Universidad de Valladolid

Escuela de ingeniería Informática

Grado en Ingeniería Informática

TRABAJO DE FIN DE GRADO:

Modelado de la demanda de transporte urbano mediante redes neuronales de grafos (GNNs)

AUTOR:
Roberto Riesco Valdés

TUTOR:
José Vicente Álvarez Bravo

Junio 2025

Índice

Índice de figuras

Parte I - Descripción del proyecto

1. Planteamiento del proyecto

1.1. Contexto y motivación

El presente trabajo se enmarca dentro del campo de la investigación aplicada, con un enfoque eminentemente exploratorio. Su objetivo principal ha sido estudiar el potencial de las Graph Neural Networks (GNNs) para modelar sistemas complejos donde las relaciones entre entidades desempeñan un papel clave. Más concretamente, se ha optado por aplicar esta tecnología al ámbito del transporte urbano, un entorno donde la conectividad, la dinámica temporal y la estructura de red ofrecen un escenario ideal para evaluar la capacidad de representación de las GNNs.

En sus fases iniciales, el proyecto partió sin un conjunto de datos reales disponible, lo que motivó la construcción de un entorno simulado basado en datos sintéticos. Esta aproximación permitió diseñar progresivamente una serie de modelos experimentales (toy models) con los que explorar diferentes aspectos del aprendizaje sobre grafos: desde la codificación temporal de la información, hasta la evolución de la demanda en franjas horarias, pasando por la integración de varias líneas de autobús y su conexión con una estación intermodal.

Este camino experimental no solo sirvió para validar los fundamentos teóricos y técnicos de las GNNs, sino también para afinar los métodos de representación y entrenamiento en grafos dirigidos, en un entorno controlado. Posteriormente, la segunda fase del trabajo consistió en aplicar los conocimientos adquiridos a un caso real, utilizando una base de datos de trayectos de autobuses interestatales de Brasil, lo que supuso un salto en complejidad y volumen de datos, y permitió contrastar la aplicabilidad práctica del enfoque.

Así, el trabajo combina el desarrollo de un marco conceptual sólido con la experimentación práctica progresiva, haciendo del análisis de redes de transporte urbano un caso de estudio representativo para evaluar el comportamiento de las Graph Neural Networks. La motivación última del proyecto ha sido doble: por un lado, investigar en profundidad una tecnología emergente dentro del aprendizaje automático; por otro, contribuir a la búsqueda de soluciones inteligentes y adaptativas para problemas reales de movilidad urbana.

1.2. Objetivos del trabajo

El objetivo principal de este Trabajo de Fin de Grado es diseñar, implementar y analizar un sistema basado en Graph Neural Networks (GNNs) que permita modelar y evaluar el comportamiento de una red de transporte compuesta por dos líneas de autobús conectadas a una estación de tren. A partir de una serie de experimentos controlados y representaciones del grafo en distintas franjas horarias, se busca estudiar la influencia de factores temporales y topológicos en el rendimiento del modelo.

De forma más específica, los objetivos concretos del proyecto son los siguientes:

- Estudiar los fundamentos teóricos de las GNNs y su aplicación al modelado de datos estructurados en forma de grafo.
- Diseñar una representación de red de transporte que refleje relaciones espaciales y temporales entre paradas, líneas y conexiones.

- Implementar distintos modelos experimentales o *toy models*, que integren variantes en la estructura del grafo y en la codificación del tiempo.
- Evaluar el comportamiento de los modelos en diferentes franjas horarias, analizando métricas como la precisión, la pérdida y la distribución de activaciones.
- Desarrollar herramientas de visualización que faciliten la interpretación de los resultados, tanto a nivel numérico como gráfico.
- Aplicar los conocimientos teórico-prácticos obtenidos a la resolución de un problema real relacionado con el transporte urbano.
- Reflexionar sobre la utilidad de este enfoque en contextos reales de movilidad urbana y proponer posibles líneas de mejora o expansión futura
- Establecer una base teórico-práctica que sirva de referencia a futuros trabajos en este ámbito

1.3. Metodología general

El desarrollo del presente trabajo se ha planteado siguiendo una metodología iterativa y experimental, centrada en la construcción progresiva de una serie de modelos simplificados que permiten explorar distintos aspectos del uso de Graph Neural Networks (GNNs) en el análisis de redes de transporte urbano.

En una primera fase, se definió la estructura básica del grafo a partir de un conjunto de paradas y conexiones representativas de dos líneas de autobús y su vinculación con una estación de tren. Esta estructura se modeló con herramientas como NetworkX y PyTorch Geometric, permitiendo una manipulación flexible de los nodos, aristas y sus atributos.

A continuación, se desarrollaron distintas versiones del modelo (denominadas *toy-model-1*, *toy-model-2*, etc.), incorporando progresivamente elementos como múltiples franjas horarias, codificación temporal seno-coseno, o visualización consolidada de los resultados en un único grafo. Cada versión fue evaluada de forma independiente, con el objetivo de identificar qué configuraciones ofrecían un comportamiento más robusto o interpretable.

El entrenamiento y la evaluación de los modelos se realizaron sobre datos sintéticos organizados por franjas horarias, lo que permitió estudiar el comportamiento del sistema en diferentes contextos temporales. Para ello, se definieron métricas cuantitativas (como la función de pérdida y la precisión) y representaciones gráficas (como mapas de calor y grafos coloreados) que facilitaron la interpretación de los resultados.

La última fase del proyecto sufrió algunas modificaciones,, puesto que los datos sobre las líneas de autobús no fueron cedidos para su análisis, por lo que se buscó una base de datos externa, concretamente Autobuses de Brasil para realizar algunos análisis predictivos, teniendo en cuenta lo aprendido durante los modelos anteriores.

Finalmente, se documentaron las conclusiones obtenidas en cada iteración, así como las limitaciones encontradas y las posibles mejoras para trabajos futuros, manteniendo siempre una orientación práctica y aplicable al contexto de la movilidad urbana.

1.4. Formato de la memoria

La presente memoria se ha estructurado en tres partes diferenciadas para facilitar la comprensión del proyecto, desde su motivación teórica hasta la implementación práctica y los resultados obtenidos.

En la Parte I se describe el planteamiento del proyecto, incluyendo el contexto general, los objetivos que se persiguen, la metodología de trabajo empleada y los fundamentos teóricos necesarios para entender el uso de Graph Neural Networks (GNNs) en el ámbito del modelado de redes de transporte.

La Parte II recoge el desarrollo del sistema, desde la definición de la estructura del grafo y el tratamiento de los datos horarios hasta la implementación técnica de los modelos, las herramientas utilizadas, las visualizaciones generadas y el análisis de resultados. Esta parte constituye el núcleo del trabajo y refleja el enfoque experimental adoptado.

Por último, la Parte III incluye los apéndices, en los que se detallan los pasos para instalar y ejecutar el entorno de trabajo, así como las instrucciones necesarias para reproducir los experimentos realizados y visualizar los resultados.

Con esta organización se pretende ofrecer al lector una visión clara, progresiva y reproducible del proyecto desarrollado.

El código utilizado para obtener los resultados aquí expuestos puede encontrarse en el siguiente repositorio:

```
git clone https://github.com/ruperame/gnntoymodel
```

2. Planificación

2.1. Fases de desarrollo

El desarrollo del proyecto se ha dividido en varias fases claramente delimitadas, siguiendo una estrategia incremental que ha permitido validar cada componente del sistema antes de avanzar al siguiente nivel de complejidad. Esta aproximación ha resultado especialmente útil dado el carácter experimental del trabajo y la necesidad de probar distintos enfoques de representación y entrenamiento.

