Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Inteligencia Artificial

Predicción de tráfico mediante aprendizaje profundo y Transformers

Trabajo fin de máster presentado por:	Jon Inazio Sánchez Martínez
Tipo de trabajo:	Desarrollo
Director:	Omar Velázquez López
Fecha:	24 de mayo del 2025

Resumen

La predicción precisa del tráfico es clave para optimizar la movilidad y el uso de infraestructuras, especialmente en la Comunidad Autónoma del País Vasco y más en concreto en la provincia de Bizkaia, donde la orografía, densidad urbana y condiciones meteorológicas suponen retos añadidos. Sin embargo, la complejidad inherente a los datos de tráfico, caracterizados por fuertes correlaciones espacio-temporales y patrones no lineales, dificulta su modelado y predicción exacta. Este trabajo propone un modelo de predicción basado en redes neuronales con arquitectura Transformer, capaz de capturar dependencias espacio-temporales complejas mediante mecanismos de atención que asignan pesos dinámicos a segmentos relevantes. Se emplearán datos abiertos de tráfico y meteorología procedentes de la provincia de Bizkaia. La validación se realizará con datos reales y se comparará frente a modelos tradicionales de redes neuronales, demostrando mejoras significativas en precisión. El sistema resultante ofrece una herramienta robusta y eficiente para la gestión del tráfico en tiempo real, con potencial para anticipar congestiones y optimizar la toma de decisiones operativas.

Palabras clave: Predicción del tráfico, Aprendizaje profundo, Transformers, Redes neuronales, Correlación espacio-temporal, Datos abiertos, Comunidad Autónoma del País Vasco, Bizkaia

Abstract

Accurate traffic forecasting is essential for optimising mobility and infrastructure usage, especially in the Autonomous Community of the Basque Country (CAPV) and particularly in the province of Bizkaia, where the region's orography, urban density, and weather variability present additional challenges. However, the inherent complexity of traffic data, characterised by strong spatio-temporal correlations and non-linear patterns, makes its modelling and prediction difficult. This work proposes a prediction model based on neural networks with Transformer architecture, capable of capturing complex spatio-temporal dependencies through attention mechanisms that dynamically assign weights to relevant segments. Bizkaia's open traffic and weather datasets will be used. The model will be validated with real-world data and compared against traditional neural network models. The outcome will be a robust and efficient tool for real-time traffic management, with the potential to anticipate congestion and support the optimisation of operational decision-making.

Keywords: Traffic forecasting, Deep learning, Transformers, Neural networks, Spatio-temporal correlation, Open data, Autonomous Community of the Basque Country, Bizkaia

Índice de contenidos

Índice de figuras

Índice de tablas

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, disrupciones u otros eventos que puedan comprometer la fluidez de la circulación. La inteligencia Artificial y, en particular, del Aprendizaje Profundo, ha impulsado el desarrollo de modelos que integran datos en tiempo real y capturan patrones complejos mediante el uso de arquitecturas sofisticadas.

