complejas Omar et al. (2024); Wu (2024); Aditya et al. (2020).

- **Redes neuronales profundas:**

- Recurrent Neural Networks o Redes Neuronales Recurrentes, y variantes como Long Short-Term Memory y Gated Recurrent Units, diseñadas para capturar dependencias temporales a corto y largo plazo en series de tráfico Hochreiter & Schmidhuber (1997); Cho et al. (2014); Zhao et al. (2017); Ma et al. (2023).
- Graph Neural Networks, incluyendo Graph Convolutional Networks y Graph Attention Networks (adaptadas a redes viarias), que permiten modelar explícitamente la topología de la red viaria y las relaciones entre distintos nodos Khan et al. (2023); Chen et al. (2021).
- Modelos basados en transformer, que han demostrado una capacidad sobresaliente para captar dependencias espaciotemporales y superar limitaciones de modelos anteriores Chang et al. (2025).

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 identificar 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

Las redes neuronales recurrentes (Recurrent Neural Networks o Redes Neuronales Recurrentes (RNN)) han demostrado gran eficacia en la predicción de series temporales en el ámbito del tráfico, gracias a su capacidad para modelar dependencias a largo plazo mediante estructuras como las LSTM Hochreiter & Schmidhuber (1997) y las GRU Cho et al. (2014). Diversos estudios han validado su aplicabilidad en este campo, destacando resultados como los obtenidos por Zhao et al. con LSTM, logrando un error relativo medio (MRE) del 6,41 % en predicción de flujo de tráfico a corto plazo Zhao et al. (2017), o la propuesta de Ma et al., que combinan CNN y GRU para alcanzar un MAPE del 8,60 % en predicción de velocidad Ma et al. (2023). Estas evidencias posicionan a las RNN y sus variantes como alternativas robustas frente a métodos tradicionales, especialmente en tareas de predicción a corto plazo.

Graph Neural Networks

Las Graph Neural Networks han permitido extender el aprendizaje profundo a datos no euclidianos como los grafos, resultando especialmente adecuadas para modelar redes de carreteras en sistemas inteligentes de transporte Scarselli et al. (2009). Estas arquitecturas, como las Graph Convolutional Networks (GCN), introducidas en Kipf & Welling (2017), Graph Attention Networks (GAT), introducidas en Veličković et al. (2018) y las Gated Graph Neural Networks (GGNN), destacan por su capacidad para capturar dependencias espaciales y temporales entre nodos e incorporar mecanismos avanzados como la atención o puertas de control. Diversos estudios han demostrado su efectividad en la predicción del flujo y la densidad del tráfico; por ejemplo, el trabajo de Khan et al. (2023) evidencia que las GGNN superan a otras arquitecturas en precisión, mientras que propuestas como TrafficStream Chen et al. (2021) combinan GNN con aprendizaje continuo para adaptarse a cambios dinámicos en los patrones de tráfico.

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.

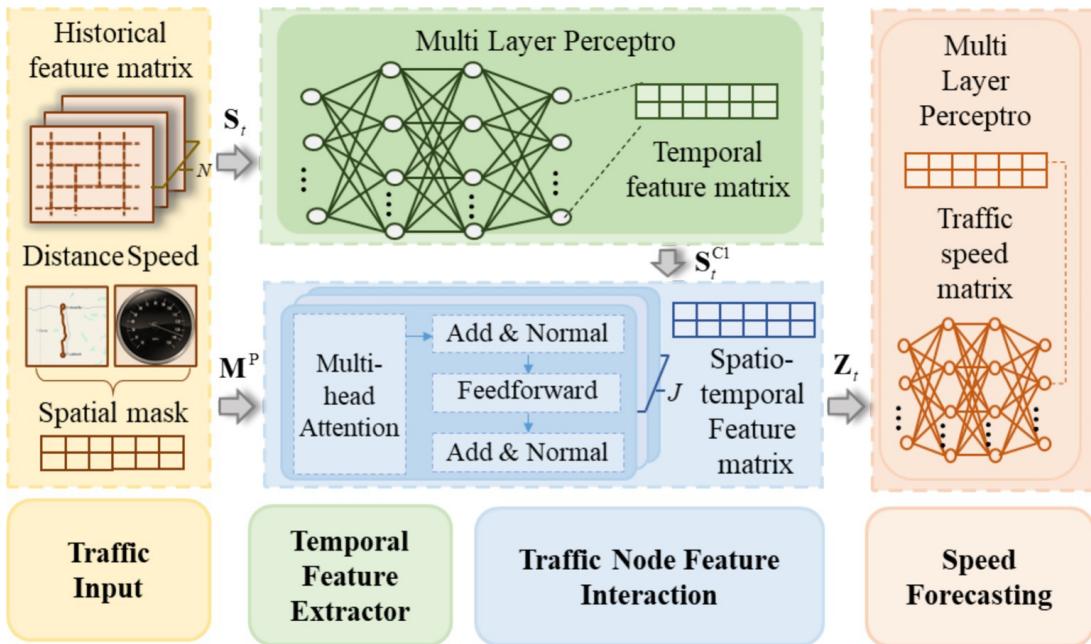
La clave de los Transformers reside en el **multi-head self-attention**, que otorga al modelo la capacidad de ponderar dinámicamente la importancia relativa de distintos elementos dentro de una secuencia. Este mecanismo, además de ofrecer un rendimiento computacional más eficiente, mejora la capacidad 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 traffic 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.

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



Fuente: Chang et al. (2025)

En cuanto a la arquitectura, el modelo Trafficformer (véase Figura 2) 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 paralelización 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)	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 fuerte, pero poco eficiente para relaciones espaciales	Excelente modelado espacial, dificultad para integrar dinámica temporal	Modelado explícito y desacoplado de 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

Fuente: Elaboración propia.

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.

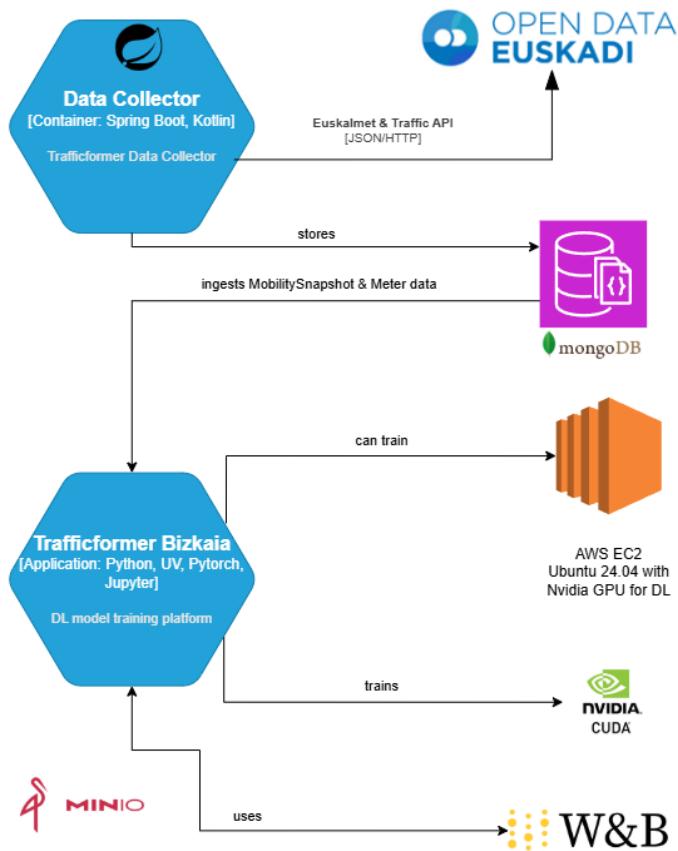
Estas decisiones se ven respaldadas por análisis comparativos recientes, como el realizado por DataCamp, donde se destaca que *PyTorch* suele ser más rápido y proporciona mejores capacidades de depuración que *Keras*. Estas características lo hacen especialmente adecuado para entornos de investigación 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 siguiendo un flujo de trabajo organizado en varias fases clave.

Figura 3: Diagrama de arquitectura y pipeline general del proyecto.



Fuente: Elaboración propia.

Este esquema proporciona una visión clara y sintetizada del flujo de trabajo, facilitando la comprensión global y la replicabilidad de las metodologías empleadas a lo largo de todo el proceso de investigación y desarrollo llevado a cabo en este TFM.

A continuación, se detallan las principales fases y componentes metodológicos del proyecto:

- 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.

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.

3.4 Fundamentos matemáticos del modelo Trafficformer

Para aportar una visión rigurosa y formal del modelo propuesto, a continuación se detalla la formulación matemática de Trafficformer, describiendo sus componentes principales y la notación empleada a lo largo de este trabajo.

Componentes principales

1. Traffic Input.

El punto de partida del modelo es una matriz histórica de aforos:

$$S_t^I \in \mathbb{R}^{N \times I},$$

donde N representa la longitud de la ventana temporal (número de pasos de tiempo considerados) e I es el número de nodos o sensores de la red. Para cada nodo i , se recoge una secuencia temporal de sus últimas N mediciones de aforos, lo que puede visualizarse como un tensor tiempo \times nodo.

El modelo también emplea una máscara espacial, construida a partir de la distancia física y las relaciones topológicas entre nodos:

$$M^P \in \mathbb{R}^{I \times I}.$$

Se trata de una matriz binaria que determina qué pares de nodos pueden influirse mutuamente, filtrando las interacciones irrelevantes (por ejemplo, nodos distantes o no conectados directamente en la red viaria).

2. Temporal Feature Extractor

Cada columna de S_t^I , correspondiente a la serie temporal de un nodo, se procesa mediante un *Multi-Layer Perceptron* (MLP) compartido. El resultado es una matriz de embeddings temporales por nodo:

$$S_t^{C1} \in \mathbb{R}^{I \times d_t}.$$

3. Traffic Node Feature Interaction

Las representaciones temporales se combinan a través de un mecanismo de *Multi-head Self-Attention*, donde la máscara M^P restringe la atención a los nodos vialmente conectados. La estructura sigue el esquema Transformer estándar: atención → ADD & NORM → Feed-Forward → ADD & NORM. El resultado es un embedding espaciotemporal Enriquecido:

$$Z_t \in \mathbb{R}^{I \times d_s}.$$

4. Traffic Capacity Forecasting

Finalmente, un segundo MLP por nodo transforma Z_t en las predicciones para los siguientes H pasos temporales:

$$\hat{V}_{t+1:t+H} \in \mathbb{R}^{I \times H},$$

donde H es el horizonte de predicción.

Resumen y puntos clave

La Figura 2 muestra el flujo de información entre los diferentes módulos del modelo. A continuación, se sintetiza en una sola expresión el pipeline completo:

$$S_t^I \xrightarrow[\text{MLP}]{\text{Temporal}} S_t^{C1} \xrightarrow[\text{\textit{M}^P}]{\text{Self-Attention}} Z_t \xrightarrow[\text{MLP}]{\text{Regresión}} \hat{V}_{t+1:t+H}$$

Para mayor claridad, en la Tabla 2 se resume la notación empleada en esta sección.

Tabla 2: Notación de los principales símbolos utilizados en la formulación de Trafficformer.

Símbolo	Descripción
N	Longitud de la ventana histórica (nº de pasos de tiempo)
I	Número total de nodos/sensores en la red
S_t^I	Matriz de características históricas de tamaño $N \times I$
M^P	Máscara espacial ($I \times I$) que pondera la atención entre nodos
S_t^{C1}	Embedding temporal por nodo ($I \times d_t$)
Z_t	Embedding espaciotemporal final ($I \times d_s$)
$\hat{V}_{t+1:t+H}$	Aforos predichos para los próximos H pasos

Fuente: Elaboración propia.

Por último, se destacan a continuación varios aspectos clave del diseño de Trafficformer que justifican su elección y eficacia para la predicción de tráfico:

- El modelo *desacopla* el aprendizaje temporal del aprendizaje espacial.
- La máscara M^P introduce conocimiento a priori de la topología vial, lo que acelera la convergencia y mejora la interpretabilidad.
- Los MLP son ligeros y comparten pesos entre nodos, reduciendo la complejidad frente a arquitecturas basadas en LSTM o CNN.

Esta formulación matemática resume de forma rigurosa el funcionamiento interno de Trafficformer y proporciona la base conceptual sobre la que se desarrolla y adapta el modelo en el presente trabajo.

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.

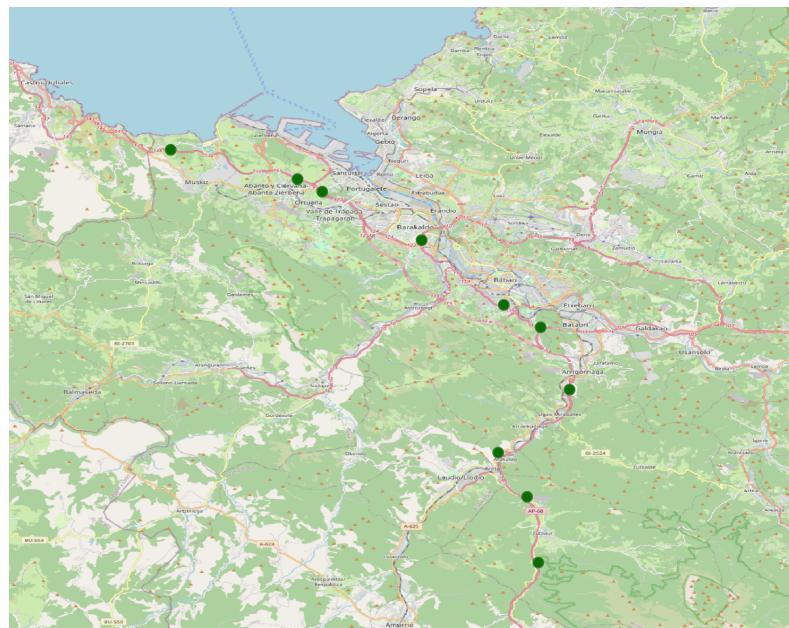
Sourceld	Organización	Meters	Flows	Incidences
1	Gobierno Vasco	✓	✓	✓
2	Diputación Foral de Bizkaia	✋	✋	✓
3	Diputación Foral de Álava	✗	✗	✓
4	Diputación Foral de Gipuzkoa	✓	✗	✓
5	Ayuntamiento Bilbao	✓	✓	✓
6	Ayuntamiento Vitoria-Gasteiz	✓	✓	✓
7	Ayuntamiento de Donostia-San Sebastián	✓	✗	✓

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

El presente caso de uso se circumscribe a la provincia de Bizkaia, por lo que únicamente se emplean las fuentes de datos identificadas con los códigos sourceld 1, 2 y 5. La fuente proporcionada por la Dirección de Tráfico del Gobierno Vasco abarca principalmente la red viaria principal de la Comunidad Autónoma del País Vasco (CAPV), incluyendo carreteras de alta capacidad. Por su parte, la base de datos de la Diputación Foral de Bizkaia recoge aforos situados, en su mayoría, en carreteras secundarias y forales. Finalmente, la fuente de datos del Ayuntamiento de Bilbao representa el entorno urbano, al contar con sensores ubicados en intersecciones y vías principales de la ciudad.

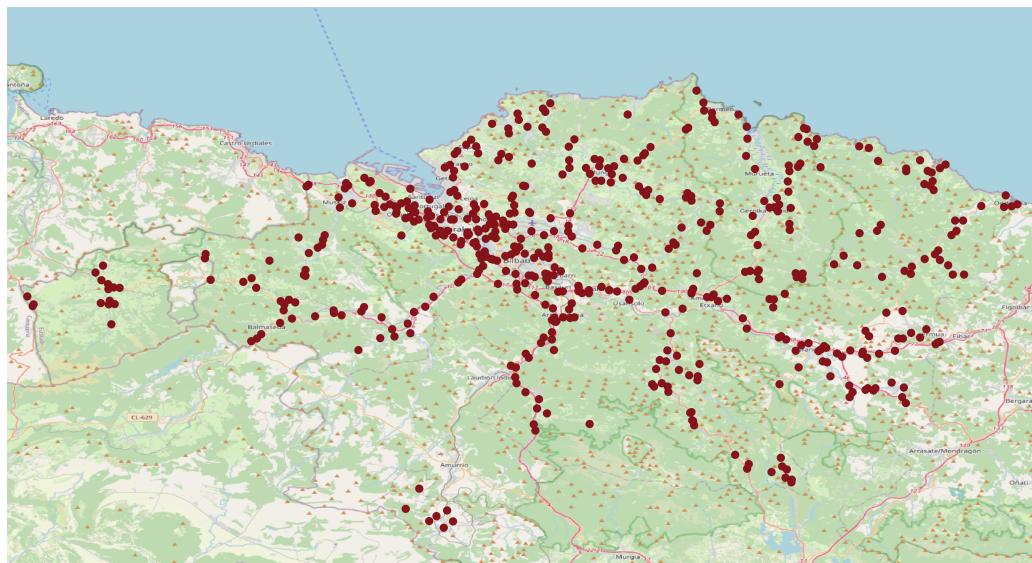
La Figura 4 muestra la ubicación de los sensores de la fuente source_id = 1 (Gobierno Vasco). En muchos de estos puntos existen múltiples medidores, ya que se instala un sensor por carril en el mismo emplazamiento físico.

Figura 4: Ubicación de aforos de la fuente Gobierno Vasco (source_id = 1)



La Figura 5 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.