1. **Análisis preliminar y revisión bibliográfica:** en esta etapa se recopilaron referencias clave sobre Graph Neural Networks, representación de grafos y aplicaciones en el ámbito de la movilidad urbana. También se definieron los objetivos generales y el alcance del proyecto.

2. **Diseño del modelo de grafo:** se definió una estructura inicial de nodos y aristas que representara de forma abstracta dos líneas de autobús y su conexión con una estación de tren. Se identificaron los atributos necesarios y se establecieron las reglas para la creación de grafos por franja horaria.

3. **Construcción de modelos base (*toy-models*):** se desarrollaron distintas versiones del modelo para estudiar de manera controlada aspectos como la codificación temporal, la conectividad del grafo o la inclusión de atributos. Cada modelo fue probado de forma independiente.

4. **Aplicación a un problema real:** se buscó una alternativa a los datos de autobuses encontrando una base de datos dsitponible, que se usó para intentar reflejar dos problemas de predicción en los modelos GNN/GCN.

5. **Visualización y análisis de resultados:** se generaron gráficos, mapas de calor y representaciones del grafo coloreado para facilitar la interpretación de los resultados obtenidos en cada versión y franja horaria.

6. **Documentación y conclusiones:** se organizaron los resultados en esta memoria, incorporando reflexiones sobre el enfoque seguido y posibles mejoras para futuras versiones o trabajos relacionados.

2.2. Planificación temporal

La planificación temporal de este Trabajo de Fin de Grado se ha estructurado en torno a una serie de hitos encadenados, distribuidos a lo largo de varios meses de trabajo desde marzo hasta junio de 2025. Esta planificación ha seguido una metodología iterativa, permitiendo avanzar desde la exploración teórica inicial hasta la implementación de modelos aplicados a datos reales.

Marzo 2025 – Estudio preliminar e ideación del proyecto Durante el mes de marzo se realizó una revisión bibliográfica intensiva sobre Graph Neural Networks (GNNs), aprendizaje sobre grafos y sus aplicaciones en el ámbito del transporte. A partir de esta fase exploratoria se definieron los objetivos del proyecto y se delimitaron los casos de uso simulados, centrados en redes de autobús conectadas con una estación de tren.

Abril 2025 – Desarrollo de toy models A partir de abril se inició el diseño e implementación de los primeros modelos experimentales (*toy-model-1* y *toy-model-2*). Estos modelos sirvieron como entorno controlado para evaluar el funcionamiento básico de una GNN sobre grafos simples. Cada dos semanas se introdujeron mejoras sustanciales, incorporando franjas horarias, codificación temporal (*toy-model-3*) y entrenamiento multifranja (*toy-model-4*), lo que permitió simular escenarios urbanos con mayor fidelidad.

Mayo 2025 – Transición a datos reales y casos aplicados En mayo, al no disponer finalmente de los datos de autobuses de Segovia, se optó por emplear un dataset público de trayectos interestatales de autobuses en Brasil. Esta nueva etapa implicó un esfuerzo adicional en limpieza, codificación y transformación del dataset original. A partir de esta base se desarrollaron dos modelos aplicados: uno orientado a la predicción de enlaces entre paradas, y otro centrado en la predicción de demanda (regresión sobre aristas).

Junio 2025 – Evaluación, mejoras y documentación Finalmente, durante el mes de junio se consolidaron los modelos más eficaces, se evaluaron los resultados obtenidos y se introdujeron mejoras técnicas, especialmente en la representación de nodos y la segmentación por franjas de demanda. Paralelamente, se redactó la memoria del proyecto, documentando tanto el proceso técnico como los aprendizajes obtenidos.

3. Fundamentos teóricos

3.1. Introducción a los grafos

3.1.1 Elementos fundamentales de la teoría de grafos

La teoría de grafos es una rama de las matemáticas discretas que estudia estructuras formadas por objetos conectados entre sí. Estas estructuras, llamadas grafos, son herramientas especialmente útiles para modelar relaciones y flujos entre entidades, motivo por el cual su uso se ha extendido a numerosos campos, incluyendo la informática, las redes sociales, la biología, la logística y, en particular, los sistemas de transporte.

Formalmente, un grafo G se define como un par $G = (V, E)$, donde:

- V es un conjunto finito de *nodos* o *vértices*, que representan las entidades individuales del sistema.
- $E \subseteq V \times V$ es un conjunto de *aristas* o *enlaces*, que representan las conexiones entre pares de nodos.

En función de la dirección de las aristas, los grafos pueden clasificarse en:

- **Grafos dirigidos**, en los que cada arista tiene un sentido definido, es decir, una relación de origen y destino entre nodos.
- **Grafos no dirigidos**, en los que las aristas representan relaciones bidireccionales o simétricas.

Además de su estructura básica, los grafos pueden enriquecerse con información adicional asociada a sus elementos. Esta información se incorpora mediante **atributos**:

- *Atributos de nodos*: permiten caracterizar cada nodo con propiedades específicas, como una etiqueta, una coordenada geográfica, un valor numérico, etc.
- *Atributos de aristas*: pueden representar, por ejemplo, la distancia, el tiempo de viaje o la frecuencia de conexión entre dos nodos.

Otro concepto relevante es el de *adyacencia*, que describe la relación inmediata entre nodos. Dos nodos u y v se consideran adyacentes si existe una arista que los conecta. Esta relación puede representarse mediante una **matriz de adyacencia** A, donde $A_{ij} = 1$ si existe una arista entre el nodo i y el nodo j, y 0 en caso contrario. En grafos ponderados, esta matriz puede contener los valores asociados a las aristas (por ejemplo, pesos o distancias).

En aplicaciones prácticas como la modelización de redes de transporte, los nodos pueden representar paradas, estaciones o puntos de interés, mientras que las aristas pueden reflejar trayectos directos entre ellos. Este enfoque permite capturar no solo la infraestructura del sistema, sino también propiedades dinámicas como horarios, frecuencias o niveles de congestión.

Una característica importante de los grafos en este tipo de sistemas es su naturaleza dinámica o temporal. A menudo, la estructura del grafo cambia a lo largo del tiempo debido a la disponibilidad de rutas, el comportamiento de los usuarios o la variación en los servicios ofrecidos. Esta dimensión temporal se puede incorporar mediante grafos dinámicos o mediante el uso de múltiples instancias del grafo asociadas a distintas franjas horarias, como se hace en este trabajo.

En resumen, los grafos proporcionan un marco versátil para representar y analizar sistemas complejos de forma estructurada. Su simplicidad formal, combinada con su capacidad para modelar relaciones ricas y variadas, los convierte en una elección natural para problemas donde las conexiones entre entidades son tan importantes como las entidades mismas.

3.1.2 Aplicación de la teoría de grafos a redes de transporte

La representación de redes de transporte mediante grafos es una de las aplicaciones más consolidadas y eficaces de la teoría de grafos en el mundo real. Este enfoque permite modelar infraestructuras y servicios de transporte de forma estructurada, facilitando el análisis de conectividad, optimización de rutas, estimación de tiempos de viaje, detección de cuellos de botella y otras tareas clave para la gestión y planificación urbana.

En este contexto, los nodos del grafo suelen representar elementos estáticos del sistema como paradas de autobús, estaciones de tren, cruces o puntos de transferencia. Las aristas, por su parte, representan trayectos posibles entre dichos nodos. Estas aristas pueden ser dirigidas o no, dependiendo de si el sistema admite trayectos en ambos sentidos (bidireccional) o en uno solo (como en líneas de autobús de sentido único).