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

1.2 Planteamiento del problema

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

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

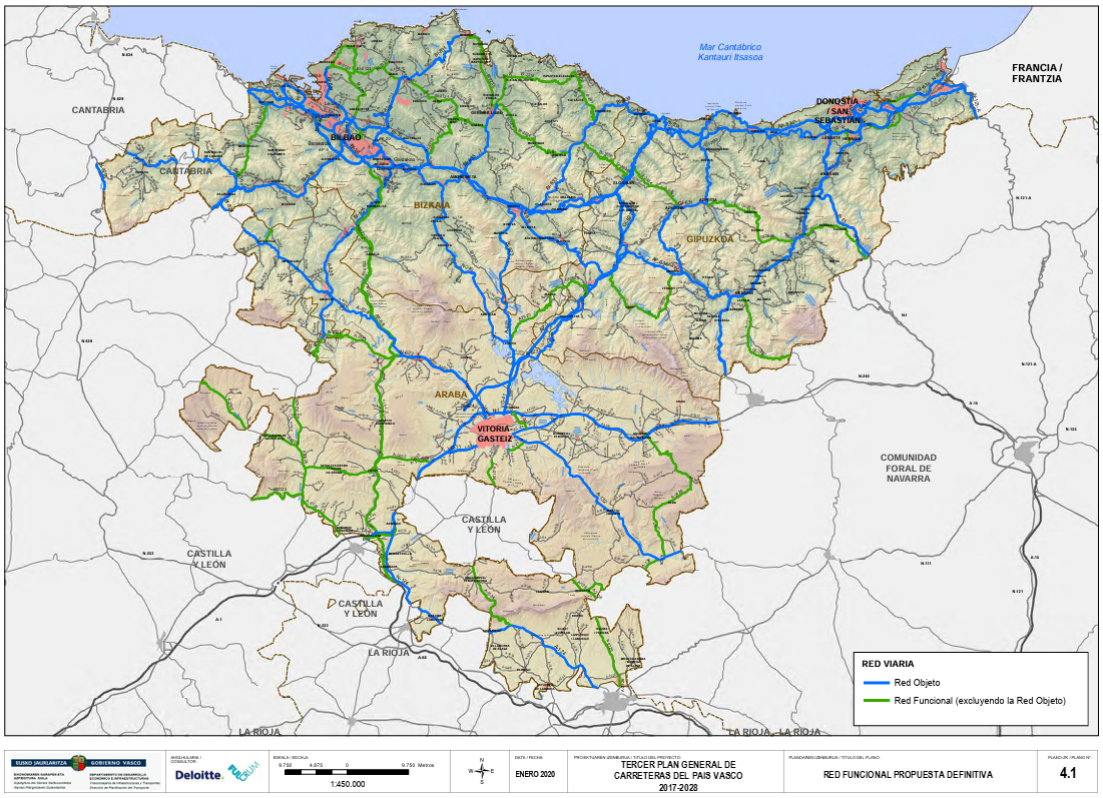

Figura 1: Red viaria de la CAPV, extraído de Gobierno Vasco, Eusko Jaurlaritza (2020)

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

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

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

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

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

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

2 Contexto y estado del arte

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

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

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

2.1 Técnicas existentes para la predicción del tráfico

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

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

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

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

2.1.1 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.

2.1.2 Modelos clásicos de Machine Learning

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

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

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

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

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

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

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

2.1.3 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.

2.1.4 Redes neuronales recurrentes

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

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

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

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

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

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

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

2.1.5 Graph Neural Networks

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

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

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

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

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

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

2.1.6 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. (2025a), titulado *Transformer-based short-term traffic forecasting model considering traf-*

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

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

(1) Extracción de características temporales mediante Multi Layer Perceptron o Perceptrones Multi Capa.

(2) Interacción espacial basada en codificadores Transformer con máscaras de atención topológicas.

(3) Predicción de velocidades mediante una red MLP final.

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

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

El trabajo de Chang et al. (2025a) 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. (2025a) 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

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. (2025a), 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 10ª 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 7ª generación, con 16 GB de RAM DDR4 y dos discos duros de 4 TB configurados en RAID 1. Este servidor albergará una base de datos MongoDB para almacenamiento estructurado de datos y MinIO como sistema de almacenamiento tipo S3, útil para almacenamiento masivo y backups. Todos esos servicios van a funcionar como contenedores Docker.

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

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

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

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

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

3.3 Metodología de trabajo

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

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

  - **Codificador Transformer**: mediante multi-head attention, el modelo extrae relaciones espaciales entre nodos cercanos, permitiendo una representación contextualizada del tráfico.
  - **Predicción**: se proyectan las representaciones espacio-temporales a una predicción del número de vehículos que circularán por cada punto de la red en el siguiente intervalo.