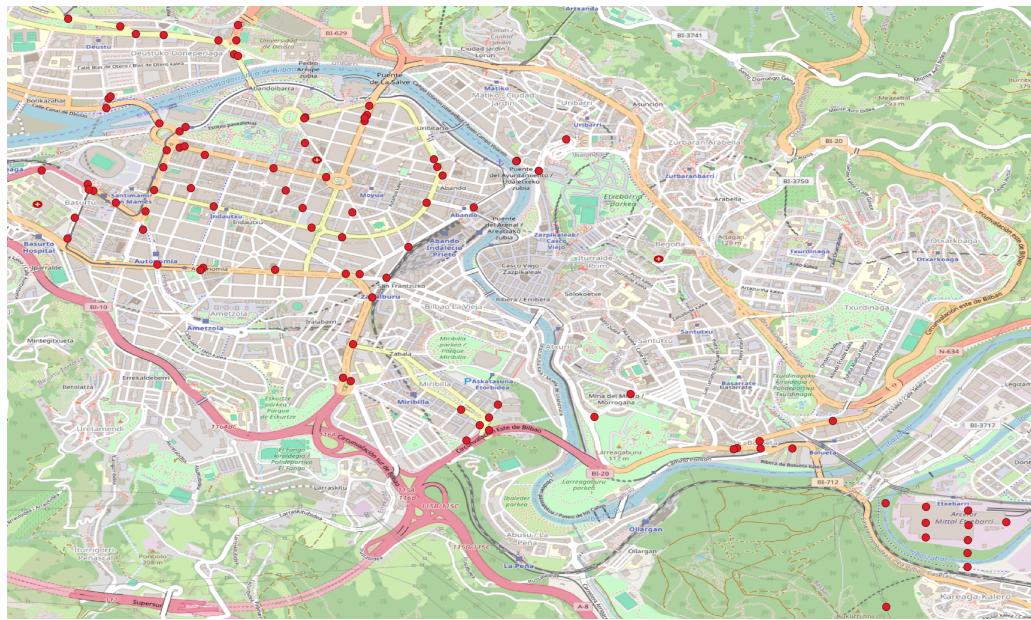
Figura 5: Ubicación de aforos de la fuente Diputación Foral de Bizkaia (source_id = 2)



Finalmente, la Figura 6 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 6: Ubicación de aforos de la fuente Ayuntamiento de Bilbao (*source_id* = 5)



Las incidencias se obtuvieron completamente para todos los sourceld.

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 7.

Figura 7: Ejemplo de error a la hora de consultar el API de Euskalmet.

Key	Value	Description
stationId	CODE	(Required) The id of the station
sensorId	G4C1	(Required) The id of the sensor
measureTypeId	measuresForWater	(Required) The id of the measure type
measureId	precipitation	(Required) The id of the measure
YYYY	2024	(Required) The four digits of a year
MM	1	(Required) The two digits of a month
DD	2	(Required) The two digits of a day
HH	19	(Required) The digits representing the hour of the day

```

1 | {
2 |   "code": 404,
3 |   "message": "[ group=100 code=2 code=2 severity=RECOVERABLE ]: Persistence error when executing 'LOAD' method: ENTITY_NOT_FOUND group=100 / code=2 severity=RECOVERABLE"
4 |

```

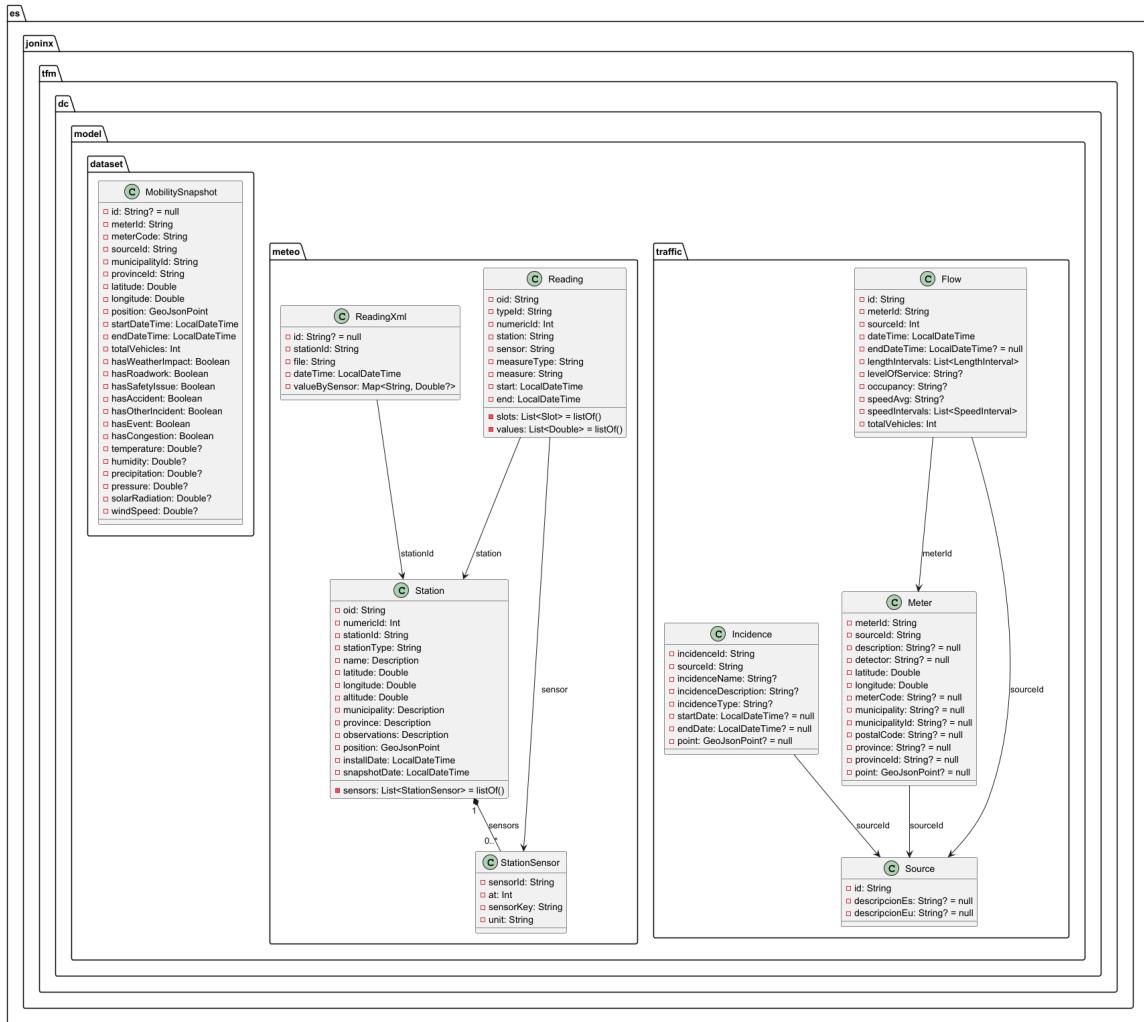
Ante la incertidumbre sobre la calidad y disponibilidad de los datos inicialmente previstos, se exploraron alternativas para la obtención de información relevante. En este proceso, se identificó dentro del propio portal de Open Data del Gobierno Vasco un recurso adicional: las lecturas

brutas recogidas por las estaciones meteorológicas correspondientes al año 2024, disponibles en formato XML y accesibles públicamente 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 8 se puede ver las clases persistidas en la base de datos.

Figura 8: 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.

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.

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.

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.

Source

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

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.

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.

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.

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).

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, Repository, Service Layer, Data Transfer Object, Helper/Utility, Facade y Factory/Singleton.

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 4 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 4: *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 criterios 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 5.