Una característica importante es que las aristas pueden estar enriquecidas con información adicional, como el tiempo estimado de viaje, la frecuencia del servicio, el coste, o incluso variables dinámicas como la densidad de pasajeros en cada tramo. Este tipo de información es esencial para realizar simulaciones realistas o para alimentar modelos predictivos como los basados en aprendizaje automático.

El uso de grafos en redes de transporte también permite incorporar la dimensión temporal mediante distintas estrategias. Una de las más habituales consiste en generar grafos diferentes para franjas horarias concretas, de modo que se puedan comparar comportamientos en distintos momentos del día. Este enfoque es precisamente el adoptado en el presente trabajo, donde se han definido varios grafos para las horas 10:00, 12:00, 16:00 y 20:00, permitiendo así analizar cómo varía la conectividad y la actividad del sistema en función del momento.

En el caso concreto modelado en este proyecto, se ha representado una red de transporte compuesta por dos líneas de autobús conectadas con una estación de tren. Las paradas de autobús se han representado como nodos, mientras que los trayectos directos entre ellas se han modelado como aristas. La estación de tren actúa como un nodo de conexión entre ambas líneas, cumpliendo una función clave en la topología del grafo.

Esta representación ha permitido construir grafos coherentes en los que se puede analizar la conectividad de los diferentes puntos de la red, así como evaluar el efecto de la hora del día sobre dicha estructura. Además, gracias al uso de atributos en los nodos (como el tipo de parada o la línea a la que pertenecen) y en las aristas (como la dirección del trayecto), se ha logrado capturar la semántica necesaria para alimentar modelos basados en Graph Neural Networks (GNNs).

Cabe destacar que este tipo de representación es altamente extensible y puede adaptarse fácilmente a casos reales más complejos, incluyendo variables como la demanda, el estado del tráfico o el comportamiento de los usuarios. Por ello, la combinación de teoría de grafos y modelos de aprendizaje profundo orientados a grafos constituye una herramienta poderosa para la mejora de los sistemas de transporte y la movilidad urbana sostenible.

3.2. Aprendizaje sobre grafos

3.2.1 Fundamentos del aprendizaje sobre grafos

El aprendizaje sobre grafos, o Graph Representation Learning (GRL), es un campo del aprendizaje automático que tiene como objetivo extraer representaciones vectoriales útiles de los elementos de un grafo —principalmente nodos, aristas y el propio grafo completo— que conserven la estructura y las propiedades relevantes de dicho grafo. Esta capacidad de convertir una estructura relacional y compleja en vectores numéricos manejables abre la puerta a tareas de predicción, clasificación,

agrupamiento, búsqueda de similitudes, detección de anomalías y muchas más. [3]

El reto principal del aprendizaje sobre grafos es que, a diferencia de los datos tabulares o imágenes que tienen una estructura regular, los grafos son inherentemente no euclidianos, es decir, no pueden representarse como matrices fijas sin perder información topológica esencial. Por esta razón, no es viable aplicar modelos convencionales directamente sobre ellos sin una transformación previa que respete su conectividad.

El aprendizaje sobre grafos busca entonces mapear la información estructural y semántica de un grafo a un espacio latente continuo. Es decir, asignar a cada nodo (y, en algunos casos, a cada arista o al grafo completo) un vector de características, también conocido como *embedding*, que capture tanto las propiedades locales del nodo como su contexto estructural en el grafo.

Este proceso de aprendizaje puede abordarse de diferentes formas dependiendo de la tarea:

- **Aprendizaje supervisado sobre nodos:** el objetivo es predecir etiquetas para nodos individuales (por ejemplo, si una parada pertenece a una línea específica). Se requiere un conjunto de nodos con etiquetas conocidas para entrenar el modelo.
- **Aprendizaje no supervisado:** en este caso se busca aprender embeddings útiles sin etiquetas, por ejemplo, para detectar comunidades o realizar agrupamientos en la red.
- **Clasificación de grafos:** se predice una etiqueta para un grafo completo. Esta tarea es común en química (por ejemplo, clasificar moléculas como tóxicas o no tóxicas), y también podría aplicarse a grafos horarios si se quisiera, por ejemplo, etiquetar franjas como "alta demanda" o "baja demanda".
- **Predicción de enlaces:** tiene como objetivo inferir qué conexiones podrían existir en el futuro, o cuáles faltan. En el contexto del transporte, esto podría servir para sugerir conexiones potenciales o detectar interrupciones.

Una de las claves del éxito del aprendizaje sobre grafos es su capacidad de explotar las relaciones entre elementos que no están explícitamente representadas por atributos. Por ejemplo, dos paradas de autobús podrían parecer similares en cuanto a su frecuencia de paso o número de usuarios, pero si se sitúan en regiones de la red con topologías distintas (una en un nodo intermedio y otra en un extremo), su comportamiento dentro del sistema puede ser completamente distinto. El aprendizaje sobre grafos permite capturar ese tipo de diferencia estructural, algo que sería mucho más difícil con enfoques tradicionales basados únicamente en datos tabulares.

Desde un punto de vista matemático, los métodos de GRL suelen combinar información sobre las conexiones del nodo (topología) con la información contextual de sus vecinos. Así, el embedding de un nodo se construye como una función de sus propios atributos y de los atributos de los nodos con los que se conecta. Esta idea es la base de las técnicas modernas como las Graph Neural Networks, que serán abordadas más adelante.

En el caso de este proyecto, el aprendizaje sobre grafos se utiliza para construir modelos que puedan reconocer patrones de conectividad en redes de transporte simuladas. Los nodos (paradas) y aristas (conexiones) se enriquecen con información temporal, y el objetivo es evaluar cómo varía el comportamiento del modelo en función de distintas estructuras y horarios. Esta aplicación práctica ilustra cómo el aprendizaje sobre grafos permite generar conocimiento a partir de datos complejos, relacionales y temporales.

3.2.2 Técnicas y modelos representativos

Desde los primeros enfoques basados en heurísticas hasta los modelos más avanzados que emplean redes neuronales profundas, el aprendizaje sobre grafos ha evolucionado rápidamente, incorporando técnicas cada vez más potentes para capturar tanto la estructura como la semántica de los grafos. A continuación, se presentan algunas de las metodologías más representativas, clasificadas en tres grandes grupos: técnicas basadas en grafos clásicos, modelos de representación mediante aprendizaje profundo, y Graph Neural Networks (GNNs), estas últimas como base central del presente trabajo. [4]

1. Métodos clásicos de incrustación (embedding): Antes del auge del aprendizaje profundo, los métodos más habituales para representar nodos en un grafo eran algoritmos de tipo *shallow embedding*, que construyen una matriz de vectores de forma estática, sin capacidad de generalización a nuevos nodos. Entre los más conocidos destacan:

- **DeepWalk:** inspirado en el modelo Word2Vec, utiliza recorridos aleatorios (random walks) sobre el grafo para generar secuencias de nodos, tratando cada nodo como una palabra y cada recorrido como una frase. Luego entrena un modelo de skip-gram para aprender los embeddings.
- **Node2Vec:** extiende DeepWalk incorporando un sesgo controlado en los recorridos aleatorios, lo que permite ajustar el enfoque hacia exploraciones más amplias (*breadth-first*) o más profundas (*depth-first*), capturando distintos tipos de similitud entre nodos.
- **LINE (Large-scale Information Network Embedding):** se centra en preservar explícitamente tanto las proximidades de primer orden (adyacencia directa) como de segundo orden (similitud de vecinos).