- Entrenamiento y evaluación.
  - **Métricas**: se utilizarán MAE, RMSE y MAPE para comparar la predicción de conteos de vehículos.
  - **Modelos base**: se tendrá como referencia el modelo MLP desarrollado anteriormente.
  - **Validación**: se aplicará validación temporal (walk-forward) para simular condiciones reales de predicción.

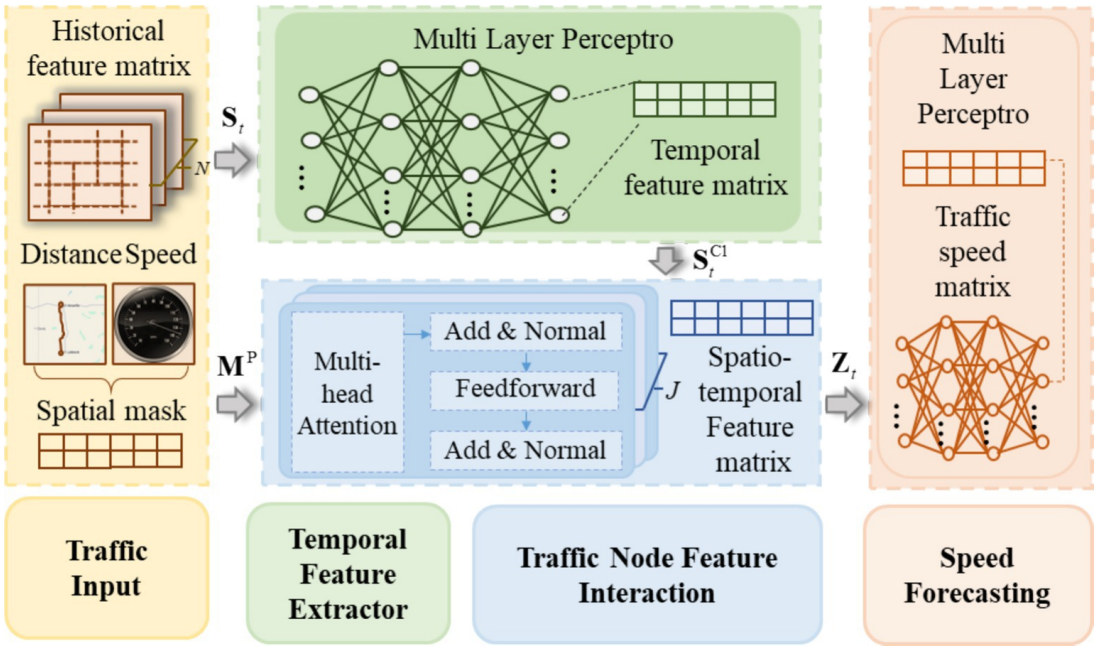

Figura 2: Arquitectura general del modelo Trafficformer Chang et al. (2025a)

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

- Extracción y almacenamiento de datos. Desarrollo de un módulo en Kotlin para conectar

con las públicas de tráfico y meteorología de Open Data Euskadi. Almacenamiento de los datos crudos en MongoDB y objetos binarios (imágenes, CSVs temporales) en MinIO.

- Preprocesamiento y transformación de datos. Conversión de datos crudos en estructuras listas para entrenamiento, mediante scripts en Python. Aplicación de técnicas de normalización, resampleo temporal y gestión de datos faltantes.

- Entrenamiento de modelos neuronales. Implementación de un modelo base MLP para establecer una línea base de rendimiento. Posterior implementación de un modelo Transformer adaptado al tráfico, con atención espacio-temporal y uso de máscaras de conectividad vial.

- Evaluación comparativa. Evaluación de los modelos con un conjunto de métricas estándar (MAE, RMSE, MAPE). Comparativa entre modelos para seleccionar el más adecuado para el caso de uso.

- Validación y conclusiones. Validación en escenarios reales y extracción de conclusiones sobre la utilidad del modelo. Estudio de la viabilidad de su aplicación en entornos reales, como el soporte a ITS o planificación urbana.