Tabla 5: 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	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: Variablesopcionales

```
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 B, 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 6.

Tabla 6: 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:

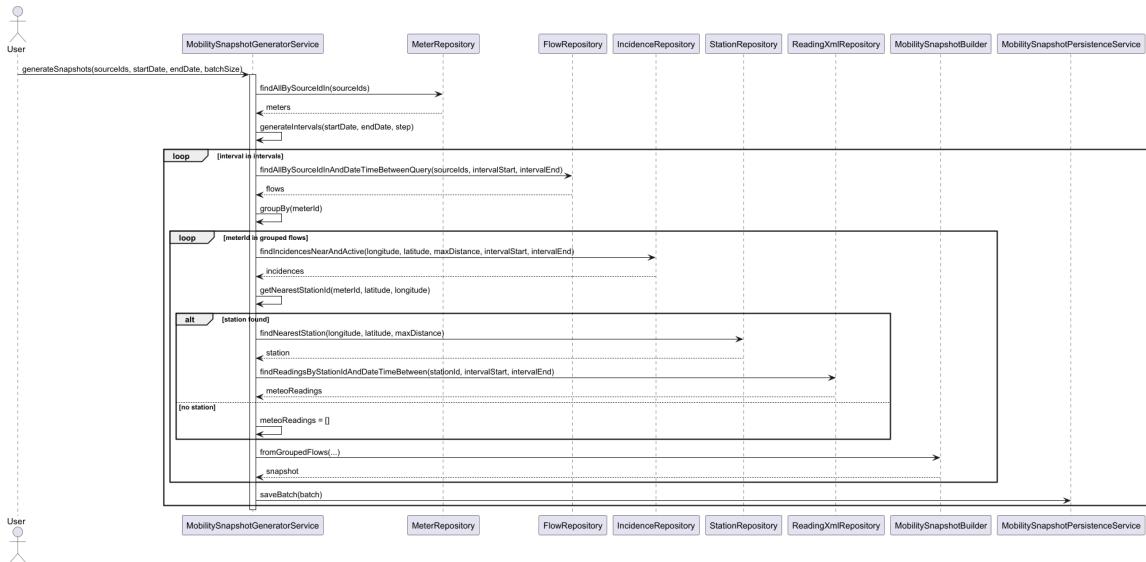
- **Flujos de tráfico (Flow)**: variables como meterId, dateTime, totalVehicles, y metadatos geográficos.
- **Meteorología (Reading)**: valores de los sensores seleccionados asociados espacial y temporalmente al flujo.
- **Incidencias (Incidence)**: información de incidencias activas cercanas en tiempo 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 9, mediante un diagrama de secuencia UML, que ilustra claramente el intercambio entre repositorios, lógica de negocio y servicios de persistencia.

Figura 9: 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 C, 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 con-

figurable (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 7: 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 micro-análisis	Equilibrio general	Análisis macro o planificación

En la Tabla 7 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.

4.4 Arquitectura y pipeline general del proyecto

A continuación, se presenta una descripción completa de la arquitectura y el pipeline diseñado para este proyecto, abarcando desde la recolección inicial de datos hasta la generación y validación de modelos predictivos basados en aprendizaje profundo.

Todo el proceso comienza con el artefacto Data Collector, un proyecto desarrollado en Kotlin utilizando el framework Spring Boot. Su función principal consiste en la construcción integral del dataset utilizado posteriormente para el entrenamiento de modelos predictivos.

En primer lugar, se ha recopilado la información meteorológica. Los datos estáticos, tales como la ubicación de las estaciones meteorológicas, se obtienen directamente mediante la API de Euskalmet proporcionada por el Gobierno Vasco. Debido a problemas técnicos en la extracción automática de las mediciones diarias, estas han sido extraídas mediante la descarga manual de un archivo ZIP del portal Open Data del Gobierno Vasco, que contiene todas las lecturas meteorológicas del año 2024.

En segundo lugar, el dataset incorpora información de aforos vehiculares proveniente de distintas fuentes, centradas exclusivamente en Bizkaia: una del Gobierno Vasco, dos de la Diputación Foral de Bizkaia y cinco del Ayuntamiento de Bilbao. La información relativa a las fuentes del Gobierno Vasco y del Ayuntamiento de Bilbao se ha extraído mediante su API de Tráfico, incluyendo tanto estaciones como mediciones e incidencias del tráfico. Por otro lado, los datos proporcionados por la Diputación Foral de Bizkaia han sido facilitados directamente mediante un acuerdo expreso con dicha institución.

Una vez obtenida y consolidada toda esta información, se almacena en una base de datos MongoDB, y posteriormente se procede a construir el dataset definitivo en la colección denominada MobilitySnapshot. Este proceso se describe detalladamente en secciones previas del presente capítulo.

Tras esta fase de recolección y consolidación, se entra en la etapa del proyecto principal deno-

minado Trafficformer Bizkaia. Este proyecto está desarrollado en Python, utilizando librerías relevantes para el aprendizaje profundo como PyTorch. Se han implementado dos modelos predictivos principales: un modelo base MultiLayerPerceptron (MLP) y el modelo definitivo Trafficformer, basado en una arquitectura moderna de tipo Transformer.

Para gestionar los experimentos y entrenamientos, se han definido una serie de procesos específicos descritos en profundidad en el Capítulo 5. Se han diseñado diferentes entrenamientos para evaluar ambas arquitecturas (MLP y Trafficformer) y para realizar múltiples pruebas variando hiperparámetros significativos.

Estos entrenamientos se ejecutan principalmente en notebooks Jupyter sobre una máquina local equipada con una GPU de alto rendimiento, suficiente para el entrenamiento de redes neuronales profundas. Adicionalmente, se dispone de una infraestructura preparada para realizar entrenamientos de forma escalable en la nube mediante instancias EC2 de Amazon Web Services (AWS) y utilizando imágenes AMI configuradas con soporte Nvidia.

Durante el proceso de entrenamiento y evaluación, se ha utilizado la plataforma Weight & Biases para realizar un seguimiento detallado de los experimentos, facilitando así la elección y validación de los mejores modelos entre los 120 experimentos realizados (un experimento por cada sourceId y tipo de modelo).

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.

Propósito e hipótesis 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 a una *MLP*.

Antes de abordar las particularidades técnicas de cada arquitectura en apartados posteriores,

en la Tabla 8 se presenta una comparación conceptual entre los dos modelos evaluados: un perceptrón multicapa clásico (MLP) y el modelo propuesto basado en atención (Trafficformer).

Tabla 8: 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 empleadas en el desarrollo del proyecto se encuentra detallado en el archivo `pyproject.toml`. A continuación, se enumeran las principales librerías utilizadas, acompañadas de una breve descripción de su función dentro del flujo de trabajo:

- **numpy y pandas:** Bibliotecas fundamentales para la manipulación y análisis eficiente de datos numéricos y estructurados en Python. Permiten operaciones vectorizadas, gestión de grandes volúmenes de datos y transformaciones avanzadas, esenciales en el preprocesamiento y análisis exploratorio.
- **scikit-learn:** Conjunto de herramientas de aprendizaje automático clásico, utilizado para la normalización de datos, particionado de conjuntos y evaluación mediante métricas estándar, así como para el desarrollo y comparación de modelos base.

- **torch, torchvision y torchaudio:** Paquete principal de PyTorch, framework de referencia para el desarrollo y entrenamiento de modelos de deep learning. Permiten la implementación eficiente de arquitecturas neuronales avanzadas, como Transformers, así como la integración con módulos especializados para visión y audio.
- **wandb:** Plataforma para el seguimiento, registro y visualización de experimentos de aprendizaje automático, facilitando la comparación reproducible de resultados y el ajuste de hiperparámetros.
- **matplotlib y seaborn:** Herramientas de visualización de datos que permiten generar gráficos y representaciones visuales de resultados, métricas de entrenamiento y análisis exploratorio.
- **pymongo:** Cliente de Python para la base de datos MongoDB, utilizado para la gestión y consulta eficiente de grandes volúmenes de datos históricos de tráfico y meteorología.
- **geopy y folium:** Librerías para el procesamiento y visualización de datos geoespaciales. Permiten la geolocalización, el cálculo de distancias y la creación de mapas interactivos con información relevante sobre la red viaria.
- **onnx, onnxscript y onnxruntime:** Conjunto de herramientas para la exportación y ejecución de modelos en formato ONNX, facilitando la interoperabilidad y el despliegue de modelos en entornos heterogéneos o en producción.
- **osmnx:** Biblioteca para la descarga y análisis de datos de redes viarias desde OpenStreetMap, utilizada para la construcción de grafos topológicos que representan la infraestructura de carreteras de Bizkaia.
- **tabulate:** Herramienta auxiliar para la generación de tablas en formato texto, útil para la presentación estructurada de resultados y métricas.
- **python-dotenv:** Permite la gestión de variables de entorno y configuración sensible, facilitando la portabilidad y seguridad del entorno de experimentación.

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).

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.

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.

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 D.

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 que representa la red viaria de sensores. Cada nodo corresponde a un `meter`, mientras que las aristas se definen en función de su proximidad física o conectividad vial. Para su construcción, se parte de las posiciones geográficas de los sensores, agrupándolos mediante heurísticas espaciales y considerando su distribución sobre el territorio. Esta lógica se implementa en el módulo `sensor_network_map.py`, apoyándose en información abierta de OpenStreetMap y cálculos de distancia geodésica.