Aunque estos métodos son computacionalmente eficientes y han mostrado buenos resultados en tareas como la detección de comunidades o la predicción de enlaces, tienen limitaciones importantes: no incorporan atributos de nodos ni aristas, y no pueden adaptarse a cambios dinámicos en el grafo sin volver a entrenar desde cero.

2. Modelos basados en aprendizaje profundo: Con la aparición de frameworks como TensorFlow y PyTorch, se comenzaron a desarrollar arquitecturas más flexibles y potentes para operar sobre datos estructurados. El paso clave fue la incorporación del concepto de *message passing*, que consiste en que cada nodo actualice su representación a partir de información agregada de sus vecinos.

Este enfoque sentó las bases para las Graph Neural Networks, y permitió la aparición de modelos que sí integran tanto la topología del grafo como los atributos de sus nodos o aristas. Entre los más representativos destacan:

- **GCN (Graph Convolutional Network):** propone una generalización de las convoluciones clásicas (como en redes CNN) al dominio de los grafos. A cada capa, el vector de cada nodo se actualiza combinando su propio estado con una media ponderada de sus vecinos.
- **GAT (Graph Attention Network):** introduce mecanismos de atención para que cada nodo pueda asignar pesos diferentes a sus vecinos en función de su relevancia, en lugar de asumir un promedio uniforme como en GCN. Este modelo es especialmente útil cuando los vecinos no son todos igual de informativos.

- **GraphSAGE:** en lugar de usar la matriz completa de adyacencia, GraphSAGE emplea una estrategia inductiva basada en muestreo y agregación, lo que le permite generalizar a nodos no vistos durante el entrenamiento y escalar mejor a grafos grandes.

Todos estos modelos tienen en común la estructura en capas, donde cada nodo agrega y transforma información recibida de sus vecinos. Cuantas más capas se apilan, más lejos puede llegar esta agregación en el grafo, aunque existe un riesgo de sobre-saturación o "sobresuavizado" si se propaga demasiado.

3. Aplicación de GNNs en sistemas reales: Las GNNs han demostrado su eficacia en tareas complejas como la predicción de tráfico, la recomendación de rutas, la detección de fraudes o el análisis de redes sociales. En el ámbito del transporte urbano, su capacidad para integrar la estructura de la red, los atributos de cada nodo (como tipo de parada, posición, línea, etc.) y la variabilidad temporal las convierte en una solución especialmente potente.

En el presente trabajo se emplean modelos basados en GCN como primera aproximación, aplicados a una red simulada de líneas de autobús conectadas a una estación de tren. A través del análisis por franjas horarias y de la evolución de las representaciones de los nodos, se evalúa la capacidad del modelo para capturar patrones útiles relacionados con la estructura del sistema y su comportamiento a lo largo del tiempo.

Este enfoque constituye una base sólida para futuras extensiones con datos reales, arquitecturas más avanzadas como GAT o GraphSAGE, y tareas más ambiciosas como predicción de demanda, detección de incidencias o recomendación de rediseños estructurales.

3.3. Redes Neuronales con Grafos (GNNs)

3.3.1 Fundamentos y arquitectura general de las GNNs

Las Graph Neural Networks (GNNs) representan una extensión natural de las redes neuronales tradicionales al dominio de estructuras no euclidianas, como los grafos. Su principal objetivo es aprender representaciones de nodos, aristas o grafos completos capturando tanto los atributos de cada elemento como la información estructural de su entorno en la red.

El funcionamiento de una GNN se basa en un principio fundamental: la información relevante para una entidad en un grafo no solo reside en sus características individuales, sino también en la estructura del grafo que la rodea. Este principio se materializa mediante un proceso iterativo conocido como **message passing** o **propagación de mensajes**, que se aplica capa a capa en la red.

A grandes rasgos, cada capa de una GNN ejecuta tres operaciones principales:

1. **Agregación:** cada nodo recopila información de sus vecinos. Este paso puede implicar calcular una media, suma, máximo o una combinación más sofisticada (por ejemplo, usando atención).

2. **Actualización:** el nodo actualiza su representación combinando su propio estado anterior con la información agregada de los vecinos. Esta operación suele implicar una transformación lineal seguida de una función de activación no lineal.

3. **Propagación:** la nueva representación se propaga a la siguiente capa, donde el proceso vuelve a repetirse.

Formalmente, si $h_v^{(k)}$ representa el embedding del nodo v en la capa k, y $\mathcal{N}(v)$ es el conjunto de vecinos de v, el paso de actualización puede expresarse como:

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot \mathrm{AGG}^{(k)}\left(\{h_u^{(k-1)} : u \in \mathcal{N}(v) \cup \{v\}\}\right)\right)$$

donde:

- AGG es una función de agregación (media, suma, concatenación, atención, etc.).
- $W^{(k)}$ son los pesos aprendidos en la capa k.
- σ es una función de activación (ReLU, tanh, etc.).

Este mecanismo permite a cada nodo "recibir información" de sus vecinos directos en la primera capa, de los vecinos de sus vecinos en la segunda, y así sucesivamente. De esta forma, el modelo puede capturar patrones estructurales complejos sin necesidad de diseñar manualmente características derivadas.

Una ventaja fundamental de las GNNs frente a otros enfoques es su capacidad de generalizar a grafos con distinto tamaño y forma. A diferencia de las redes convolucionales convencionales, que requieren entradas con dimensiones fijas, las GNNs operan sobre estructuras arbitrarias, adaptándose dinámicamente a cada instancia del grafo.

En el presente trabajo, esta arquitectura se emplea para modelar redes de transporte, donde los nodos representan paradas y las aristas los trayectos entre ellas. Cada nodo codifica información relevante (por ejemplo, su papel en la red o su franja horaria), y a través de la propagación de mensajes se espera que el modelo pueda aprender patrones globales a partir de interacciones locales.

3.3.2 Variantes de GNNs y su aplicabilidad

A lo largo de los últimos años han surgido múltiples variantes de Graph Neural Networks, cada una diseñada para superar ciertas limitaciones de los modelos originales o para adaptarse mejor a tareas concretas. Estas variantes difieren principalmente en los mecanismos de agregación y atención, en la forma de procesar la información topológica, o en su capacidad para operar sobre grafos dinámicos y heterogéneos.

Entre las variantes más representativas se encuentran:

- **GCN (Graph Convolutional Network):** es una de las arquitecturas más simples y ampliamente utilizadas. Realiza una agregación normalizada de los vecinos mediante la matriz de adyacencia y una transformación lineal. Su simplicidad permite un entrenamiento eficiente y resultados competitivos en muchas tareas de clasificación de nodos.
- **GraphSAGE (Sample and Aggregation):** propone una estrategia inductiva, en la que cada nodo agrega información de un subconjunto de vecinos mediante funciones como media, LSTM o pooling. Este enfoque es útil cuando el grafo es muy grande o cambia frecuentemente, como ocurre en sistemas de transporte con eventos dinámicos.
- **GAT (Graph Attention Network):** introduce mecanismos de atención que permiten a cada nodo ponderar de forma diferenciada la información que recibe de sus vecinos. Esto es especialmente útil en redes donde algunas conexiones son más relevantes que otras, como puede ocurrir entre paradas de transporte con mayor flujo de pasajeros o nodos intermodales clave.

- **GIN (Graph Isomorphism Network):** enfocado en la discriminación entre estructuras de grafos, GIN tiene mayor capacidad para distinguir grafos estructuralmente distintos y se ha mostrado competitivo en tareas de clasificación de grafos.
- **Dynamic GNNs:** son arquitecturas capaces de adaptarse a cambios en el grafo a lo largo del tiempo. Incorporan mecanismos temporales (como memoria o codificadores recurrentes) y se emplean en contextos donde la topología cambia, como redes de transporte, redes sociales o sensores IoT.