4 Construcción del dataset y arquitectura del sistema

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

4.1 Fuentes de datos y análisis de disponibilidad

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

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

La cobertura de los datos obtenidos se representa en la tabla 2, extraída y adaptada de la documentación técnica del repositorio del proyecto.

Tabla 2: Cobertura de datos por fuente y tipo.

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

Leyenda:

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

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

Las incidencias se obtuvieron completamente para todos los sourceId.

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

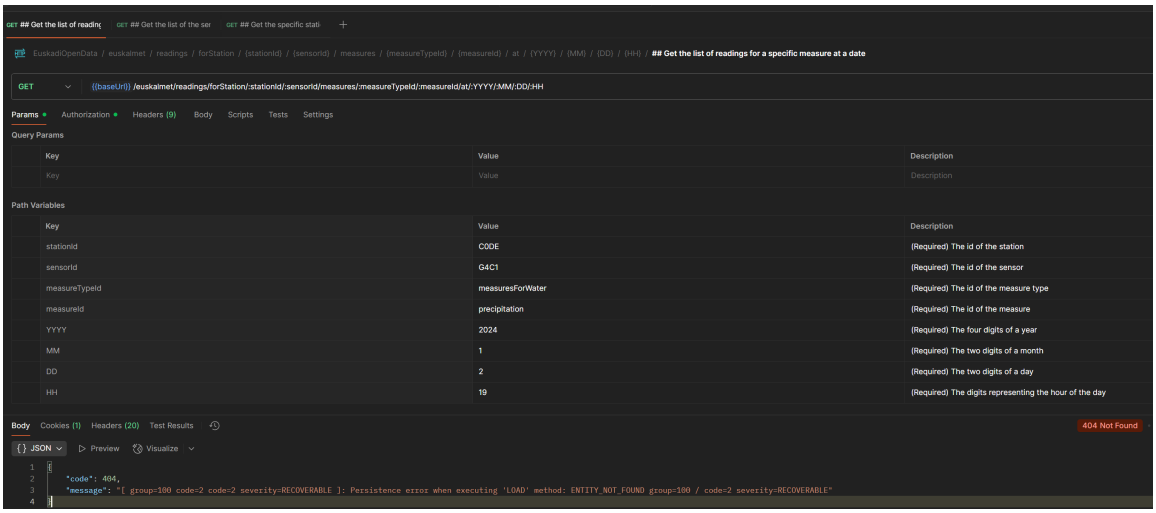

Figura 3: Ejemplo de error a la hora de consultar el API de Euskalmet.