A modo ilustrativo, en las Figuras 10, 11 y 12 se muestran las distribuciones espaciales de sensores para las distintas entidades proveedoras de datos: Gobierno Vasco (sourceld 1), Diputación Foral de Bizkaia (sourceld 2) y Ayuntamiento de Bilbao (sourceld 5), respectivamente. Estos gráficos permiten apreciar la heterogeneidad geográfica en la cobertura sensorial y su concentración en zonas urbanas o periurbanas.

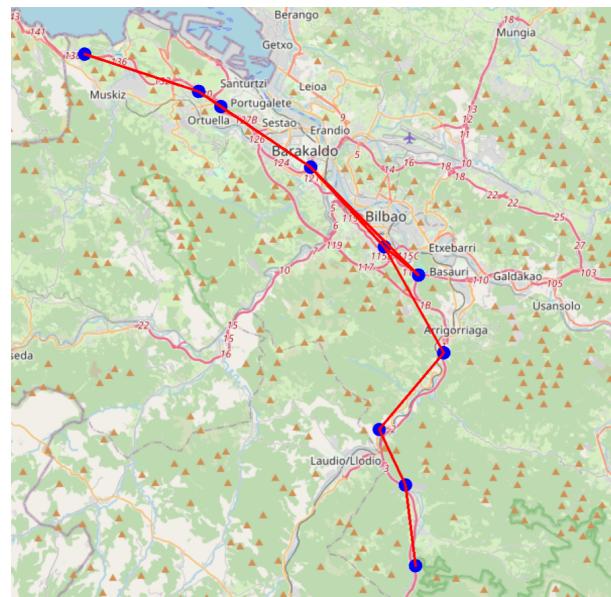
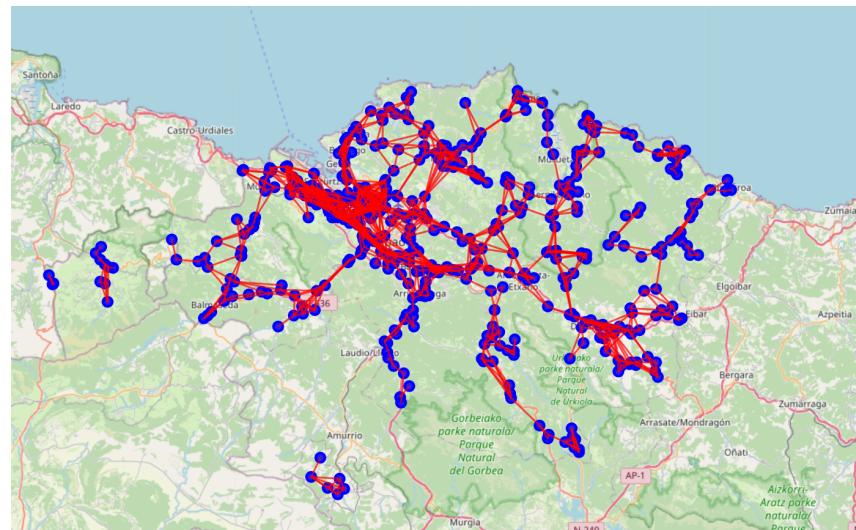
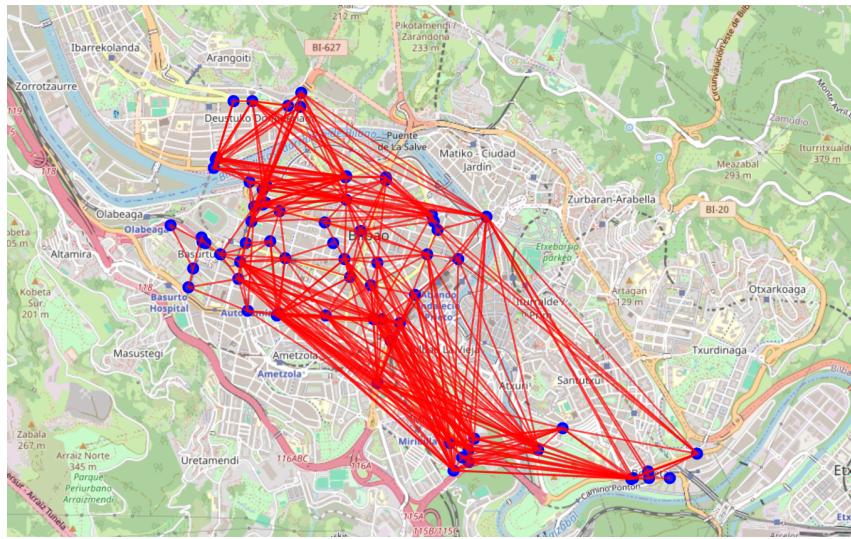
Figura 10: Distribución espacial de sensores para Gobierno Vasco (sourceId 1).**Figura 11:** Distribución espacial de sensores para Diputación Foral de Bizkaia (sourceId 2).

Figura 12: Distribución espacial de sensores para Ayuntamiento de Bilbao (sourceId 5).



A partir del grafo de sensores se puede derivar la matriz de adyacencia, que resulta clave para la aplicación de técnicas de enmascaramiento espacial y la explotación de correlaciones topológicas en los modelos de predicción.

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 dos estrategias principales:

- `BinarySpatialMaskStrategy`: enmascaramiento binario según distancia máxima.
- `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.

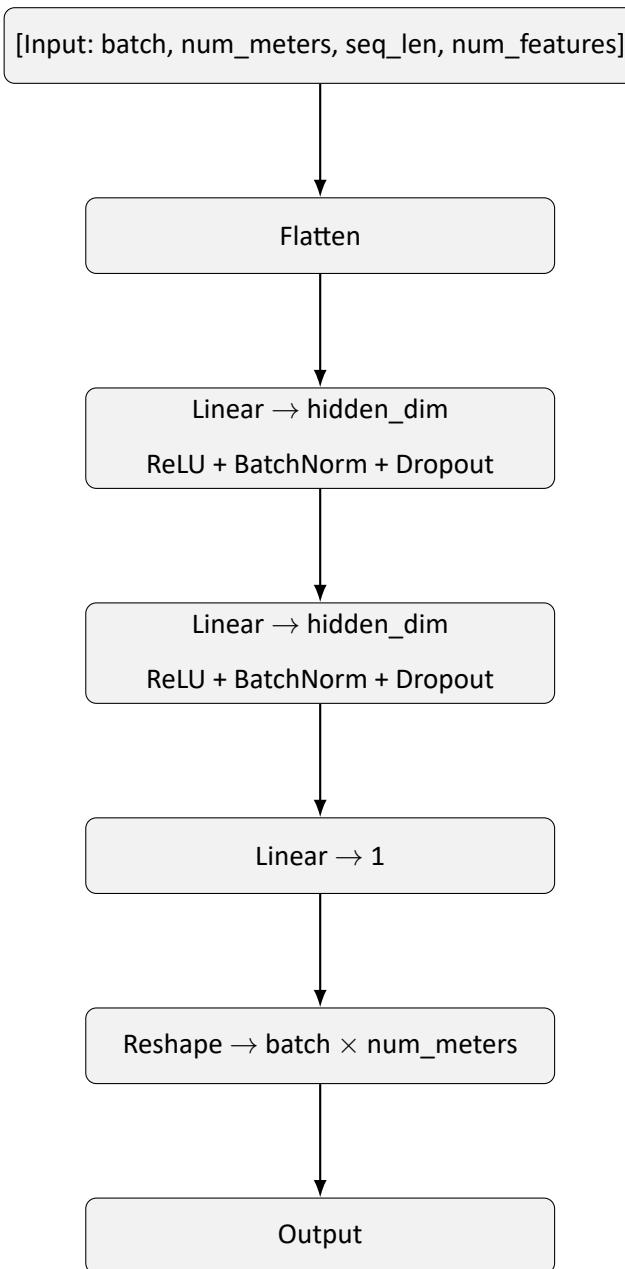
Perceptrón Multicapa (MLP)

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 13 se presenta un esquema lógico de su flujo de capas.

Figura 13: Esquema lógico del flujo de capas en el modelo MLP.

Fuente: Elaboración propia.

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 E, 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.

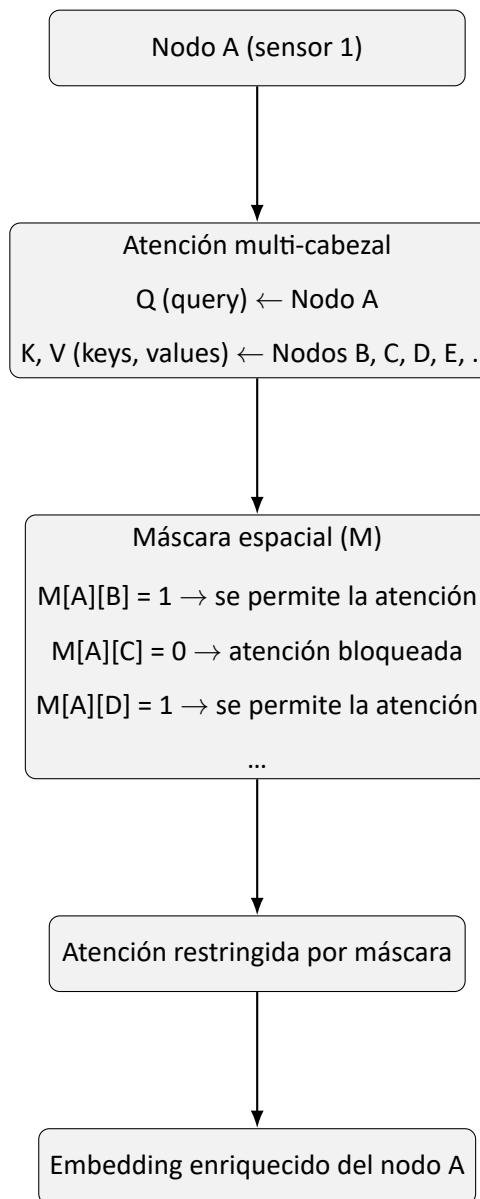
Seguidamente, cada nodo procesa localmente su secuencia temporal a través de un extractor de características basado en un perceptrón multicapa (MLP), dotado de capas Linear, activaciones ReLU, normalización LayerNorm y Dropout. Esto da lugar a un embedding por nodo que encapsula su evolución temporal 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-cabezal (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 14, 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 14: Aplicación de la máscara espacial en el mecanismo de atención de *Trafficformer*.

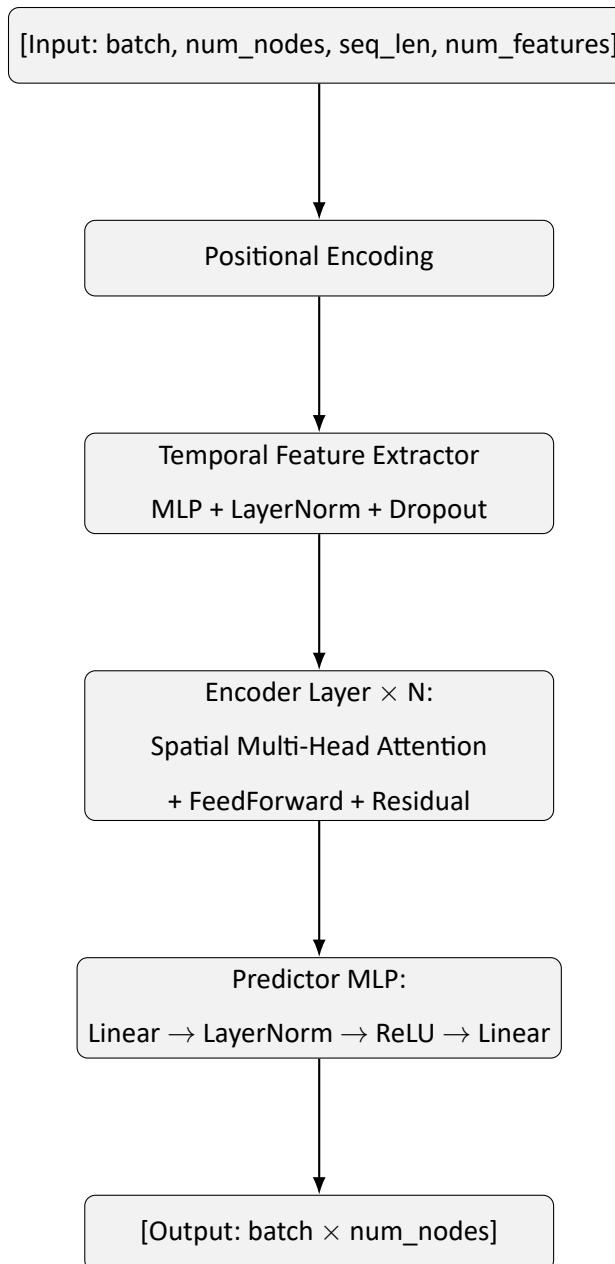


Fuente: Elaboración propia.

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 15: Esquema lógico del flujo de capas del modelo Trafficformer.



Fuente: Elaboración propia.

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. En concreto, el total de combinaciones generadas asciende a **120 experimentos**, resultado de combinar las 3 fuentes de datos preparadas (Gobierno Vasco, Diputación Foral de Bizkaia y Ayuntamiento de Bilbao), 2 arquitecturas principales (MLP y Trafficformer), y 20 configuraciones distintas de hiperparámetros. Cada uno de estos experimentos 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 9 se presentan los valores explorados para cada parámetro, clasificados por arquitectura.

Tabla 9: *Configuraciones experimentales evaluadas*

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

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:

- **Sourceld (SID):** La fuente de datos.
- **Modelo (M):** El modelo.
- **SEQ_LEN (SL):** 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 (LR):** 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 (BS):** 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, Patience (E/ESP):** 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 (NH) y EMBEDDING_DIM (ED):** 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 (NL) y FF_HIDDEN_DIM (FF):** 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 F 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.

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 autenti-

cación 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).

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 %).

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.

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.

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.

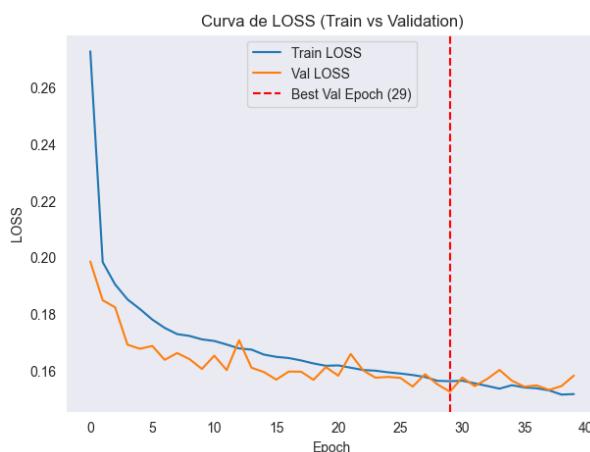
En cada iteración se guardan:

- 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.

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 se corresponde con la mínima pérdida de validación y se resalta en cada gráfica. Estas curvas permiten detectar patrones de sobreentrenamiento, estancamiento o mejoras progresivas en el aprendizaje.

Figura 16: Curvas de evolución de pérdida para entrenamiento y validación, con indicación de la mejor época.



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 G 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 10 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 10: *Resultados de los mejores modelos por combinación `sourceId`-arquitectura*

SID	M	EP	TL	TMAE	TRMSE	TMAPE	TR2
1	mlp	67	0.000	25.466	38.489	1.853e0	0.759
1	trafficformer	30	0.143	19.125	30.784	1.745e0	0.822
2	mlp	30	0.000	31.017	38.177	1.82e14	0.866
2	trafficformer	96	0.002	5.832	15.046	1.51e3	0.982
5	mlp	39	0.000	213.850	343.603	4.83e5	-1.797
5	trafficformer	29	0.153	175.975	306.134	1.74e5	0.640

SID: `sourceId`.

M: Modelo.

EP: Epoch óptima.

TL: Test Loss.

TMAE: Test MAE.

TRMSE: Test RMSE.

TMAPE: Test MAPE.

TR2: 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 17: Curvas de entrenamiento para el modelo MLP con datos del Gobierno Vasco (sourceId 1).

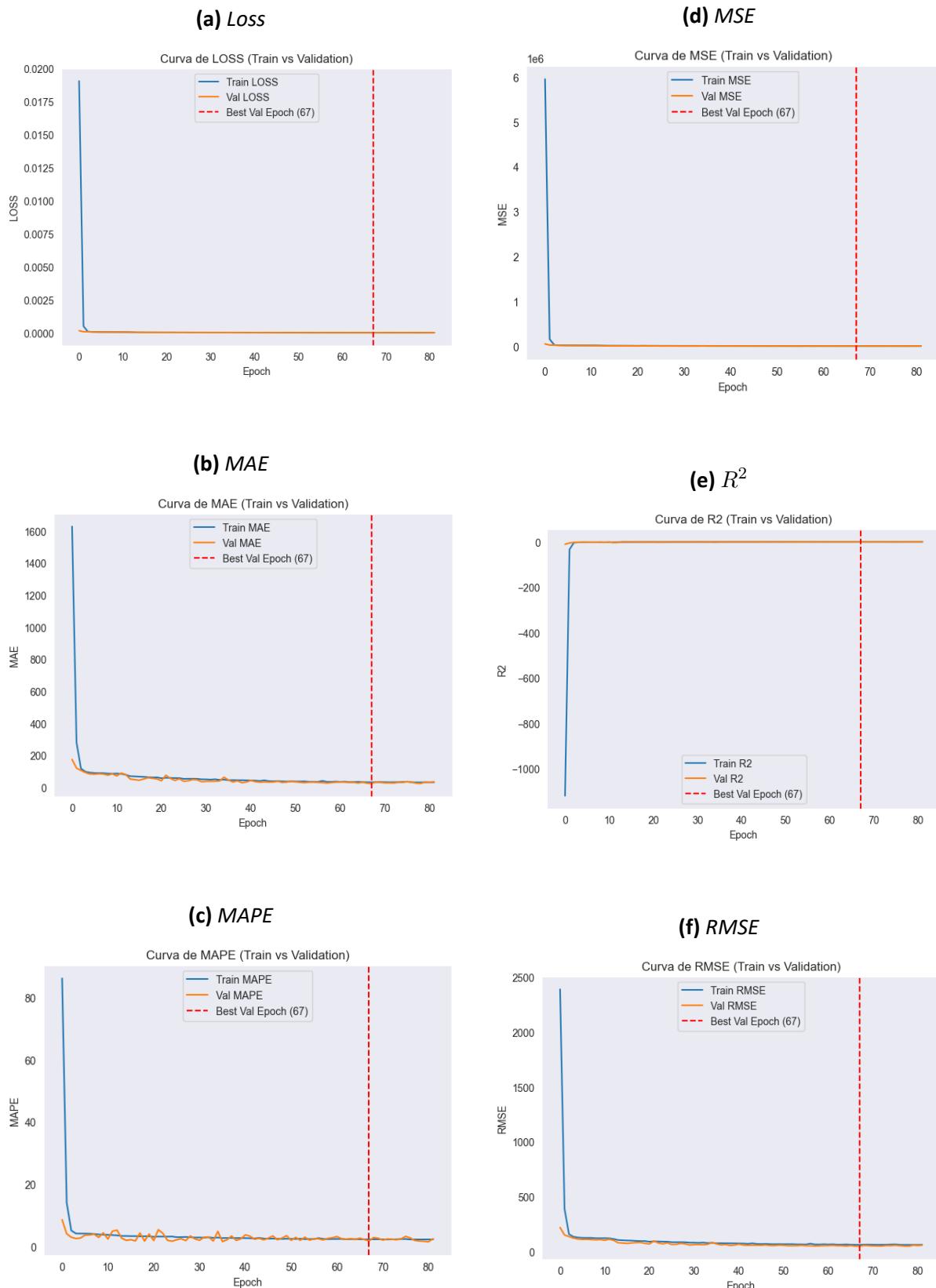


Figura 18: Curvas de entrenamiento para el modelo *Trafficformer* con datos del Gobierno Vasco (sourceId 1).

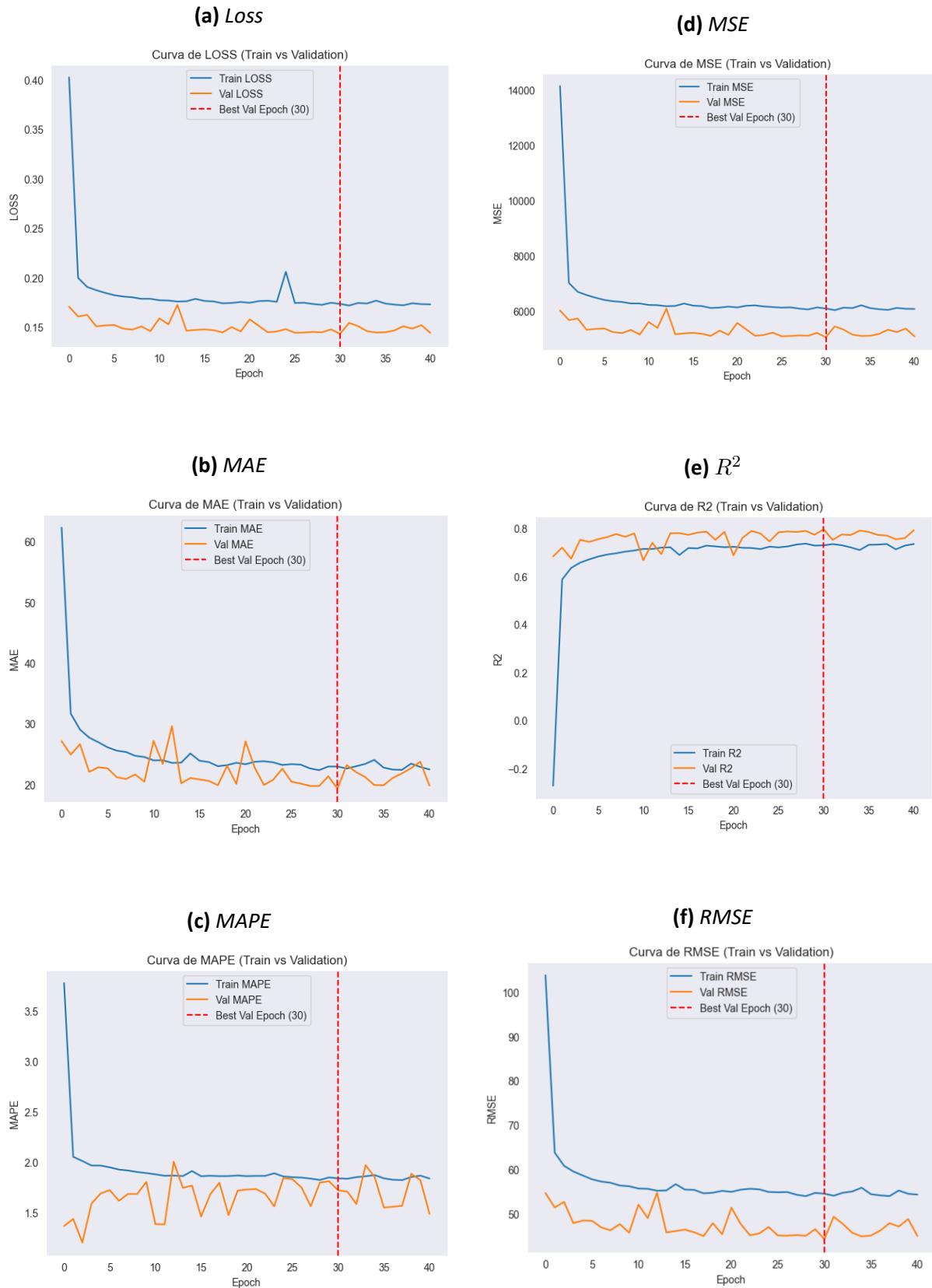


Figura 19: Curvas de entrenamiento para el modelo MLP con datos de la Diputación Foral de Bizkaia (sourceId 2).

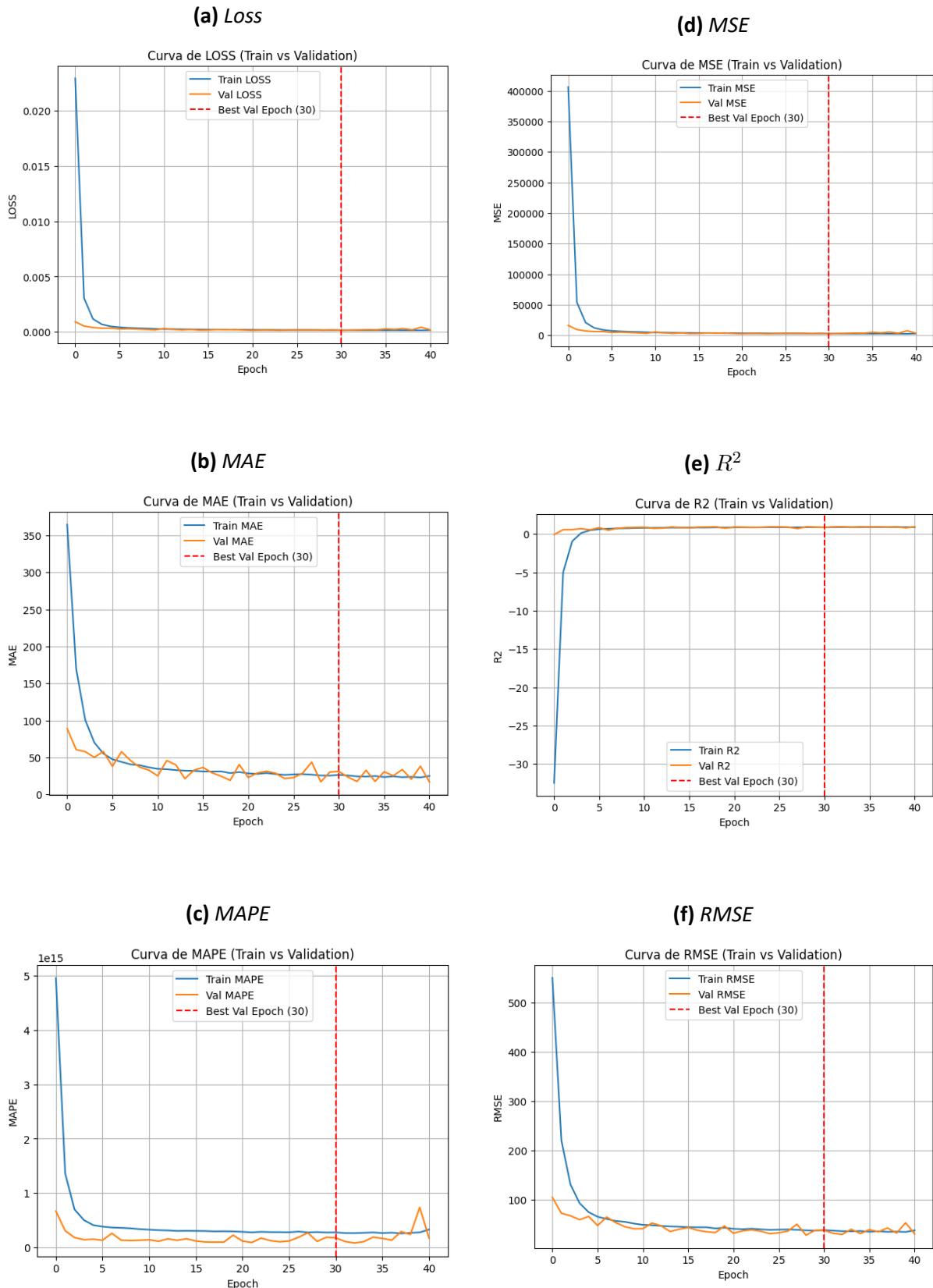


Figura 20: Curvas de entrenamiento para el modelo *Trafficformer* con datos de la Diputación Foral de Bizkaia (sourceId 2).

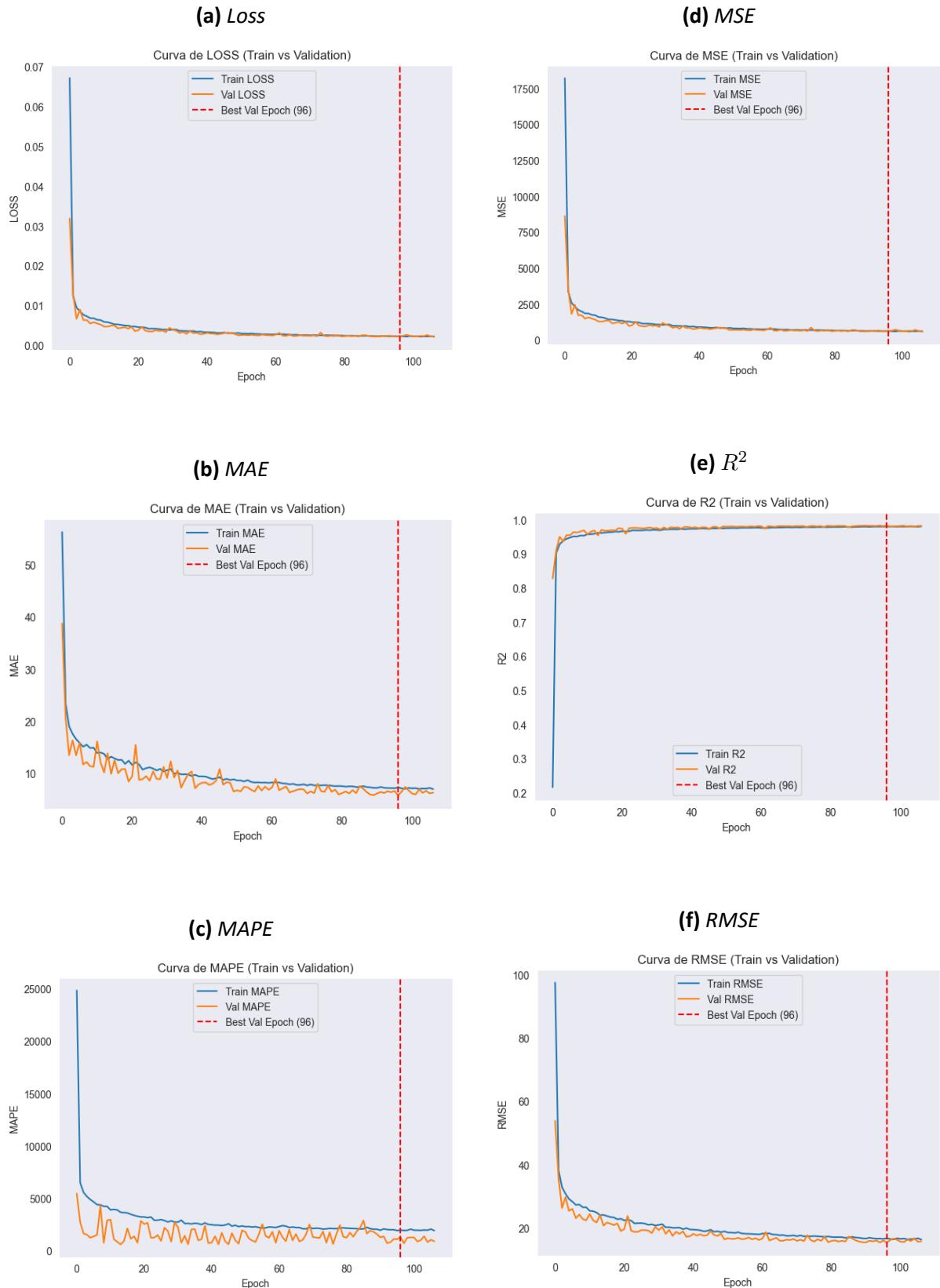


Figura 21: Curvas de entrenamiento para el modelo MLP con datos del Ayuntamiento de Bilbao (sourceId 5).

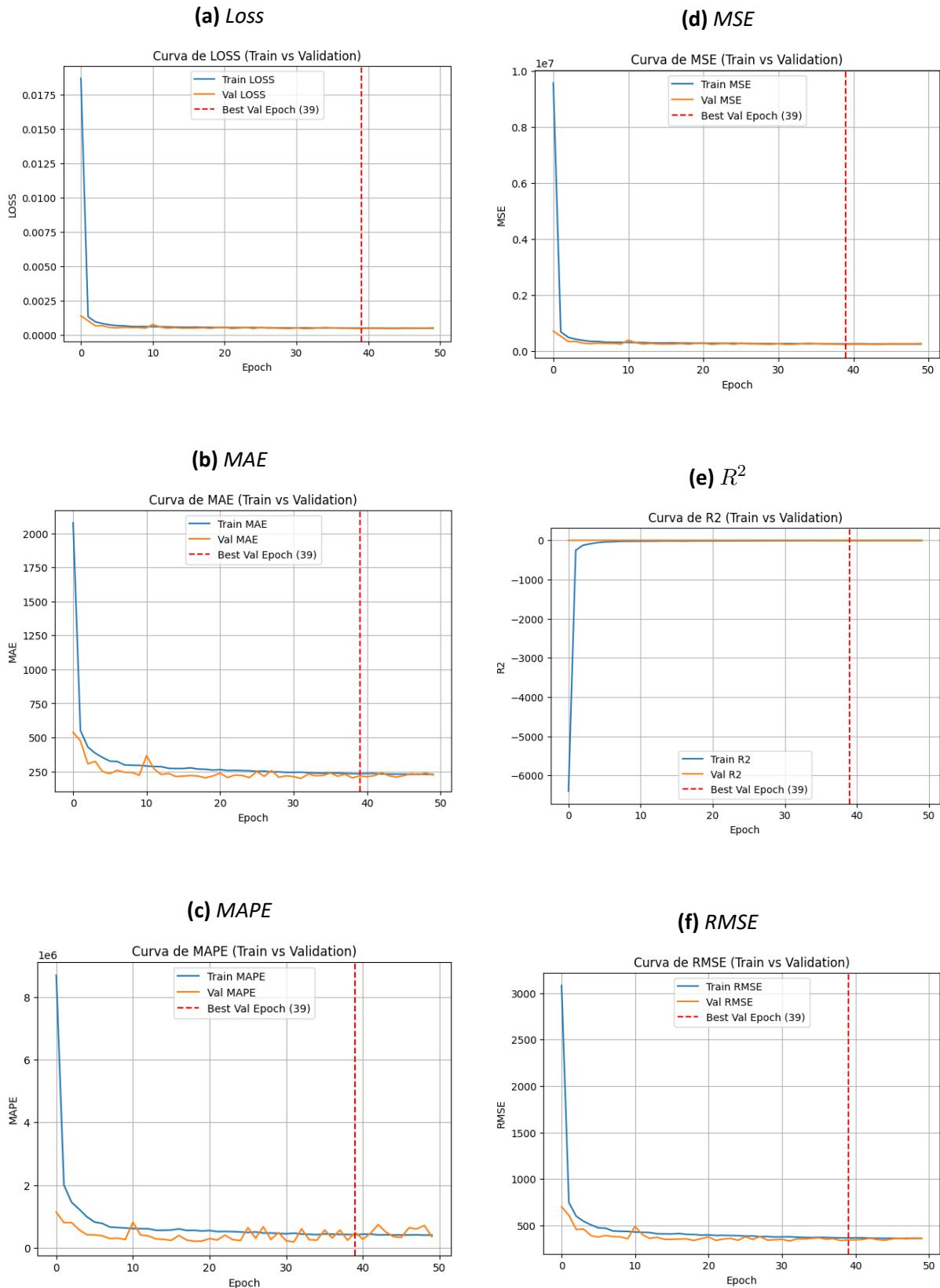
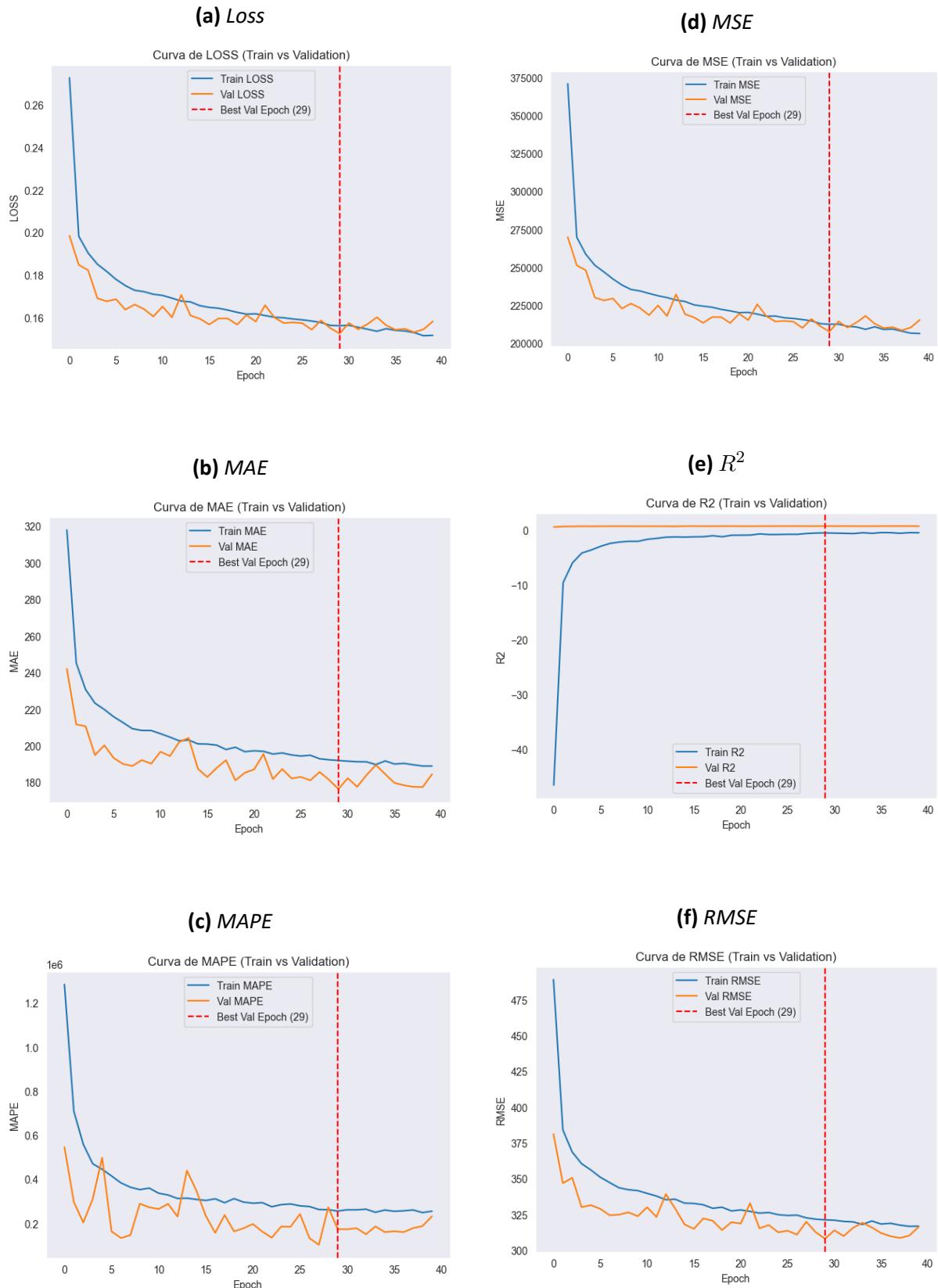


Figura 22: Curvas de entrenamiento para el modelo *Trafficformer* con datos del Ayuntamiento de Bilbao (sourceId 5).



Antes de analizar en profundidad los resultados, conviene contextualizar los experimentos rea-

lizados respecto al modelo original Trafficformer propuesto por Chang et al. (2025). En dicho estudio, la evaluación se realizó sobre el *Seattle Loop Detector Dataset*, un conjunto de datos altamente estructurado y representativo de una red viaria en anillo con características propias, tanto en topología como en calidad de señal.

Por el contrario, en el presente trabajo se han utilizado datos abiertos de diferentes fuentes públicas, correspondientes a la provincia de Bizkaia y el entorno urbano de Bilbao, incluyendo sensores instalados en redes viarias de distinta naturaleza (autonómica, foral y municipal) y sometidos a procesos adicionales de integración, limpieza y preprocesamiento. Esta heterogeneidad y variabilidad inherente a los datos introduce desafíos adicionales en el modelado y la predicción.

A pesar de estas diferencias, los resultados obtenidos son comparables en términos de magnitud y tendencia a los reportados por Chang et al., lo que evidencia la solidez y capacidad de generalización de la arquitectura Trafficformer, incluso en contextos con fuentes de datos abiertas, distribuidas y heterogéneas. Esta comparativa refuerza la validez de las conclusiones presentadas a continuación y pone en valor la adaptabilidad del modelo a diferentes escenarios reales de predicción de tráfico.

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 10, junto con las gráficas de entrenamiento para cada modelo, para ofrecer un análisis detallado sobre la superioridad y limitaciones observadas.

Comparativa en datos del Gobierno Vasco (sourceld 1)

El conjunto de datos del Gobierno Vasco incluye 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 10 (véase capítulo anterior). Cada sensor

representa un carril individual, lo que genera una alta densidad espacial en áreas críticas, pero una baja dispersión geográfica global. El dataset empleado para el entrenamiento tiene una dimensionalidad de entrada de 14 828 ventanas temporales, con una longitud de secuencia (`seq_len`) de 8 (equivalente a ventanas de 4 horas). El conjunto de entrenamiento consta de 10 379 muestras, el de validación de 2 980 y el de test de 1 469. Esta configuración temporal amplia permite a los modelos captar patrones dinámicos prolongados, aspecto especialmente favorable para el mecanismo de atención espacial de Trafficformer.