La elección de la arquitectura adecuada depende en gran medida del problema a resolver, la naturaleza del grafo, y los recursos computacionales disponibles. En sistemas de transporte, donde la conectividad puede variar según el horario, la demanda o los eventos externos, modelos como GAT o GraphSAGE resultan especialmente prometedores por su capacidad de adaptación y generalización.

En este trabajo, se ha optado por un enfoque inicial con GCN, dada su simplicidad y robustez. No obstante, la arquitectura del sistema está diseñada de forma modular para permitir la futura incorporación de variantes más complejas, como GAT, especialmente si se integran datos reales o se busca extender el sistema a tareas más sofisticadas como predicción de demanda o evaluación de accesibilidad.

En definitiva, el campo de las GNNs continúa evolucionando rápidamente, y sus aplicaciones en el ámbito del transporte inteligente constituyen una línea de investigación activa con gran potencial para transformar la manera en que entendemos, planificamos y optimizamos la movilidad urbana.

3.4. Casos de uso y estado del arte

La creciente disponibilidad de datos estructurados y semiestructurados en ámbitos como el transporte, la logística o las telecomunicaciones ha generado un interés creciente por el uso de Graph Neural Networks (GNNs) como herramienta analítica de alto nivel. En los últimos años, se han publicado numerosos trabajos que demuestran la eficacia de estas redes en problemas donde las relaciones entre entidades son tan importantes como las entidades mismas. A continuación se presenta una revisión de algunos casos de uso relevantes, especialmente centrados en el dominio de la movilidad urbana, el transporte público y los sistemas inteligentes de transporte (ITS). [1]

Una de las aplicaciones más consolidadas de las GNNs es la **predicción del tráfico**. En este tipo de trabajos, las intersecciones viales o estaciones de transporte se modelan como nodos, y las carreteras o rutas de transporte como aristas. Modelos como DCRNN (Diffusion Convolutional Recurrent Neural Network) o ST-GCN (Spatio-Temporal Graph Convolutional Network) han sido utilizados con éxito para predecir la intensidad del tráfico en intervalos futuros, combinando grafos espaciales con dinámicas temporales. Estas soluciones se utilizan ya en contextos reales por parte de ayuntamientos y operadores de movilidad.

Otro caso de uso extendido es la **recomendación de rutas y planificación multimodal**. Aquí, los nodos representan modos de transporte (autobús, tren, bicicleta, etc.), paradas o estaciones, y las aristas representan transbordos posibles. Las GNNs pueden aprender patrones de uso, tiempos de conexión, o preferencias de los usuarios para ofrecer rutas óptimas o anticiparse a la saturación de ciertos tramos. Este enfoque resulta especialmente útil en entornos urbanos complejos o cuando se busca fomentar la intermodalidad. [2]

En el ámbito del **transporte público**, las GNNs se han empleado para detectar patrones de demanda, identificar cuellos de botella en la red o predecir incidencias. Algunos trabajos recientes han utilizado representaciones de grafos horarios para analizar cómo varía la estructura de conectividad

a lo largo del día, de forma muy similar al enfoque adoptado en este trabajo. Esto permite detectar franjas críticas, evaluar la resiliencia del sistema o realizar simulaciones de rediseños operativos.

Un ejemplo especialmente cercano a este proyecto lo encontramos en estudios que analizan la **movilidad urbana basada en datos de sensores o vehículos conectados**. En este contexto, los datos de geolocalización (GPS), conteo de pasajeros o estado de las vías pueden integrarse en grafos dinámicos. Las GNNs pueden entonces aprender no solo patrones espaciales sino también temporales, contribuyendo a construir sistemas de alerta temprana, mapas de accesibilidad o herramientas de planificación estratégica.

Además de estas aplicaciones prácticas, existe un número creciente de trabajos académicos que abordan cuestiones más fundamentales sobre el uso de GNNs: cómo evitar el sobresuavizado en arquitecturas profundas, cómo escalar a grafos de gran tamaño, o cómo interpretar las predicciones de las redes. Esto ha dado lugar a bibliotecas especializadas como PyTorch Geometric o DGL (Deep Graph Library), que han facilitado la aplicación de estas técnicas en entornos reales, permitiendo a equipos de investigación y empresas prototipar soluciones con rapidez.

Cabe señalar también que algunas ciudades y empresas del sector transporte ya están integrando estas tecnologías en sus plataformas. Por ejemplo, iniciativas de ciudades inteligentes han explorado el uso de GNNs para evaluar la accesibilidad en diferentes barrios, simular escenarios de rediseño de líneas o optimizar la distribución de recursos. La versatilidad de los grafos permite integrar datos de múltiples fuentes —vehículos, sensores, usuarios— en un mismo modelo, lo que resulta clave en entornos cada vez más complejos y conectados.

De todo lo anterior se concluye que las Graph Neural Networks no solo son una herramienta prometedora en contextos académicos, sino que ya están siendo aplicadas en soluciones prácticas que impactan directamente en la planificación urbana, la eficiencia del transporte público y la experiencia del usuario. El presente trabajo se sitúa en esta línea de investigación aplicada, explorando desde un enfoque didáctico y modular cómo representar y modelar redes de transporte mediante GNNs, con vistas a futuras extensiones hacia contextos reales.

Parte II - Desarrollo del proyecto

4. Descripción del sistema

4.1. Definición del problema y enfoque

En el marco del creciente interés por la mejora del transporte urbano, tanto desde el punto de vista de la sostenibilidad como de la eficiencia operativa, se hace necesario el desarrollo de modelos que permitan representar y analizar redes de movilidad de forma flexible, dinámica y realista. La gestión adecuada de redes de autobús, tren o transporte intermodal requiere herramientas que vayan más allá de los enfoques clásicos basados en optimización estática o reglas fijas. En este contexto, el aprendizaje automático —y en particular, las Graph Neural Networks (GNNs)— ofrece nuevas posibilidades para representar, comprender y anticipar el comportamiento de sistemas de transporte complejos.

Este Trabajo de Fin de Grado se plantea como una aproximación exploratoria al uso de GNNs en el modelado de redes de transporte. A diferencia de otros trabajos que se basan directamente en datos reales, el enfoque adoptado aquí se fundamenta en la construcción progresiva de un entorno de simulación controlado. El sistema se define como un conjunto de paradas pertenecientes a dos líneas de autobús conectadas a una estación de tren central. Esta estructura modular permite representar una red con nodos bien definidos, relaciones topológicas claras y atributos temporales que varían a lo largo del día. El objetivo es observar cómo distintas representaciones gráficas, franjas horarias y estrategias de codificación afectan al comportamiento del modelo.

La hipótesis de partida es que las GNNs pueden ser una herramienta eficaz para extraer representaciones de las paradas (nodos) que reflejen tanto su posición estructural dentro de la red como la información contextual (como la franja horaria o el tipo de conexión). Estas representaciones podrían, en trabajos futuros, alimentar sistemas de predicción de demanda, simulación de flujos o planificación operativa. Sin embargo, en esta fase, el foco se centra en validar que el modelo es capaz de capturar la estructura de la red y generar resultados coherentes al modificar sus componentes clave.

Para ello, se ha optado por dividir el trabajo en una serie de versiones incrementales del modelo, denominadas *toy models*, en los que se introducen elementos como:

- La creación de grafos diferenciados por franja horaria (por ejemplo, 10:00, 12:00, 16:00 y 20:00), permitiendo analizar cómo se comporta la red en distintos momentos del día.
- La incorporación de una codificación temporal seno-coseno, que convierte la hora del día en un vector continuo que puede ser procesado por la GNN.
- El diseño de visualizaciones que ayuden a interpretar tanto la estructura del grafo como la evolución de los valores generados por el modelo.