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

4.1.1 Modelo de datos almacenado

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

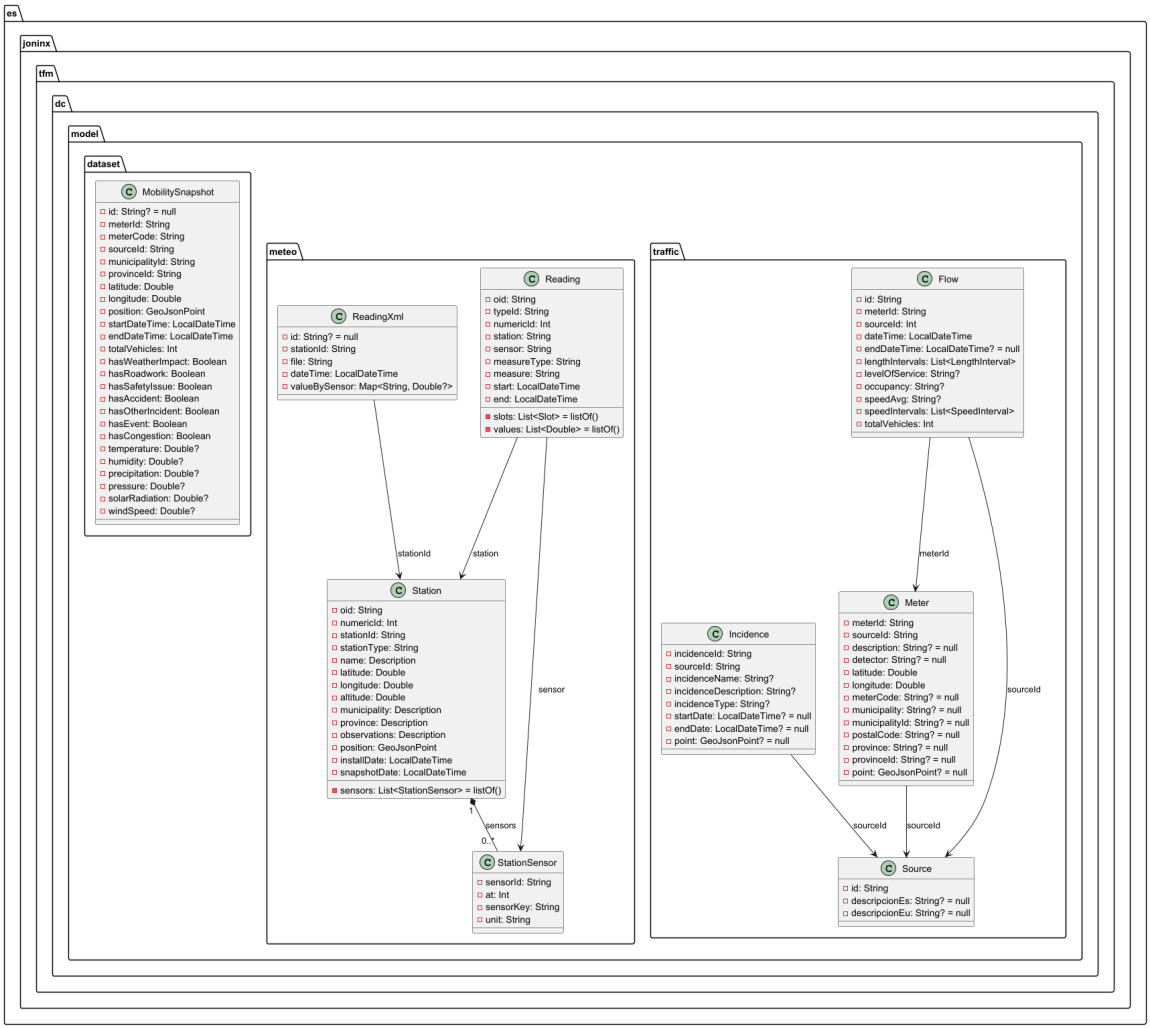

Figura 4: Modelo de clases en UML de las entidades principales del sistema.

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

4.1.1.1 Flow

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

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

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

4.1.1.2 Meter

Contiene la información estática de los aforadores:

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

4.1.1.3 Incidence

Recoge información sobre incidencias viales que pueden afectar al tráfico:

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

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

4.1.1.4 Source

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

4.1.1.5 Reading

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

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

4.1.1.6 ReadingXml

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

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

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

4.1.1.7 Station

Define las estaciones meteorológicas que proporcionan las lecturas:

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

4.1.1.8 StationSensor

Define la configuración física de un sensor en una estación:

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

4.1.1.9 MobilitySnapshot

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

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

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

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

4.2 Recolección, arquitectura y patrones de diseño empleados

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

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

El sistema de integración y procesamiento de datos ha sido desarrollado en Kotlin, un lenguaje moderno que combina características orientadas a objetos y funcionales. La clase `MobilitySnapshotGene` 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.

4.3.1 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 3 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 3: 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 4.