La comparativa entre modelos MLP (figura 17) y Trafficformer (figura 18) muestra claramente la superioridad de la arquitectura basada en Transformer. Trafficformer obtiene valores significativamente mejores en las métricas de pérdida (`loss`), MAE, RMSE y MAPE, así como un mayor coeficiente de determinación (R^2). En particular, Trafficformer reduce el MAE de 25.466 a 19.125 y el RMSE de 38.489 a 30.784, lo que refleja una mayor capacidad para capturar las complejidades espaciales y temporales de los datos de tráfico en este entorno.

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

La disposición lineal y agrupada de los sensores en puntos estratégicos de la red viaria (véase Figura 10) facilita que el modelo Trafficformer aproveche las relaciones espaciales locales, incrementando así su desempeño frente a modelos menos estructurados como el MLP.

Comparativa en datos Diputación Foral de Bizkaia (sourceld 2)

La fuente de datos de la Diputación Foral de Bizkaia presenta una distribución geográfica amplia y diversa, cubriendo exhaustivamente la red viaria de Bizkaia, como se aprecia en la Figura 11 (véase capítulo anterior). Inicialmente se consideraron 520 sensores, aunque tras el filtrado por ausencia de datos el número efectivo de sensores fue de 462. Esta elevada cantidad y dispersión espacial implica una alta complejidad en términos de relaciones y 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 (`seq_len` = 4, equivalentes a ventanas de 2 horas),

adaptada a la mayor densidad y complejidad espacial de la red. La máscara espacial generada revela numerosas conexiones entre sensores distantes, lo que beneficia de manera significativa al modelo Trafficformer gracias a su capacidad de capturar relaciones espaciales complejas mediante atención multi-cabeza.

La superioridad del modelo Trafficformer respecto al modelo MLP se acentúa aún más en los datos de la Diputación Foral de Bizkaia. Como se observa en las figuras 19 y 20, 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) a 0.982 (Trafficformer), lo que indica 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.

La amplia y densa distribución geográfica de los sensores para la Diputación Foral de Bizkaia (véase Figura 11) 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 datos Ayuntamiento de Bilbao (sourceld 5)

El tercer conjunto de datos se caracteriza por una cobertura urbana densa, centrada en la ciudad de Bilbao, como se aprecia en la Figura 12 (véase capítulo anterior). Aunque inicialmente se disponía de 92 sensores, el filtrado por calidad y disponibilidad redujo el número efectivo a 70. Esta elevada concentración espacial en un entorno urbano genera relaciones altamente correlacionadas y complejas entre sensores, lo que presenta retos específicos para los modelos de predicción de tráfico. El dataset consta de 17 330 ventanas temporales, de las cuales 12 131 se utilizaron para entrenamiento, 3 483 para validación y 1 716 para test. Al igual que en el caso anterior, se utilizó una ventana temporal de 2 horas (`seq_1en = 4`), permitiendo al modelo captar de manera eficiente las dinámicas urbanas rápidas propias de la ciudad.

Para la fuente sourceld 5, el modelo Trafficformer sigue mostrando una clara ventaja frente al modelo MLP. Según las figuras 21 y 22, se observa una mejora sustancial en todas las métricas

principales. El MAE disminuye de 213.850 a 175.975, y el RMSE de 343.603 a 306.134. El coeficiente de determinación R^2 experimenta una mejora especialmente relevante, pasando de un valor negativo e inadecuado de -1.797 (MLP) a un aceptable 0.640 (Trafficformer), lo que indica una mayor capacidad para explicar la variabilidad del tráfico urbano.

Las curvas de entrenamiento muestran que Trafficformer converge rápidamente (época 29), evidenciando que su mecanismo de atención espacial resulta especialmente efectivo en escenarios urbanos densos, donde la interacción entre sensores cercanos es particularmente compleja.

La configuración urbana y la alta densidad espacial de los sensores del Ayuntamiento de Bilbao (véase Figura 12) genera una compleja red de interacciones espaciales que favorece notablemente al modelo Trafficformer frente al MLP, que carece de mecanismos específicos para gestionar esta complejidad. Este análisis subraya la importancia de adaptar el diseño experimental y la selección de ventanas temporales a las particularidades de cada conjunto de datos, ya que estas decisiones se reflejan directamente en el rendimiento relativo de los modelos evaluados.

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 H.

Gobierno Vasco (`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.

Diputación Foral de Bizkaia (SourceId 2)

En el caso de la fuente de datos 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.

Ayuntamiento de Bilbao (SourceId 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.

Síntesis comparativa respecto al estado del arte

Con el objetivo de contextualizar los resultados obtenidos en este trabajo, se presenta una comparativa entre las métricas alcanzadas por el modelo Trafficformer sobre los distintos conjuntos de datos locales y las reportadas en el estudio original de Chang et al. (2025).

Tabla 11: Comparativa de resultados de Trafficformer respecto al estado del arte.

	Dataset	Variable Predicha	MAE	MAPE (%)	RMSE	R ²
Trafficformer (Chang et al., 2025)	Seattle Loop	Velocidad (mph)	2.10	4.70	3.08	–
Trafficformer (TFM, Sourceld 1)	Gobierno Vasco	Capacidad (veh/h)	19.125	1.745	30.784	0.822
Trafficformer (TFM, Sourceld 2)	DFB	Capacidad (veh/h)	5.832	1.51e3	15.046	0.982
Trafficformer (TFM, Sourceld 5)	Ayto. Bilbao	Capacidad (veh/h)	175.975	1.74e5	306.134	0.640

Tal y como se recoge en la Tabla 11, se aprecia una consistencia en la tendencia favorable del modelo Trafficformer frente a arquitecturas más simples como MLP, en línea con lo observado en la literatura reciente. Los resultados no son estrictamente comparables, ya que el modelo original evalúa la predicción de velocidades medias (en mph) sobre el dataset Seattle Loop, mientras que en este TFM se predice la capacidad de tráfico (vehículos/hora) en datasets locales distintos (Gobierno Vasco, DFB y Ayuntamiento de Bilbao).

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 (sourceld 5) y en redes de sensores extensas (sourceld 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.

7 Conclusiones y trabajo futuro

Este capítulo recopila y sintetiza los hallazgos más relevantes derivados del desarrollo y evaluación de los modelos propuestos en este trabajo fin de máster. A lo largo de las siguientes secciones se presentan, en primer lugar, las conclusiones generales del estudio, haciendo especial hincapié en el grado de consecución de los objetivos y la validación de la hipótesis planteada. Posteriormente, se resumen los aprendizajes principales extraídos en cada uno de los contextos evaluados, se abordan de forma crítica las limitaciones y consideraciones de validez de los experimentos realizados, y se proponen líneas de trabajo futuro. El capítulo concluye con una reflexión sobre la aportación realizada y el potencial impacto del sistema desarrollado en el ámbito de la predicción del tráfico.

7.1 Síntesis global, verificación de la hipótesis y aportación al campo

El objetivo principal de este trabajo fue diseñar, implementar y evaluar modelos avanzados de predicción del tráfico vehicular, centrándose en el desarrollo y validación de una arquitectura basada en Transformers adaptada a la complejidad de los datos reales de movilidad. La hipótesis de partida planteaba que, mediante el uso de mecanismos de atención espacial y temporal, sería posible superar la precisión y capacidad de generalización de modelos clásicos como las redes neuronales multi-capa (MLP), especialmente en entornos con alta densidad de sensores y complejidad topológica.

Los resultados obtenidos a lo largo del estudio han permitido confirmar dicha hipótesis. El modelo Trafficformer ha demostrado de forma consistente un mejor desempeño en comparación con la arquitectura MLP en los tres contextos evaluados (Gobierno Vasco, Diputación Foral de Bizkaia y Ayuntamiento de Bilbao), logrando mejoras significativas en métricas clave como MAE, MSE, RMSE y R^2 . Estas mejoras se han observado tanto en escenarios de tráfico lineal y disperso como en entornos urbanos densos, evidenciando la capacidad del modelo para capturar patrones espaciotemporales complejos y adaptarse a distintos tipos de redes viarias.

En cuanto al cumplimiento de los objetivos planteados, se han alcanzado tanto el objetivo general como los objetivos específicos definidos al inicio del trabajo:

- **Diseñar e implementar un pipeline completo de tratamiento, modelado y evaluación de datos de tráfico,** que ha permitido gestionar eficientemente múltiples fuentes de da-

tos y preparar los conjuntos de entrenamiento necesarios para la experimentación.

- **Desarrollar e integrar modelos basados en arquitecturas profundas avanzadas**, en particular el modelo Trafficformer, adaptando y optimizando sus hiperparámetros para distintos escenarios y tipos de datos.
- **Comparar de manera sistemática el rendimiento de distintas arquitecturas**, utilizando métricas rigurosas y plataformas de experimentación reproducible, como Weights & Biases.
- **Analizar la capacidad de generalización y robustez de los modelos propuestos** en contextos reales de la provincia de Bizkaia, destacando la adaptabilidad del sistema a diferentes configuraciones y tipos de red viaria.

Más allá de la validación experimental, este trabajo establece una base sólida para la aplicación de modelos avanzados de aprendizaje profundo —y en particular de arquitecturas basadas en Transformers— en el ámbito de la predicción del tráfico vehicular en entornos reales y heterogéneos. La principal aportación reside en haber demostrado que los mecanismos de atención espaciotemporal implementados en Trafficformer permiten superar las limitaciones de modelos tradicionales, logrando una mayor capacidad predictiva y una mejor adaptación a escenarios complejos y dinámicos.

Adicionalmente, el desarrollo de un pipeline reproducible y escalable, junto con el uso de plataformas modernas de experimentación, contribuye a mejorar la transparencia y la replicabilidad en futuras investigaciones. Se espera que las conclusiones extraídas y las líneas de trabajo propuestas sirvan de referencia para la comunidad investigadora y para profesionales del sector de la movilidad, facilitando el despliegue de sistemas inteligentes de predicción de tráfico en contextos urbanos y metropolitanos cada vez más exigentes.

En definitiva, este trabajo refuerza el papel de las arquitecturas Transformer como herramientas clave para abordar los desafíos emergentes en la gestión y planificación inteligente de la movilidad.

7.2 Conclusiones principales por fuente de datos

El análisis comparativo entre las distintas fuentes de datos ha permitido identificar patrones diferenciales en el rendimiento de los modelos evaluados, así como aspectos clave vinculados

a la naturaleza y topología de cada conjunto.

Gobierno Vasco (sourceId 1)

En este entorno, caracterizado por una concentración de sensores en puntos estratégicos de vías principales, el modelo Trafficformer logró una mejora clara respecto al MLP, reduciendo el MAE de 25.466 a 19.125 y el RMSE de 38.489 a 30.784. El coeficiente de determinación R^2 mejoró sustancialmente, mostrando la capacidad del modelo Transformer para capturar dependencias espaciales locales incluso en redes con dispersión limitada. El patrón de convergencia rápida y estable refuerza la idoneidad de la arquitectura propuesta para este tipo de datos.

Diputación Foral de Bizkaia (sourceId 2)

La amplia y compleja distribución geográfica de los sensores en la red viaria foral puso a prueba la capacidad de generalización espacial de los modelos. Trafficformer destacó especialmente, con una reducción del MAE desde 31.017 a 5.832 y del RMSE de 38.177 a 15.046, así como una mejora de R^2 desde 0.866 a 0.982. Estas métricas evidencian la robustez del modelo en escenarios con gran diversidad espacial y patrones de tráfico heterogéneos.

Ayuntamiento de Bilbao (sourceId 5)

En el entorno urbano denso de Bilbao, donde la correlación espacial entre sensores es máxima, Trafficformer volvió a superar al MLP, disminuyendo el MAE de 213.850 a 175.975 y el RMSE de 343.603 a 306.134. El avance más significativo se observa en el R^2 , que pasó de un valor negativo con MLP (-1.797) a un valor positivo y aceptable con Trafficformer (0.640). Estos resultados subrayan la eficacia del mecanismo de atención espacial en contextos urbanos complejos, donde la interacción entre sensores es altamente dinámica.

En síntesis, el modelo Trafficformer ha mostrado una capacidad superior para adaptarse a diferentes configuraciones espaciales y temporales, confirmando su versatilidad y rendimiento en contextos de tráfico real diversos.

7.3 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 Gobierno Vasco) 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 con sourceId 1, Diputación Foral de Bizkaia con sourceId 2 y Ayuntamiento de Bilbao con sourceId 5), 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.

7.4 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:** Implementar experimentos en infraestructuras de computación en la nube, como Amazon Web Services (AWS), aprovechando la plantilla de CloudFormation ya propuesta en este trabajo para el despliegue automatizado de recursos (EC2, SageMaker, instancias con GPU, etc.). Esta infraestructura permitirá entrenar modelos con arquitecturas más complejas y realizar búsquedas de hiperparámetros de forma eficiente. Además, se recomienda el uso de la funcionalidad *Sweeps* de Weights & Biases (W&B), que permite la exploración automática y paralela de configuraciones de hiperparámetros. La integración de W&B Sweeps con AWS facilita la orquestación de experimentos a gran escala, optimizando tanto el rendimiento del modelo como el uso de los recursos computacionales. Todo ello permitirá acelerar el proceso de optimización y reproducibilidad, así como escalar fácilmente el sistema para futuras líneas de investigación.
- **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.

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, 5 24). *Tercer plan general de carreteras del país vasco 2017-2028*. Retrieved 2025-04-18, from <https://www.euskadi.eus/tercer-plan-general-de-carreteras-del-pais-vasco-2017-2028/web01-a2bideko/es/>

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. Retrieved from https://www.jstage.jst.go.jp/article/transinf/E102.D/8/E102.D_2018EDP7330/_article 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 – Direcciones a los repositorios del TFM

En este anexo se incluyen las direcciones a los principales repositorios públicos generados y empleados a lo largo del Trabajo Fin de Máster. El objetivo es aportar transparencia y facilitar la reproducibilidad de los resultados y desarrollos presentados.

El repositorio principal de la memoria del TFM está disponible en <https://github.com/russellhoff/mia-tfm-memoria>. En él se encuentra el código fuente en \LaTeX , las imágenes, anexos, así como todos los recursos necesarios para compilar y reproducir íntegramente este documento.

El proyecto Data Collector se aloja en <https://github.com/russellhoff/mia-tfm-data-collector>. Este repositorio contiene el código fuente (Kotlin + Spring Boot) y la documentación del artefacto encargado de la recopilación y consolidación de los datos meteorológicos y de tráfico, necesarios para la generación del dataset base del TFM.

El desarrollo principal de modelos predictivos, junto con el código, notebooks y scripts utilizados para el entrenamiento y evaluación de las arquitecturas (MLP y Trafficformer), se encuentra en el repositorio de Trafficformer Bizkaia: <https://github.com/russellhoff/mia-tfm-trafficformer-bizkaia>. En él se documenta todo el proceso de preprocesamiento, entrenamiento y validación de los modelos desarrollados en Python.

Finalmente, se ha empleado la plataforma Weights & Biases para la gestión y trazabilidad de los experimentos realizados. El proyecto, accesible en <https://wandb.ai/jonin/trafficformer-tfm>, recoge los registros de entrenamientos, métricas, visualizaciones, configuración de hiperparámetros y permite comparar de forma interactiva los diferentes resultados obtenidos.

Estos recursos permiten al lector consultar el código, reproducir los experimentos y acceder tanto a los datos como a los modelos y documentación generados a lo largo del trabajo.

Anexo B – Descripción de sensores meteorológicos

En este anexo se presenta un listado exhaustivo de los sensores meteorológicos empleados en la construcción del dataset, el cual se detalla en la Tabla 12. Cada sensor dispone de un identificador único, una abreviatura técnica y una descripción textual de la variable que mide. Todos ellos proceden de estaciones automáticas distribuidas por la CAPV, cuyas mediciones resultan fundamentales para enriquecer los registros 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.).

Tabla 12: Sensores Euskalmet empleados en la construcción del dataset.

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