Este enfoque modular facilita la validación progresiva del sistema, permitiendo detectar errores conceptuales o de implementación en etapas tempranas. Además, permite realizar comparaciones entre distintas configuraciones y plantear hipótesis sobre cómo la estructura del grafo (por ejemplo, la densidad de conexiones o la posición relativa de la estación de tren) influye en las representaciones obtenidas.

Aunque el modelo desarrollado no se entrena sobre datos reales de movilidad urbana, su diseño se inspira directamente en situaciones comunes del transporte público: líneas con horarios diferentes,

puntos de conexión intermodal y comportamientos que cambian a lo largo del día. Esta estrategia de simulación permite generar un entorno controlado donde probar ideas que posteriormente se trasladan a un sistema real.

El presente trabajo define un problema de modelado estructural y temporal de una red de transporte urbana simplificada, abordado mediante Graph Neural Networks. El enfoque seguido, basado en el desarrollo iterativo de modelos y en la separación por franjas horarias, busca sentar las bases para futuras aplicaciones en el análisis y optimización de sistemas de movilidad urbana inteligentes.

4.2. Estructura general del grafo

El sistema desarrollado se basa en una representación gráfica simplificada de dos líneas reales de autobús de la ciudad de Segovia: la Línea 11 y la Línea 12, ambas conectadas a la estación de tren de alta velocidad (Estación AVE). Esta red de transporte se modela mediante grafos dirigidos, donde cada nodo representa una parada concreta y cada arista una conexión directa entre paradas, con pesos que indican el tiempo estimado de recorrido entre ellas.

Ambas líneas se estructuran con cuatro nodos: tres correspondientes a paradas intermedias y uno que representa la Estación AVE, utilizada como nodo destino común. En la Línea 11, el trayecto parte de "Acueducto 3" y pasa por "Plaza de Toros" y "Gerardo Diego" antes de llegar a la estación. En la Línea 12, el recorrido comienza en "Centro" y atraviesa "Instituto Andrés Laguna" y "Carretera San Rafael" hasta confluir también en la estación.

Cada nodo contiene un vector de atributos con varios componentes que evolucionan o se añaden según el toy-model, como por ejemplo:

- Número de pasajeros que suben.
- Número de pasajeros que bajan.
- Identificador numérico de la franja horaria (0.0 para 10:00, 1.0 para 12:00, etc.).
- Codificación seno/coseno para las horas

Además, las aristas tienen atributos que indican el tiempo de desplazamiento (en minutos) entre cada par de paradas, lo que permite incorporar información temporal directamente en la estructura del grafo.

Los grafos se generan para distintas franjas horarias (`10:00`, `12:00`, `16:00` y `20:00`), y tanto la distribución de la demanda como los valores de entrada cambian en función de la hora simulada. Esto permite analizar cómo varía el comportamiento del modelo ante distintas condiciones de carga.

A nivel de implementación, los grafos se construyen utilizando la clase `Data` de `PyTorch Geometric`, que encapsula la información de nodos (`x`), conexiones (`edge_index`), atributos de las aristas (`edge_attr`) y etiqueta objetivo (`y`). Para cada franja horaria, se genera un grafo por línea, y se entrena un modelo de tipo `GCN` para predecir el número de pasajeros que llegarán a la Estación AVE.

Finalmente, se implementa una visualización combinada de ambos grafos mediante la librería `NetworkX`, en la que se colorean y etiquetan los nodos y aristas de cada línea por separado. Esto permite representar visualmente la estructura del sistema, identificar posibles cuellos de botella y comparar recorridos. Las aristas están etiquetadas con su duración en minutos, y la estación aparece resaltada como nodo destino común.

Esta estructura modular y multigrafo facilita el entrenamiento por franjas, la visualización de resultados y la posterior interpretación del comportamiento del modelo en función de la hora y la topología de la red.

4.3. Datos y franjas horarias

El sistema modelado en este proyecto se basa en la red real de autobuses urbanos de Segovia, tomando como referencia las líneas 11 y 12 que conectan distintas zonas del centro urbano con la estación de tren de alta velocidad (Estación AVE). Para construir los grafos que representan estas líneas, se han utilizado tanto las paradas reales como sus coordenadas geográficas exactas, obtenidas a partir de mapas abiertos y fuentes oficiales. Esta decisión permite dotar al sistema de una base espacial coherente, acercando el entorno simulado a un escenario realista, aunque controlado.

Cada parada de autobús se representa como un nodo del grafo, y sus coordenadas GPS se han integrado en el modelo como atributos disponibles para futuras extensiones. Aunque en esta primera fase del trabajo las coordenadas no se utilizan directamente como input de la red neuronal, su incorporación en la estructura del grafo abre la puerta a funcionalidades como la visualización geográfica precisa, la detección de agrupamientos espaciales o la simulación basada en distancia.

Las líneas 11 y 12 se han simplificado a un total de cuatro nodos cada una, manteniendo únicamente los tramos directos entre las paradas más representativas y la Estación AVE. Las conexiones (aristas) se han definido con pesos que indican tiempos estimados de recorrido entre paradas, según los datos oficiales de horarios.

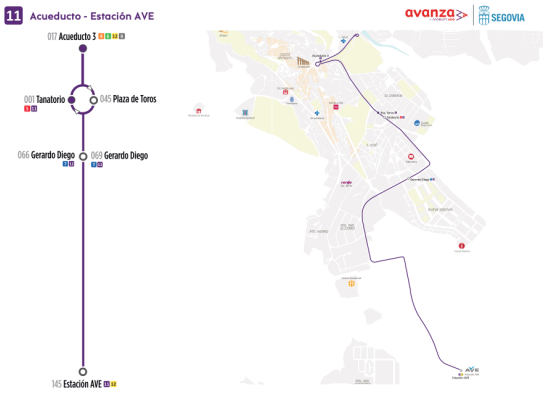
En cuanto a los datos temporales, se han seleccionado cuatro franjas horarias representativas del servicio diario:

- **10:00**: tramo de baja demanda tras el inicio de la jornada.
- **12:00**: franja media antes del mediodía.
- **16:00**: momento de repunte en el servicio vespertino.
- **20:00**: tramo final del día, antes del cierre progresivo.

Estas franjas han sido elegidas de forma manual y se corresponden con horarios reales recogidos en los paneles informativos de las líneas (véanse las Figuras **??** y **??**). No se ha tratado de replicar toda la frecuencia de paso de los autobuses, ya que el objetivo de los toy models no es construir una simulación completa del sistema urbano, sino proporcionar un entorno controlado donde evaluar el comportamiento de modelos de Graph Neural Networks en distintas condiciones.

Para cada franja horaria, se asignan manualmente valores estimados de pasajeros que suben y bajan en cada parada. Estos valores no provienen de datos empíricos, pero se han inspirado en patrones de uso típicos del transporte urbano. Además, la hora correspondiente se codifica como un valor numérico (por ejemplo, 0.0 para 10:00, 1.0 para 12:00) e incorporada como una tercera característica en cada nodo.

En conjunto, esta estructura de datos permite al sistema capturar las variaciones temporales en la red, representar con fidelidad la geografía de las líneas, y analizar el impacto de distintas configuraciones horarias sobre el rendimiento del modelo. La elección de un subconjunto reducido de horarios y paradas facilita además la validación, depuración y comparación entre distintas versiones del modelo.