Tabla 4: Agrupación final de incidencias para el modelo predictivo

Categoría general	**Tipos incluidos**	**Total casos**	**Variable sugerida**
`WEATHER`	Puertos de montaña, Viabilidad 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` (opcional)

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

Listing 1: Variables mínimas de incidencias en MobilitySnapshot

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

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

Listing 2: Variables opcionales

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

4.3.2 Selección e integración de variables meteorológicas

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

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

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

Las variables seleccionadas, junto con su justificación práctica, se muestran en la tabla 5.

Tabla 5: 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.

4.3.3 Fusión e integración: la clase `MobilitySnapshot`

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

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

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

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

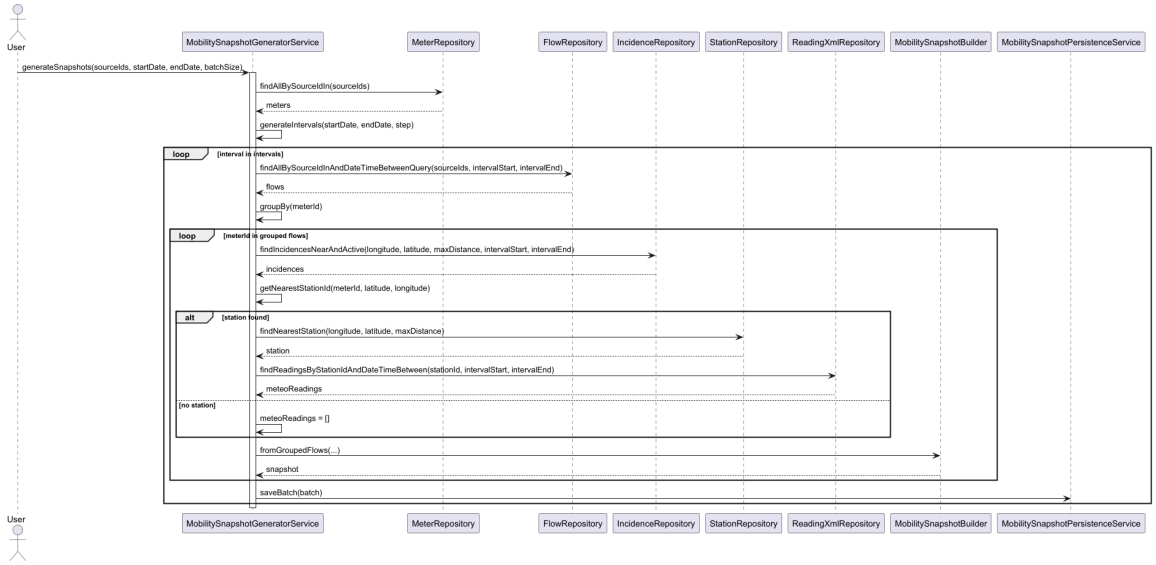

Figura 5: Diagrama de secuencia de integración y persistencia en `MobilitySnapshot`.

El código fuente completo del componente encargado de llevar a cabo esta integración puede consultarse en el Anexo 4.3.4, 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 `MobilitySnapsho` mediante el `MobilitySnapshotBuilder`.

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

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

El algoritmo se describe de la siguiente forma.

Descripción del funcionamiento:

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

Ejemplo práctico:

Listing 4: Ejemplo de uso con intervalos de 30 minutos

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

El resultado de este ejemplo sería:

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

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

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

4.3.4 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 6: Comparativa entre intervalos temporales posibles para el resampling

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

En la Tabla 6 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 (`MobilitySnapshotGeneratorServi`) 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.

Lista de Acrónimos y Abreviaturas

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

ARIMA AutoRegressive Integrated Moving Average.

CAPV Comunidad Autónoma del País Vasco.

CNN Convolutional Neural Networks o Redes Neuronales Convolucionales.

DL Deep Learning o Aprendizaje Profundo.

GAT Graph Attention Networks.

GCN Graph Convolutional Networks.

GGNN Gated Graph Neural Networks.

GNN Graph Neural Networks.

GPU Graphics Processing Unit.

GRU Gated Recurrent Units.