Continúa en la siguiente página

ID	Nombre	Descripción
94	Amonio	Amonio en mg/l
95	Turbidez	Turbidez del agua en NTU
96	Redox.	Potencial redox del agua en mV
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 (°)
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

Fuente: Elaboración propia.

Anexo C – 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. Esta clase está ubicada en la herramienta `DataCollector`.

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.ReadingXmlRepository
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 D – 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 E – 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 F – 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 13.

Tabla 13: 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

Fuente: Elaboración propia.

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 14.

Tabla 14: Combinaciones evaluadas para el modelo *Trafficformer* (por cada `sourceId`)

NN	SL	LR	BS	NH	ED	NL	FF
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

Continúa en la siguiente página

Tabla 14 – Continuación de la página anterior

NN	SL	LR	BS	NH	ED	NL	FF
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
32	8	0.0005	64	8	128	6	512

Fuente: Elaboración propia.

Leyenda de columnas:

NN: Número de combinación (ID).

SL: Secuencia temporal (*seq_len*).

LR: Learning Rate.

BS: Tamaño de batch (*Batch Size*).

NH: Número de cabezas de atención (*num_heads*).

ED: Dimensión de embedding (*embedding_dim*).

NL: Número de capas encoder (*num_layers*).

FF: Dimensión de feedforward (*ff_hidden_dim*).

Anexo G – 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 15 muestra la información relevante de cada experimento, agrupando por fuente de datos (`sourceId`) y modelo (`model`).

Tabla 15: Métricas principales de los 120 experimentos realizados (validación y test).

ID	SID	M	EP	VL	VMAE	VRMSE	VMAPE	VR2	TL	TMAE	TRMSE	TMAPE	TR2
1	1	mlp	43	3.044e-05	27.3673	64.4961	2.2938	0.6945	2.374e-05	27.7222	53.8060	2.3209	0.7159
2	1	mlp	72	3.036e-05	29.2645	64.5574	2.8053	0.6832	2.367e-05	29.6085	53.8617	2.8498	0.6969
3	1	mlp	46	3.055e-05	27.5140	64.6727	2.2397	0.6938	2.376e-05	27.6645	53.9087	2.2670	0.7031
4	1	mlp	42	3.503e-05	34.2213	73.4357	2.8389	0.5842	2.811e-05	34.7955	64.2097	2.9196	0.4764
5	1	mlp	25	2.326e-05	35.6505	64.0928	2.2577	0.5013	1.062e-05	35.5207	52.5186	2.2112	0.5798
6	1	mlp	64	2.105e-05	31.4137	58.7302	2.0184	0.5899	8.153e-06	31.1537	46.1645	2.0258	0.6214
7	1	mlp	68	1.806e-05	25.3274	50.9038	1.8261	0.7432	5.562e-06	25.4660	38.4893	1.8535	0.7585
8	1	mlp	92	1.866e-05	26.8867	53.1403	1.9448	0.6876	6.113e-06	26.9915	40.7131	1.9587	0.7185
9	1	trafficformer	22	0.2477	19.9167	56.3598	1.7111	0.7572	0.1873	20.0302	45.0394	1.7487	0.7805
10	1	trafficformer	29	0.2483	20.4565	56.5617	1.7293	0.7545	0.1876	20.6164	45.2417	1.7582	0.7801
11	1	trafficformer	48	0.2454	18.9287	55.2697	1.6296	0.7669	0.1848	19.0689	43.9281	1.6384	0.7953
12	1	trafficformer	14	0.7464	103.8004	137.6913	7.7572	-1.1496	0.6344	103.5918	130.0453	7.6483	-1.2256
13	1	trafficformer	33	0.2478	19.8555	56.2415	1.7148	0.7565	0.1870	20.0222	44.9854	1.7479	0.7807
14	1	trafficformer	59	0.2468	19.5109	56.0657	1.6534	0.7629	0.1872	19.6440	44.9525	1.6695	0.7885
15	1	trafficformer	26	0.2465	19.4106	55.7014	1.7091	0.7645	0.1856	19.5326	44.2436	1.7337	0.7943
16	1	trafficformer	29	0.2474	19.6045	56.1621	1.7219	0.7605	0.1866	19.6741	44.7342	1.7470	0.7870
17	1	trafficformer	21	0.2485	19.5906	56.5028	1.7024	0.7572	0.1876	19.6927	45.2101	1.7300	0.7807

Continúa en la siguiente página

Tabla 15 – continuación de la página anterior

ID	SID	M	EP	VL	VMAE	VRMSE	VMAPE	VR2	TL	TMAE	TRMSE	TMAPE	TR2
18	1	trafficformer	23	0.2478	20.2938	56.4201	1.7363	0.7574	0.1868	20.3037	45.0169	1.7616	0.7832
19	1	trafficformer	26	0.2463	19.5574	55.6734	1.8034	0.7649	0.1853	19.6578	44.1670	1.8159	0.7934
20	1	trafficformer	29	0.2461	19.0940	55.4469	1.6443	0.7669	0.1851	19.2760	44.0270	1.6545	0.7955
21	1	trafficformer	33	0.2488	20.0048	56.6343	1.6879	0.7539	0.1878	20.0717	45.2447	1.7061	0.7777
22	1	trafficformer	45	0.2454	20.2758	56.8247	1.6347	0.7573	0.1903	20.3568	46.1513	1.6599	0.7705
23	1	trafficformer	26	0.2476	19.4074	56.0862	1.6707	0.7611	0.1866	19.5452	44.7440	1.6981	0.7856
24	1	trafficformer	26	0.2480	19.6247	56.2368	1.6254	0.7569	0.1869	19.7194	44.8140	1.6628	0.7844
25	1	trafficformer	39	0.1450	19.7264	45.2221	1.7487	0.7868	0.0336	19.5473	31.7484	1.7828	0.8113
26	1	trafficformer	23	0.1461	20.2935	45.6032	1.9343	0.7826	0.0343	20.1076	32.1165	1.9691	0.8087
27	1	trafficformer	31	0.1431	19.3003	44.2984	1.7177	0.7970	0.0317	19.1249	30.7841	1.7450	0.8224
28	1	trafficformer	14	0.7464	103.8004	137.6913	7.7572	-1.1496	0.6344	103.5918	130.0453	7.6483	-1.2256
29	1	trafficformer	31	0.1448	19.6371	45.2183	1.7185	0.7890	0.0345	19.5824	32.0374	1.7547	0.8130
30	1	trafficformer	23	0.1459	19.7391	45.3349	1.8099	0.7859	0.0340	19.5916	31.8775	1.8449	0.8128
31	1	trafficformer	31	0.1448	20.1698	45.3347	1.6983	0.7811	0.0338	20.0034	31.8529	1.7339	0.8061
32	1	trafficformer	16	0.1452	19.5518	45.2186	1.6958	0.7883	0.0339	19.4342	31.8252	1.7341	0.8137
33	1	trafficformer	27	0.1446	19.7041	45.0887	1.7687	0.7868	0.0334	19.5389	31.6368	1.8078	0.8130
34	1	trafficformer	34	0.1445	19.3311	45.0851	1.6798	0.7863	0.0332	19.2849	31.5956	1.7225	0.8145
35	1	trafficformer	26	0.1442	19.4088	44.7938	1.6842	0.7918	0.0333	19.3726	31.5396	1.7211	0.8184
36	1	trafficformer	39	0.1435	19.0724	44.6739	1.6081	0.7891	0.0323	18.9448	31.1205	1.6394	0.8186
37	1	trafficformer	31	0.1454	19.4723	45.1622	1.6451	0.7890	0.0342	19.3863	31.9373	1.6891	0.8099
38	1	trafficformer	33	0.1447	19.9323	45.4050	1.6914	0.7878	0.0345	19.8166	32.1159	1.7312	0.8109
39	1	trafficformer	26	0.1446	19.7380	45.0724	1.7720	0.7896	0.0336	19.6169	31.7322	1.8183	0.8130
40	1	trafficformer	23	0.1453	19.4899	45.1263	1.6001	0.7881	0.0336	19.3999	31.6417	1.6459	0.8147
41	2	mlp	7	1.637e-04	28.0791	36.8578	2.016e14	0.9048	1.647e-04	28.0168	36.8970	2.074e14	0.9041
42	2	mlp	21	1.283e-04	18.5151	29.5182	1.106e14	0.9454	1.280e-04	18.5445	29.4798	1.173e14	0.9451
43	2	mlp	14	1.408e-04	28.0640	36.3934	2.367e14	0.8964	1.413e-04	28.1206	36.4151	2.419e14	0.8955
44	2	mlp	31	1.277e-04	30.9671	38.1444	1.749e14	0.8677	1.277e-04	31.0170	38.1772	1.825e14	0.8661

Continúa en la siguiente página

Tabla 15 – continuación de la página anterior

ID	SID	M	EP	VL	VMAE	VRMSE	VMAPE	VR2	TL	TMAE	TRMSE	TMAPE	TR2
45	2	mlp	5	2.314e-04	27.1730	40.0392	1.084e14	0.8989	2.303e-04	27.1387	40.1287	1.114e14	0.9000
46	2	mlp	9	1.900e-04	40.7031	48.3660	1.955e14	0.7688	1.812e-04	40.3057	47.7234	1.930e14	0.7780
47	2	mlp	30	1.305e-04	31.9518	39.1546	2.629e14	0.8569	1.232e-04	31.4926	38.5120	2.592e14	0.8630
48	2	mlp	28	1.587e-04	31.5733	39.1707	1.923e14	0.8766	1.541e-04	31.3744	39.0507	1.862e14	0.8792
49	2	trafficformer	74	2.978e-03	7.4492	17.8265	1285.9933	0.9771	3.057e-03	7.5054	17.6464	1384.3913	0.9767
50	2	trafficformer	15	4.682e-03	11.4837	21.6525	3045.9387	0.9686	4.622e-03	11.3591	21.3453	3086.9575	0.9681
51	2	trafficformer	84	2.425e-03	6.3972	16.1419	1873.0145	0.9808	2.476e-03	6.4728	15.9212	1995.1944	0.9801
52	2	trafficformer	7	0.2149	110.4337	151.1109	6.287e4	-0.6268	0.2166	110.8956	151.8964	6.405e4	-0.5920
53	2	trafficformer	10	5.858e-03	10.8440	23.7358	3449.0972	0.9644	5.779e-03	10.7060	23.4233	3472.8136	0.9645
54	2	trafficformer	45	4.423e-03	9.7166	20.5375	1977.9753	0.9728	4.325e-03	9.5884	20.0919	2100.8128	0.9726
55	2	trafficformer	30	4.121e-03	11.1829	20.6886	1078.4771	0.9697	4.104e-03	11.1375	20.4366	1195.1174	0.9689
56	2	trafficformer	25	4.053e-03	11.5853	20.6423	1556.4954	0.9701	4.001e-03	11.5154	20.3791	1637.0821	0.9697
57	2	trafficformer	78	3.177e-03	8.1561	18.4248	1775.9973	0.9754	3.153e-03	8.1091	18.0223	1913.2857	0.9753
58	2	trafficformer	74	2.838e-03	7.4612	17.6009	1109.8546	0.9769	2.934e-03	7.4791	17.4388	1271.0317	0.9763
59	2	trafficformer	96	2.332e-03	5.8625	15.5799	818.3318	0.9820	2.417e-03	5.9313	15.3323	989.1318	0.9812
60	2	trafficformer	97	2.222e-03	5.7872	15.3561	1308.6325	0.9827	2.270e-03	5.8316	15.0465	1508.6194	0.9819
61	2	trafficformer	14	5.381e-03	10.1857	22.3016	2140.9245	0.9690	5.300e-03	10.1056	21.9608	2239.9345	0.9684
62	2	trafficformer	97	2.790e-03	7.6145	16.9990	2146.9090	0.9793	2.840e-03	7.6850	16.7022	2237.0270	0.9786
63	2	trafficformer	87	2.656e-03	6.8338	16.4565	992.7690	0.9804	2.707e-03	6.8923	16.1354	1116.6020	0.9797
64	2	trafficformer	16	4.506e-03	10.1624	21.2992	4111.0605	0.9692	4.469e-03	10.1112	21.0899	4151.7699	0.9684
65	2	trafficformer	47	3.979e-03	9.6015	20.3507	3296.3430	0.9721	3.726e-03	9.3158	19.9340	3258.6031	0.9727