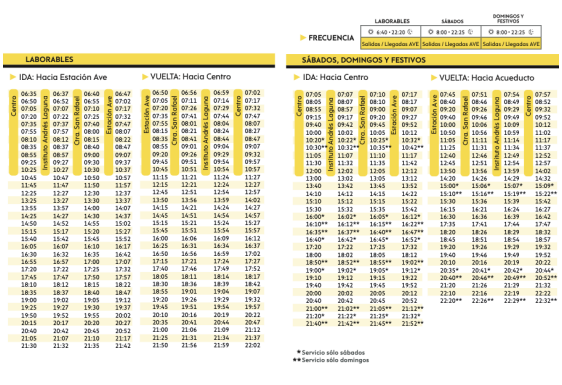


(a) Recorrido de la Línea 11

(b) Horarios de la Línea 11

Figura 1: Información visual de la Línea 11: recorrido y horarios

(a) Recorrido de la Línea 12



(b) Horarios de la Línea 12

Figura 2: Información visual de la Línea 12: recorrido y horarios

5. Implementación técnica

5.1. Herramientas de programación y entorno

El desarrollo del presente proyecto se ha llevado a cabo íntegramente en Python, un lenguaje ampliamente utilizado en el ámbito del aprendizaje automático y el análisis de datos, debido a su sintaxis accesible y su ecosistema rico en librerías científicas. Esta elección ha permitido una integración fluida entre los distintos componentes del proyecto, desde la manipulación de datos hasta la implementación de modelos de Graph Neural Networks (GNNs) y la visualización de resultados.

Para la organización del entorno de trabajo se ha optado por el uso de entornos virtuales a través del módulo estándar de Python `venv`, lo que garantiza la reproducibilidad y el aislamiento de dependencias. El entorno fue creado con el comando:

```
python -m venv gnn-tfg
```

y posteriormente activado para instalar las librerías necesarias mediante `pip`. Todas las dependencias han sido registradas en un archivo `requirements.txt` para facilitar su reinstalación en otros sistemas.

La edición y ejecución del código se realizó principalmente en el editor Visual Studio Code, aprovechando su soporte para notebooks, control de versiones con Git y terminal integrado. Este entorno ofreció además una buena compatibilidad con herramientas gráficas para la depuración y la generación de visualizaciones interactivas.

El control de versiones se realizó mediante Git, con uso de ramas diferenciadas (`toy-model-1`, `toy-model-2`, etc.) para mantener separado el trabajo correspondiente a cada etapa. Este flujo de trabajo permitió probar nuevas funcionalidades sin comprometer el desarrollo consolidado, así como realizar regresiones de forma segura si se detectaban errores.

A nivel de hardware, el proyecto se ejecutó en un entorno local sin necesidad de acceso a GPU. Dado el tamaño reducido de los grafos utilizados y la simplicidad relativa de los modelos, el tiempo de entrenamiento fue aceptable incluso con CPU. No obstante, el código se ha estructurado de forma modular para permitir futuras migraciones a plataformas con soporte de cómputo acelerado si se trabaja con datos reales de mayor escala.

5.2. Librerías para modelado de grafos y GNNs

Uno de los aspectos clave del desarrollo de este proyecto ha sido la selección adecuada de librerías que faciliten el trabajo con grafos y permitan construir modelos de Graph Neural Networks (GNNs) de forma flexible y eficiente. En este sentido, se ha hecho uso de dos herramientas principales: `NetworkX` para la generación y manipulación inicial de grafos, y `PyTorch Geometric` para la implementación y entrenamiento de los modelos neuronales sobre dichos grafos.

NetworkX: generación y exploración estructural `NetworkX` es una librería de Python diseñada específicamente para la creación, análisis y visualización de estructuras de grafos. Ha sido ampliamente utilizada en entornos académicos y permite trabajar con grafos dirigidos y no dirigidos, incorporar atributos a nodos y aristas, calcular métricas topológicas, y exportar/importar grafos en diversos formatos.

En este proyecto, `NetworkX` se empleó principalmente para:

- Crear la topología base de cada toy model, definiendo nodos y sus conexiones.
- Asignar atributos como identificadores, etiquetas, franjas horarias o codificaciones temporales.
- Visualizar de forma rápida la estructura de los grafos y verificar su coherencia antes de pasarlos a un modelo neuronal.

Una vez construidos, los grafos de `NetworkX` se convierten en objetos compatibles con `PyTorch Geometric` mediante funciones utilitarias específicas, lo que permite integrarlos directamente en el flujo de entrenamiento.

PyTorch Geometric: modelado de GNNs `PyTorch Geometric` (PyG) es una extensión del framework PyTorch orientada específicamente al trabajo con datos de grafos. Ofrece estructuras de datos optimizadas (como `Data` y `DataLoader`), una amplia colección de modelos predefinidos (GCN, GAT, GraphSAGE, etc.), y herramientas para tareas como clasificación, predicción de enlaces o regresión sobre nodos o grafos.

Las ventajas de PyG en el contexto de este proyecto son múltiples:

- Permite definir grafos con atributos arbitrarios en nodos y aristas, lo que resulta fundamental en un sistema donde se codifican variables como la franja horaria o el tipo de parada.
- Su API está completamente integrada en PyTorch, lo que facilita la definición de arquitecturas personalizadas, el uso de optimizadores estándar, y la gestión del ciclo de entrenamiento y evaluación.
- Ofrece funciones eficientes de agregación de vecinos, propagación de mensajes y manejo de grafos heterogéneos o dinámicos.

El modelo principal utilizado en este trabajo es un GCN (Graph Convolutional Network) simple, implementado como una clase heredada de `torch.nn.Module`, con dos capas de convolución sobre grafos y una función de activación intermedia. La arquitectura fue diseñada con el objetivo de ser interpretable y fácilmente extensible hacia modelos más complejos si fuera necesario.

Otras librerías de soporte Adicionalmente, se han utilizado otras librerías que, si bien no están orientadas específicamente a grafos, resultan esenciales para el desarrollo completo del sistema:

- `Pandas` y `NumPy` para la gestión de datos tabulares, lectura de CSVs y manipulaciones numéricas.
- `Matplotlib` y `Seaborn` para la generación de gráficos, mapas de calor y visualizaciones de resultados.
- `OS` y `glob` para la gestión de rutas, directorios y lectura secuencial de archivos.

Este conjunto de librerías ha proporcionado una base sólida para cubrir todo el ciclo de trabajo: desde la construcción inicial de grafos hasta el entrenamiento de modelos neuronales y la visualización final de los resultados. Su combinación ha sido fundamental para mantener el código modular, legible y reproducible.

5.3. Codificación seno-coseno de la hora

Una de las características más relevantes del sistema modelado en este trabajo es la variabilidad temporal: la red de transporte no se mantiene constante durante todo el día, sino que presenta diferencias significativas entre franjas horarias como la mañana, el mediodía, la tarde o la noche. Esta dinámica temporal se manifiesta tanto en la estructura del grafo (por ejemplo, en la disponibilidad o frecuencia de ciertas conexiones) como en el comportamiento del sistema (flujo de pasajeros, carga de nodos, tiempos de espera, etc.).

Para incorporar esta dimensión temporal en los modelos de Graph Neural Networks (GNNs), es necesario transformar la información horaria en una representación que pueda ser procesada por una

red neuronal. Sin embargo, utilizar directamente la hora como un número entero (por ejemplo, 10, 12, 16, 20) introduce discontinuidades artificiales y no refleja adecuadamente la naturaleza cíclica del tiempo. Por ejemplo, en una codificación lineal, las 23:00 y las 00:00 estarían tan alejadas como las 00:00 y las 11:00, cuando en realidad son consecutivas en el ciclo diario.