ITS Sistemas Inteligentes de Transporte.

KNN K-Nearest Neighbors.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error o Error Medio Absoluto.

MAPE Mean Absolute Percentage Error o Error Porcentual Absoluto Medio.

MLP Multi Layer Perceptron o Perceptrones Multi Capa.

MRE Mean Relative Error o Error Medio Relativo.

RAM Random Access Memory.

RF Random Forests.

RMSE Root Mean Square Error o Error Cuadrático Medio.

RNN Recurrent Neural Networks o Redes Neuronales Recurrentes.

SARIMA Seasonal AutoRegressive Integrated Moving Average.

SSD Solid State Drive.

SVM Support Vector Machines.

SVR Support Vector Regression.

Bibliografía

Aditya, F., Nasution, S., & Virgono, A. (2020, 10). Traffic flow prediction using sumo application with k-nearest neighbor (knn) method. *International Journal of Integrated Engineering*, *12*. Retrieved from https://www.researchgate.net/publication/346622922_Traffic_Flow_Prediction_using_SUMO_Application_with_K-Nearest_Neighbor_KNN_Method doi: 10.30880/ijie.2020.12.07.011

Chang, A., Ji, Y., & Bie, Y. (2025a). 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

Chang, A., Ji, Y., & Bie, Y. (2025b). Transformer-based short-term traffic forecasting model considering traffic spatiotemporal correlation. *Frontiers in Neurorobotics*, *19*. 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/abreviaturas.txt. ([Archivo de texto]. Open Data Euskadi.)

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

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

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

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

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

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

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

Katambire, V. N., Musabe, R., Uwitonze, A., & Mukanyiligira, D. (2023). Forecasting the traffic flow by using arima and lstm models: Case of muhima 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_files/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

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

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61-80. doi: 10.1109/TNN.2008.2005605

UnfoldAI. (2024). *Keras vs pytorch — which dl framework to choose in 2024?* Retrieved 2025-05-23, from https://unfoldai.com/keras-vs-pytorch-in-2024 (Accedido el 23 de mayo de 2025)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. Retrieved from https://arxiv.org/abs/1706.03762 (Publicado en 2017, última revisión (v7): 2023) doi: 10.48550/arXiv.1706.03762

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

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

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

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

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

Anexo A – Descripción de sensores meteorológicos

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

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

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

ID	Nombre	Descripción
97	Mat.Org.	Materia orgánica medida como demanda de oxígeno
11	VelMed	Velocidad media del viento en km/h
22	Tem.Sue	Temperatura del suelo en ºC
B0	Visibili	Visibilidad en metros
B1	Nivelm 1	Nivel del mar (sensor 1) en metros
B2	Nivelm 2	Nivel del mar (sensor 2) en metros
B3	Ola Med	Altura media de ola en metros
B4	Ola Max	Altura máxima de ola en metros
B5	Ola Sig	Altura significante de ola en metros
B6	Ola Per	Periodo de oleaje en segundos
B7	Pre MSP	Presión hidrostática MSP en hPa
B8	Pre LSP	Presión hidrostática LSP en hPa
B9	Vel Cor	Velocidad de corriente en cm/s
BA	Dir Cor	Dirección de la corriente en grados (º)
BB	Tem Mar	Temperatura del agua del mar en ºC
BC	Tem Ter	Temperatura termistor en ºC
BD	Ola Sig2	Altura significante de ola (variante) en metros
72	Rad.Refl	Radiación solar reflejada en w/m²
73	Rad.UV	Radiación UV en J/m² h

Anexo B – Código fuente del generador de snapshots

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

Listing 5: Clase MobilitySnapshotGeneratorService

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

import es.joninx.tfm.dc.builder.dataset.MobilitySnapshotBuilder
import es.joninx.tfm.dc.config.Cfg
import es.joninx.tfm.dc.repository.meteo.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,
windowStartDateTime = intervalStart,
windowEndDateTime = 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)
    }

}
```