66	2	trafficformer	13	6.134e-03	14.3068	25.4692	2581.3380	0.9540	6.070e-03	14.3199	25.4592	2474.1451	0.9542
67	2	trafficformer	41	3.305e-03	8.9235	18.5844	2861.3670	0.9760	3.071e-03	8.6366	18.0453	2807.4265	0.9767
68	2	trafficformer	7	0.2149	110.4337	151.1109	6.287e4	-0.6268	0.2166	110.8956	151.8964	6.405e4	-0.5920
69	2	trafficformer	25	5.733e-03	11.1188	23.5032	1934.1088	0.9635	5.673e-03	11.0758	23.4883	1765.2410	0.9634
70	2	trafficformer	8	9.884e-03	18.8836	33.1211	2657.3864	0.9139	9.848e-03	19.0081	33.2872	2637.3018	0.9134
71	2	trafficformer	17	5.598e-03	14.3002	23.8594	1219.7182	0.9635	5.607e-03	14.2808	23.8828	1104.6116	0.9636

Continúa en la siguiente página

Tabla 15 – continuación de la página anterior

ID	SID	M	EP	VL	VMAE	VRMSE	VMAPE	VR2	TL	TMAE	TRMSE	TMAPE	TR2
72	2	trafficformer	12	0.1648	98.5946	142.5468	6.407e4	-0.5915	0.1666	99.0780	143.4029	6.527e4	-0.5578
73	2	trafficformer	49	3.644e-03	8.9406	19.2400	1143.7613	0.9742	3.504e-03	8.7443	18.8387	1045.1034	0.9750
74	2	trafficformer	54	3.214e-03	8.5824	18.4719	2311.4072	0.9761	2.999e-03	8.3327	18.0042	2312.8432	0.9768
75	2	trafficformer	10	5.234e-03	11.8576	22.5139	1226.5846	0.9669	5.063e-03	11.6767	22.2087	1097.8127	0.9678
76	2	trafficformer	62	2.588e-03	7.0409	16.5870	1251.8990	0.9801	2.482e-03	6.8546	16.2386	1088.1602	0.9805
77	2	trafficformer	23	5.796e-03	11.2736	23.9760	1958.2437	0.9610	5.666e-03	11.1748	23.8099	1894.3138	0.9612
78	2	trafficformer	66	3.646e-03	8.3096	19.0715	1829.7486	0.9751	3.521e-03	8.0638	18.6899	1735.9587	0.9756
79	2	trafficformer	32	4.475e-03	11.0532	21.1526	1033.3953	0.9693	4.330e-03	10.8695	20.8977	865.7608	0.9698
80	2	trafficformer	35	4.969e-03	11.0336	22.0391	864.7823	0.9674	4.743e-03	10.7854	21.6739	725.9049	0.9679
81	5	mlp	10	4.870e-04	228.4659	362.4311	4.103e5	0.5473	4.908e-04	225.9974	359.4091	4.038e5	-1.1422
82	5	mlp	41	4.613e-04	197.5085	333.3821	1.881e5	0.6517	4.674e-04	196.4247	332.6201	1.887e5	0.2319
83	5	mlp	40	4.561e-04	216.2649	345.7661	4.848e5	0.5234	4.632e-04	213.8501	343.6029	4.827e5	-1.7974
84	5	mlp	47	4.591e-04	199.9680	334.9659	2.308e5	0.6410	4.651e-04	199.0852	334.0663	2.297e5	0.0554
85	5	mlp	37	4.672e-04	211.9732	345.1551	3.817e5	0.5696	4.674e-04	211.5616	347.1920	3.834e5	-0.6087
86	5	mlp	33	4.722e-04	210.1653	344.6294	3.004e5	0.6006	4.739e-04	209.3760	346.5635	2.916e5	-0.1825
87	5	mlp	16	4.786e-04	216.9454	351.9593	3.232e5	0.5786	4.795e-04	215.4682	353.2977	3.084e5	-0.3439
88	5	mlp	20	4.953e-04	230.5105	363.3249	2.886e5	0.5411	4.957e-04	229.3319	364.1892	2.804e5	0.0149
89	5	trafficformer	54	0.1547	182.5938	313.9699	3.330e5	0.6843	0.1579	181.9509	312.1547	3.323e5	0.4182
90	5	trafficformer	61	0.1551	178.0730	311.8513	1.669e5	0.6961	0.1557	176.8071	307.5978	1.655e5	0.5969
91	5	trafficformer	27	0.1544	178.4209	310.0991	1.178e5	0.7015	0.1568	177.5550	307.9404	1.199e5	0.6651
92	5	trafficformer	13	1.0018	799.8041	974.6226	5.867e6	-7.6879	0.9794	787.6951	963.4894	5.872e6	-77.3090
93	5	trafficformer	59	0.1551	179.7787	311.9973	2.363e5	0.6955	0.1570	179.4539	309.0864	2.358e5	0.5616
94	5	trafficformer	46	0.1549	180.2914	311.3468	1.693e5	0.6959	0.1583	179.5983	309.7968	1.689e5	0.5922
95	5	trafficformer	35	0.1539	182.0230	311.5962	2.482e5	0.6935	0.1543	180.6064	307.6337	2.489e5	0.4671
96	5	trafficformer	37	0.1544	179.4542	311.0724	1.723e5	0.6919	0.1554	178.3453	307.6291	1.723e5	0.6085
97	5	trafficformer	68	0.1545	177.5781	309.8515	1.852e5	0.6980	0.1562	176.8873	307.4111	1.842e5	0.6145
98	5	trafficformer	51	0.1549	179.2893	311.2842	2.364e5	0.6962	0.1567	178.5253	308.5846	2.331e5	0.5648

Continúa en la siguiente página

Tabla 15 – continuación de la página anterior

ID	SID	M	EP	VL	VMAE	VRMSE	VMAPE	VR2	TL	TMAE	TRMSE	TMAPE	TR2
99	5	trafficformer	30	0.1528	176.3563	308.0406	1.741e5	0.7029	0.1561	175.9748	306.1345	1.743e5	0.6404
100	5	trafficformer	32	0.1549	178.0815	310.2575	2.301e5	0.6954	0.1570	177.3610	307.3034	2.270e5	0.5960
101	5	trafficformer	59	0.1559	184.7074	314.6608	3.516e5	0.6849	0.1573	183.5689	311.8641	3.506e5	0.3796
102	5	trafficformer	56	0.1541	178.7904	311.2272	2.081e5	0.6941	0.1567	177.7445	308.0763	2.074e5	0.5787
103	5	trafficformer	29	0.1545	178.1218	310.3059	1.545e5	0.6992	0.1563	176.6220	307.4126	1.526e5	0.6568
104	5	trafficformer	34	0.1545	180.4160	311.8224	2.028e5	0.6906	0.1562	179.8230	308.6179	2.030e5	0.5514
105	5	trafficformer	32	0.1595	182.5883	313.0840	1.917e5	0.6888	0.1587	182.3305	318.1452	1.822e5	0.6183
106	5	trafficformer	24	0.1616	190.4618	318.7317	3.607e5	0.6795	0.1598	190.5159	322.7418	3.550e5	0.4108
107	5	trafficformer	25	0.1595	184.2969	314.2691	2.500e5	0.6865	0.1592	184.6056	318.8572	2.487e5	0.5596
108	5	trafficformer	13	1.0018	799.8041	974.6226	5.867e6	-7.6879	0.9794	787.6951	963.4894	5.872e6	-77.3090
109	5	trafficformer	56	0.1588	179.9533	312.6102	1.654e5	0.6923	0.1585	179.9851	317.2295	1.583e5	0.6490
110	5	trafficformer	65	0.1595	180.2213	311.6283	1.467e5	0.6929	0.1604	180.7000	317.3300	1.336e5	0.6460
111	5	trafficformer	32	0.1597	180.0112	312.4643	1.591e5	0.6903	0.1607	180.4107	318.3610	1.455e5	0.6315
112	5	trafficformer	57	0.1584	179.9658	311.3314	2.041e5	0.6905	0.1579	180.0303	316.0017	1.997e5	0.5754
113	5	trafficformer	32	0.1593	180.8343	312.7653	1.449e5	0.6920	0.1579	180.4679	317.6694	1.335e5	0.6571
114	5	trafficformer	54	0.1590	179.2704	311.3642	1.651e5	0.6904	0.1576	179.1310	315.9979	1.584e5	0.6184
115	5	trafficformer	37	0.1580	183.5876	311.8817	2.892e5	0.6886	0.1570	183.1178	315.6936	2.862e5	0.5156
116	5	trafficformer	28	0.1576	180.9950	310.2011	2.396e5	0.6954	0.1572	181.1000	315.4476	2.293e5	0.6111
117	5	trafficformer	32	0.1602	182.9043	314.4888	1.692e5	0.6903	0.1594	182.5033	319.4667	1.618e5	0.5960
118	5	trafficformer	37	0.1601	182.7138	315.3447	2.333e5	0.6799	0.1597	182.1778	319.7605	2.234e5	0.5015
119	5	trafficformer	32	0.1587	179.3557	310.5347	1.423e5	0.6935	0.1588	179.7756	317.2663	1.360e5	0.6270
120	5	trafficformer	30	0.1590	180.8759	313.7183	1.663e5	0.6896	0.1605	181.9994	321.2527	1.554e5	0.5859

Fuente: Elaboración propia.

Por motivos de legibilidad en tablas extensas, se ha aplicado sombreado alterno a las filas.

Leyenda de columnas:

ID: Número de experimento.

SID: sourceld (fuente de datos).

M: Modelo.

EP: Epoch óptima.

VL: Validation Loss.

VMAE: Validation MAE.

VRMSE: Validation RMSE.

VMAPE: Validation MAPE.

VR2: Validation R^2 .

TL: Test Loss.

TMAE: Test MAE.

TRMSE: Test RMSE.

TMAPE: Test MAPE.

TR2: Test R^2 .

Anexo H – 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 (sourceld 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.

Sourceld 1

Figura 23: Gráfico de dispersión (predicción vs. valores reales) para Sourceld 1.

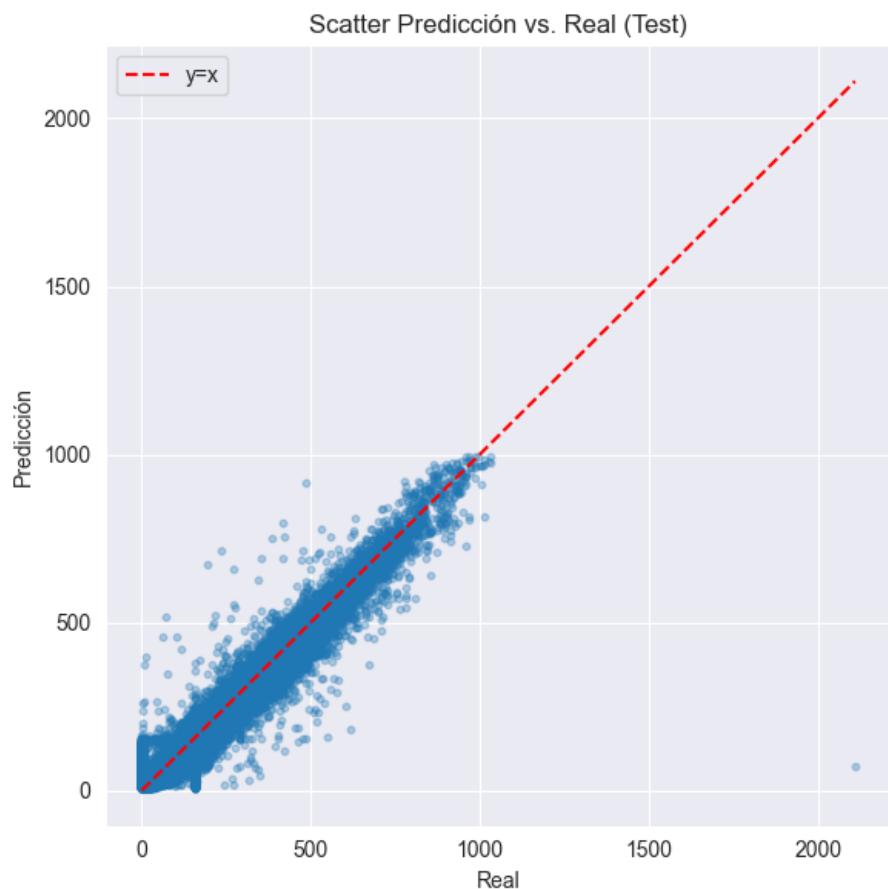


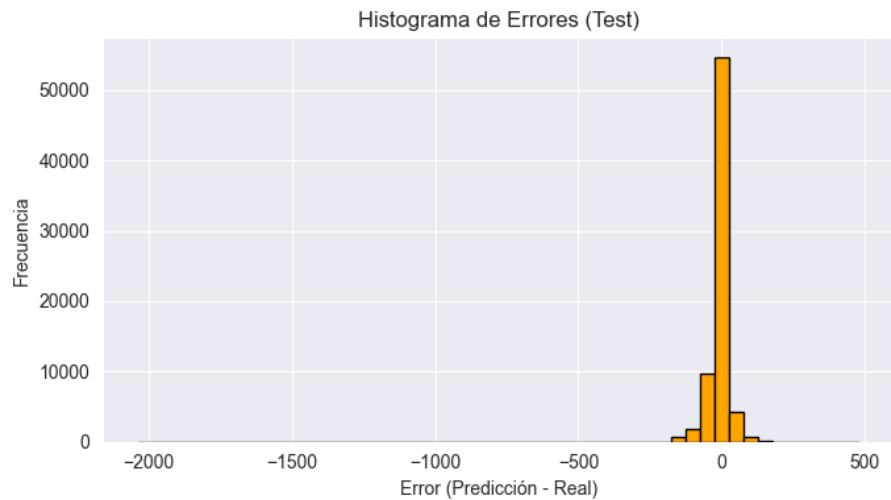
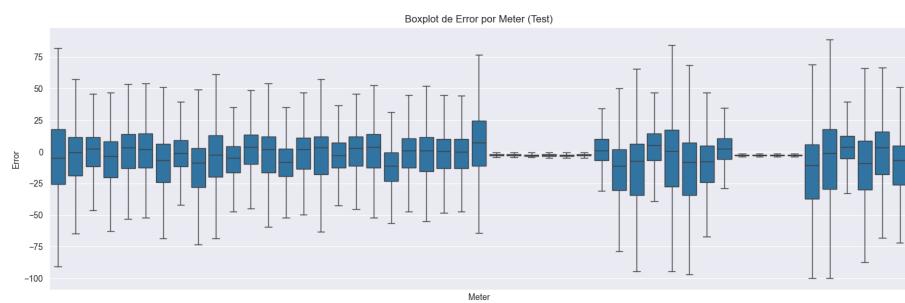
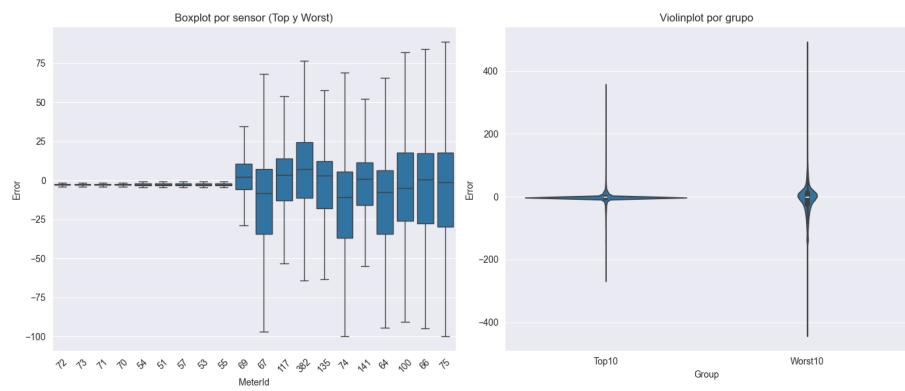
Figura 24: Histograma del error absoluto (residual) para SourceId 1.**Figura 25:** Boxplot del error absoluto por sensor para todos los sensores del SourceId 1.**Figura 26:** Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del SourceId 1.

Figura 27: Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 1.

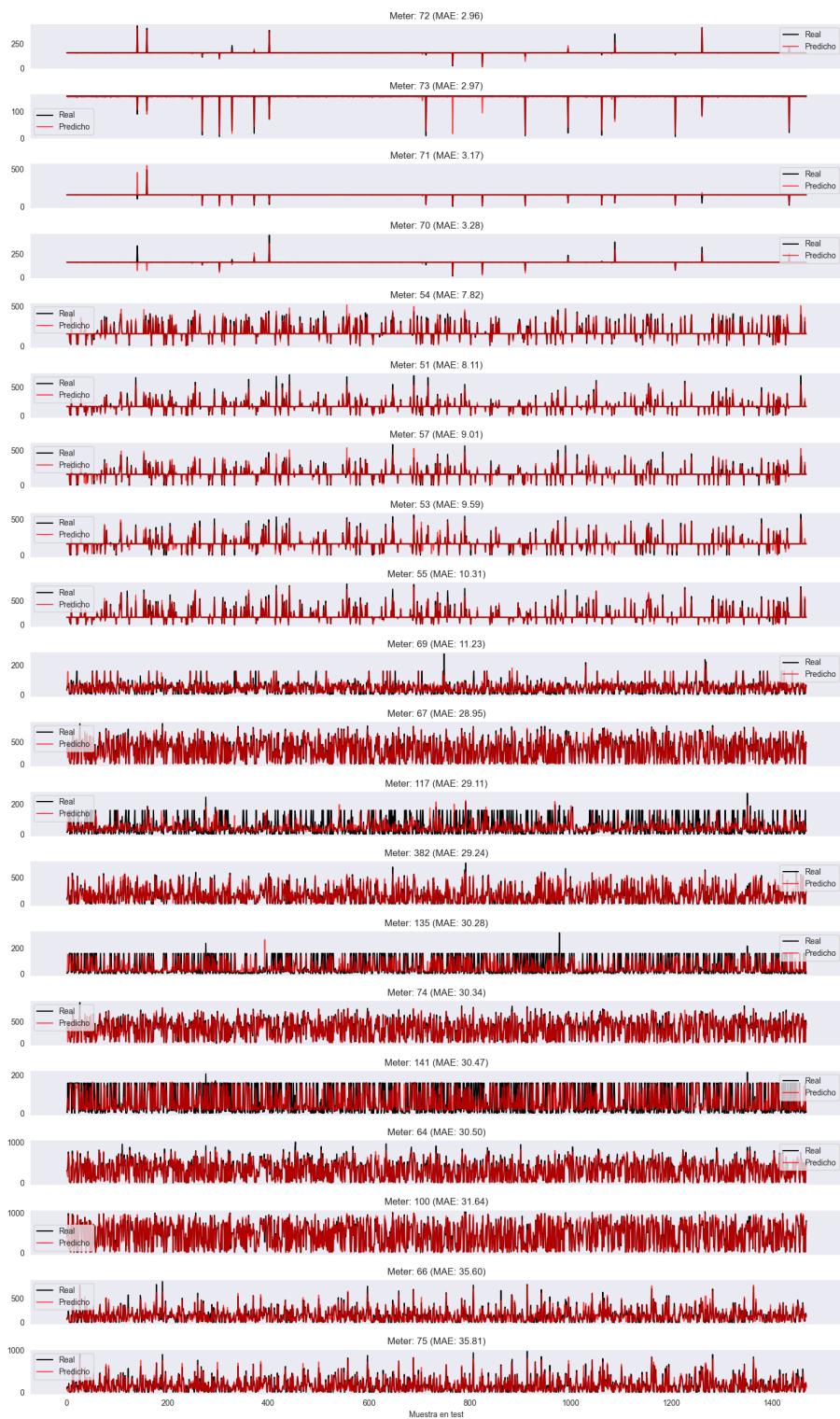
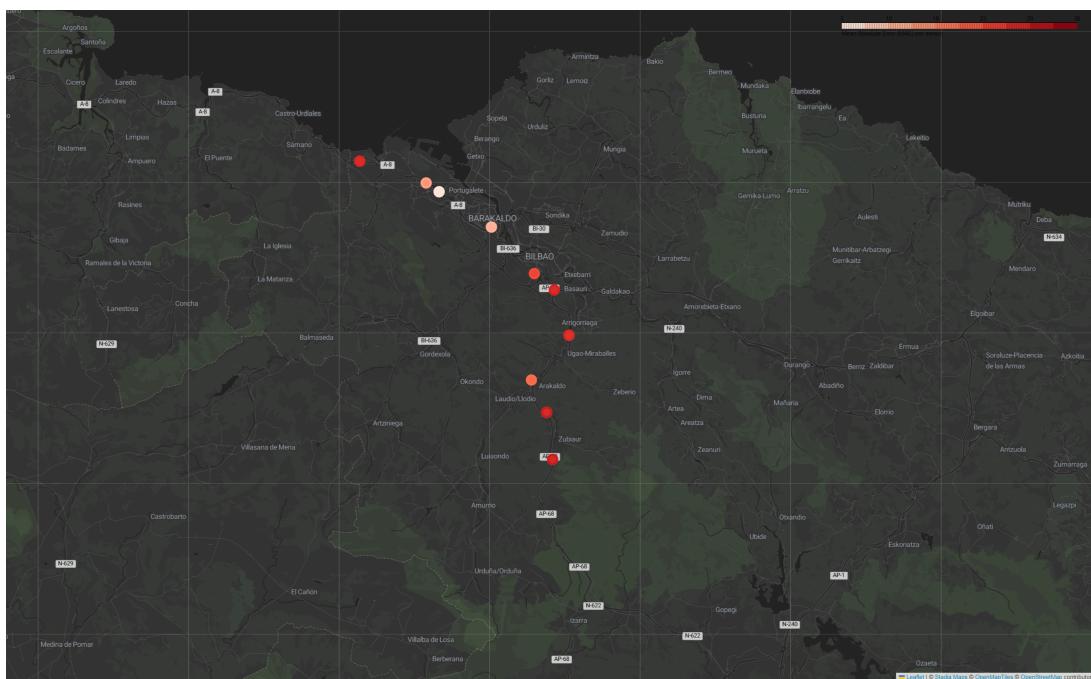


Figura 28: Mapa espacial de errores (MAE) para sensores del SourceId 1.



Sourceld 2

Figura 29: Gráfico de dispersión (predicción vs. valores reales) para Sourceld 2.

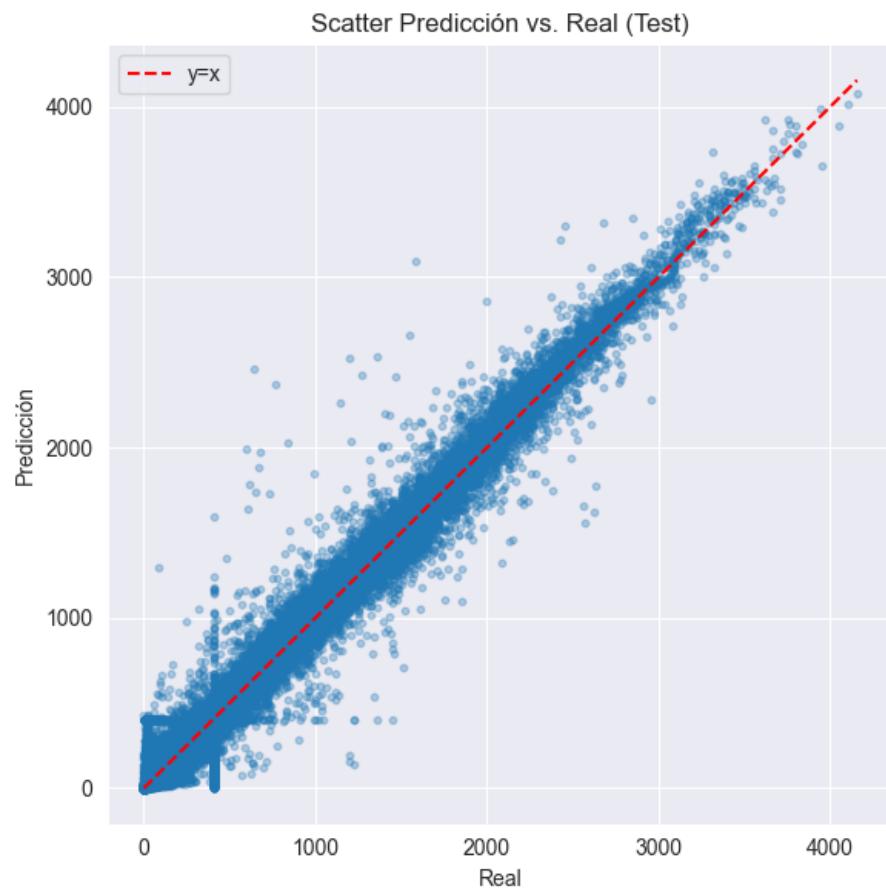


Figura 30: Histograma del error absoluto (residual) para Sourceld 2.

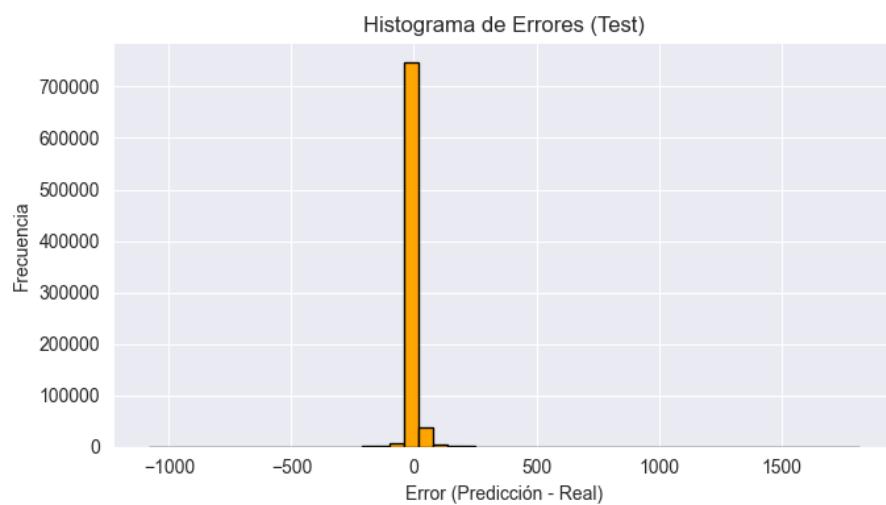


Figura 31: Boxplot del error absoluto por sensor para todos los sensores del Sourceld 2.

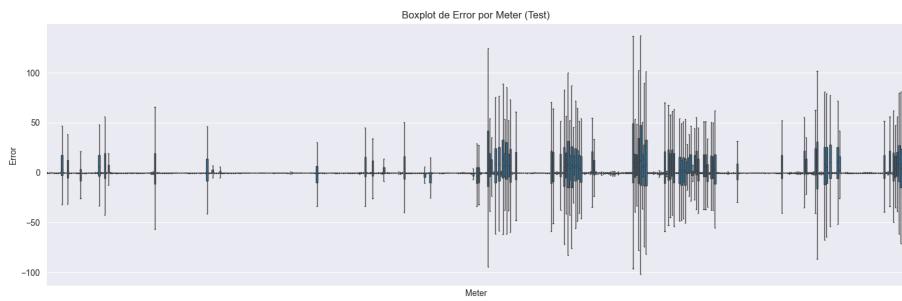


Figura 32: Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 2.

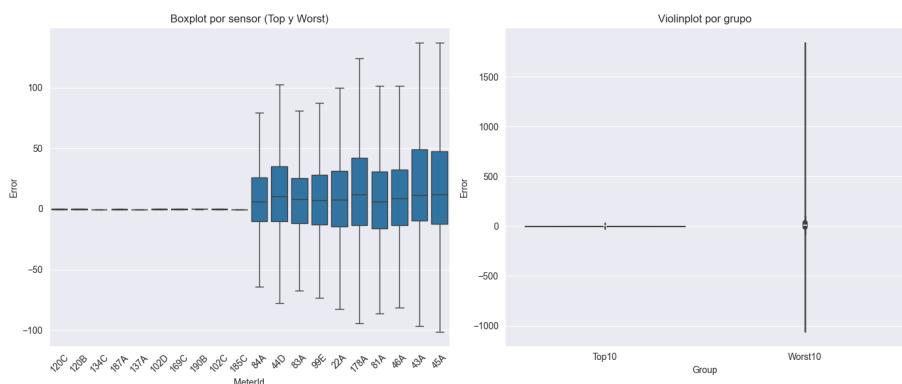


Figura 33: Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 2.

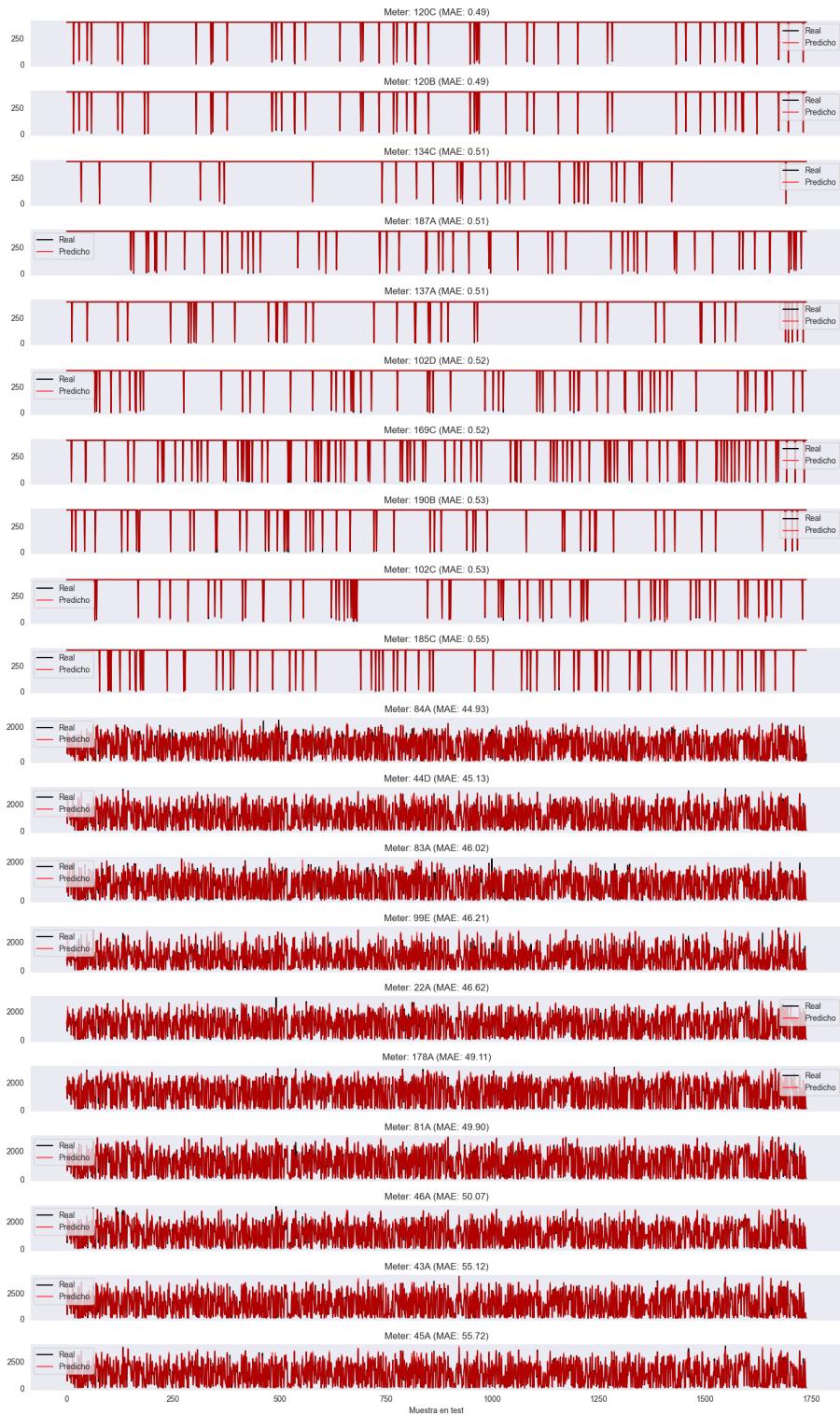
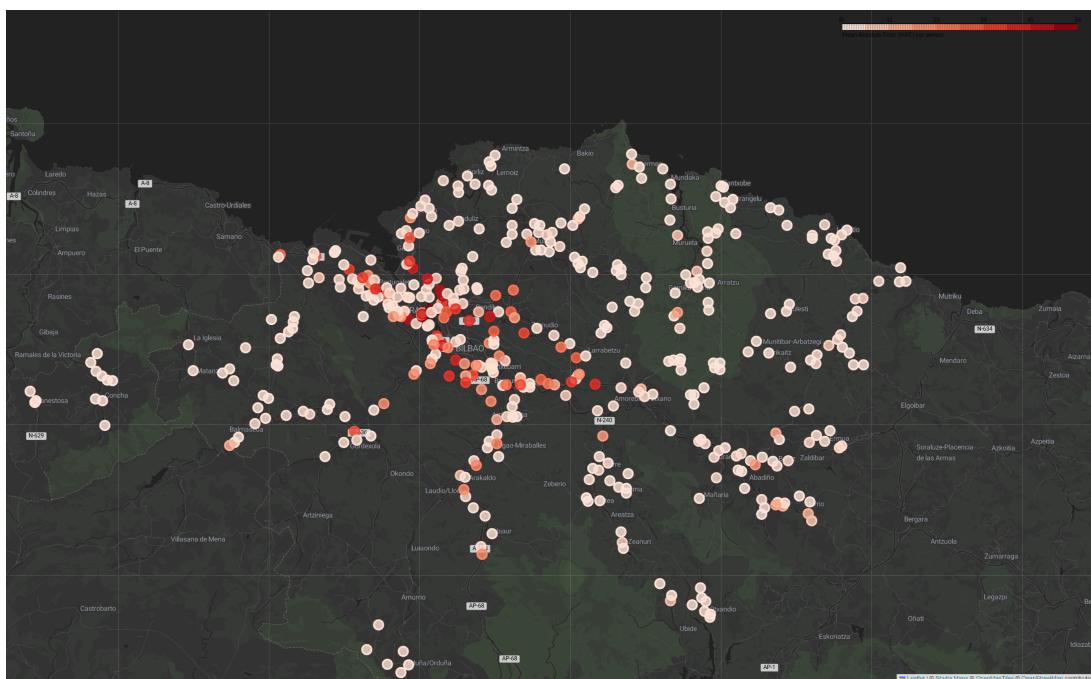


Figura 34: Mapa espacial de errores (MAE) para sensores del SourceId 2.



Sourceld 5

Figura 35: Gráfico de dispersión (predicción vs. valores reales) para Sourceld 5.



Figura 36: Histograma del error absoluto (residual) para Sourceld 5.

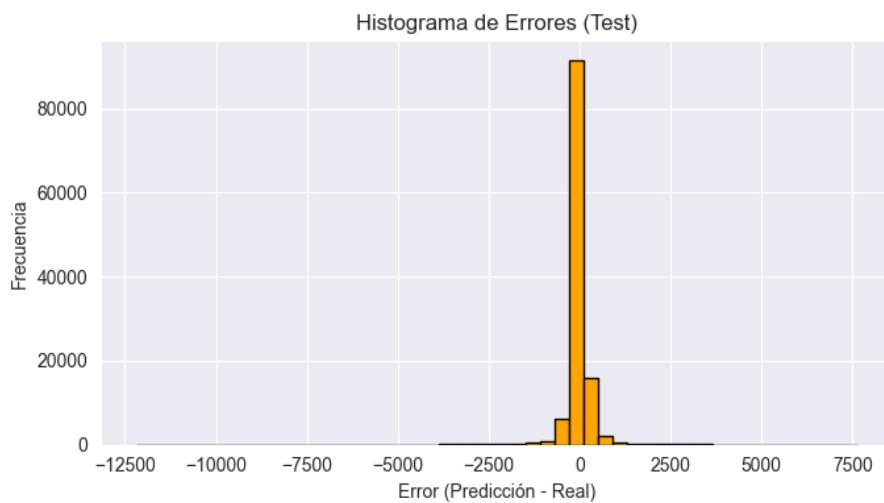


Figura 37: Boxplot del error absoluto por sensor para todos los sensores del Sourceld 5.

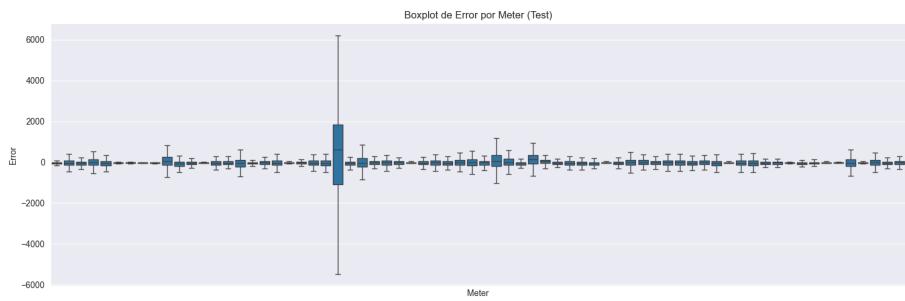


Figura 38: Boxplot y Violinplot de errores para los 10 mejores y los 10 peores sensores del Sourceld 5.

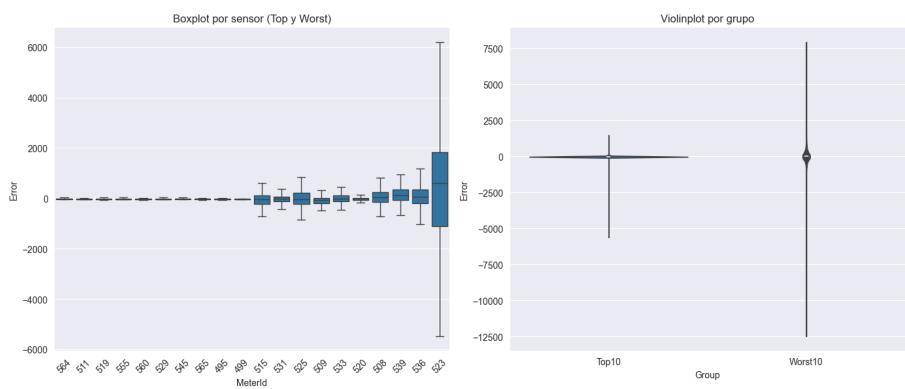


Figura 39: Serie temporal comparativa (predicción vs. real) para los 10 mejores y 10 peores sensores del Sourceld 5.

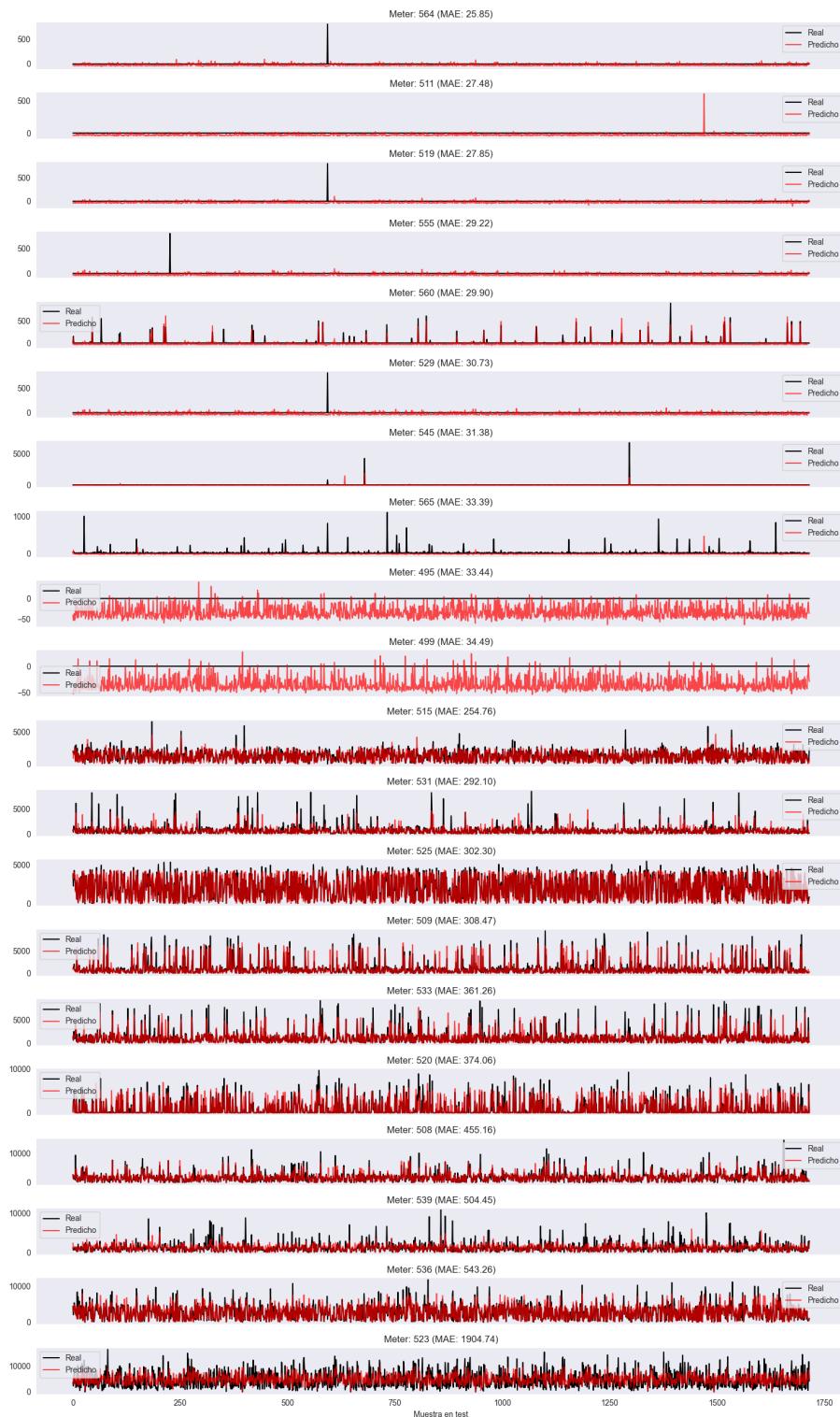


Figura 40: Mapa espacial de errores (MAE) para sensores del SourceId 5.