Para resolver este problema, se ha optado por una **codificación circular basada en funciones seno y coseno**, que permite representar las horas del día como puntos en el círculo trigonométrico, preservando su continuidad y relaciones angulares. Esta técnica es ampliamente utilizada en aprendizaje automático para representar variables cíclicas como horas, días de la semana o meses del año.

Fórmulas de codificación La codificación seno-coseno se implementa mediante las siguientes fórmulas, considerando una variable horaria $h \in [0, 24)$:

$$\text{hora\_seno} = \sin\left(2\pi \cdot \frac{h}{24}\right) \qquad \text{hora\_coseno} = \cos\left(2\pi \cdot \frac{h}{24}\right)$$

De esta forma, cada hora se convierte en un par de valores continuos entre -1 y 1 que representan su posición angular en el día. Esta codificación conserva la cercanía entre horas adyacentes y cierra el ciclo al unir correctamente la medianoche con las últimas horas del día.

Implementación y uso en el modelo En este trabajo, la codificación seno-coseno se utiliza como parte de los atributos de los nodos en cada grafo horario. Al generar los grafos correspondientes a las horas seleccionadas (10:00, 12:00, 16:00 y 20:00), cada nodo recibe dos atributos adicionales: `hora_seno` y `hora_coseno`. Estos atributos se concatenan con otras características del nodo (como el tipo de parada o el identificador de línea) y se utilizan como entrada para el modelo GNN.

El cálculo de estos atributos se realiza en el preprocesamiento, utilizando la librería NumPy de la siguiente manera:

```
import numpy as np

def codificar_hora(hora):
    radianes = 2 * np.pi * (hora / 24)
    return np.sin(radianes), np.cos(radianes)
```

Esta función se aplica a cada grafo al momento de su construcción, asignando los valores codificados a todos los nodos de forma uniforme (ya que en esta fase del proyecto todos los nodos comparten la misma franja horaria por grafo). En una extensión futura, donde los grafos puedan incluir nodos con distintas horas (por ejemplo, mediante trazas individuales), esta codificación seguiría siendo válida y útil.

Ventajas y posibles extensiones El uso de esta codificación presenta varias ventajas:

- Permite capturar la naturaleza periódica del tiempo sin introducir discontinuidades.
- Aumenta la expresividad del grafo al añadir información contextual relevante.
- Puede combinarse con mecanismos de atención o capas temporales en modelos GNN más complejos.

Además, esta técnica puede extenderse fácilmente para representar otras variables cíclicas, como días de la semana, meses, o patrones estacionales, lo que la convierte en una herramienta versátil para tareas que impliquen datos temporales en grafos.

5.4. Modelos desarrollados: Toy Models

Con el objetivo de explorar de forma progresiva el comportamiento de las Graph Neural Networks (GNNs) en un entorno controlado, se han desarrollado una serie de versiones incrementales del sistema, denominadas *toy models*. Cada modelo introduce mejoras o modificaciones respecto al anterior, permitiendo evaluar de forma aislada el impacto de distintos factores en la representación del grafo, el entrenamiento del modelo y la interpretación de los resultados. A la hora de programar, han sido de especial utilidad para usar como referencia los artículos [5] y [6].

5.4.1 toy-model-1

El primer modelo desarrollado constituye una versión simplificada del problema de predicción de demanda de pasajeros en una línea de autobús. Este modelo tiene como objetivo verificar el funcionamiento básico de una Graph Neural Network (GNN) en un entorno controlado, aislando los elementos esenciales del sistema: la estructura del grafo, las características de los nodos y la capacidad de aprendizaje del modelo.

Diseño del grafo Se modeló una pequeña red compuesta por seis nodos: cinco correspondientes a paradas intermedias (`P1` a `P5`) y uno final que representa la estación de destino. Las conexiones entre nodos se definieron mediante un grafo dirigido que representa la secuencia de paradas, con aristas unidireccionales desde cada parada a la siguiente, culminando en la estación. Esto da lugar a una topología lineal, simple pero suficiente para evaluar la capacidad del modelo de propagar información a lo largo del grafo.

Los atributos asociados a los nodos son unidimensionales y corresponden al número de pasajeros que suben en cada parada. El nodo correspondiente a la estación no tiene pasajeros asociados (valor cero), ya que su valor será estimado por la red. La variable objetivo del modelo (`y`) es la cantidad total de pasajeros que se espera que lleguen a la estación, la cual se obtiene como suma directa de los valores de las paradas intermedias (en este caso, 28).

Modelo de red y entrenamiento La arquitectura utilizada es una Graph Convolutional Network (GCN) básica, compuesta por dos capas convolucionales. La primera capa proyecta la entrada unidimensional de cada nodo a un espacio de 8 dimensiones latentes, seguida por una activación ReLU. La segunda capa reduce nuevamente la representación a una dimensión, generando una predicción por nodo. El entrenamiento se realiza optimizando el error cuadrático medio (MSE) entre la salida del nodo correspondiente a la estación y el valor real.

El modelo se entrena durante 200 épocas con un optimizador Adam y una tasa de aprendizaje de 0.01. La siguiente salida de consola recoge la evolución de la función de pérdida:

```
Data(x=[6, 1], edge_index=[2, 5], y=[1])
Nodos: 6
Aristas: 5
Features por nodo: 1
Epoch 0 - Loss: 847.0753
```

```
Epoch 20 - Loss: 691.2837
Epoch 40 - Loss: 544.2904
Epoch 60 - Loss: 390.9058
Epoch 80 - Loss: 237.3172
Epoch 100 - Loss: 111.5857
Epoch 120 - Loss: 36.0081
Epoch 140 - Loss: 6.6140
Epoch 160 - Loss: 0.4677
Epoch 180 - Loss: 0.0002
Predicción para la estación: 28.08
Valor real: 28.00
```

Resultados y análisis El modelo consigue predecir con gran precisión el valor objetivo, alcanzando un valor estimado de 28.08 frente al real de 28.00. Aunque este modelo es puramente exploratorio y no tiene valor predictivo real, sirve como punto de partida para validar el funcionamiento general del flujo de trabajo y sentar las bases para modelos posteriores más complejos.

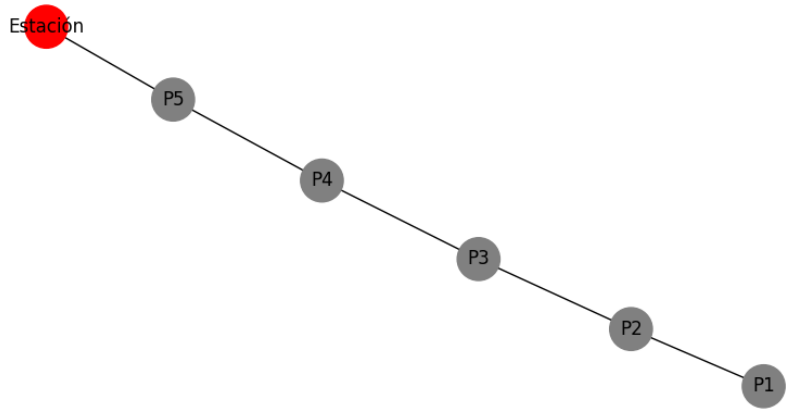


Figura 3: Representación del grafo lineal utilizado en `toy-model-1`, con cinco paradas (P1–P5) y una estación final.

5.4.2 toy-model-2

Este segundo modelo introduce una mejora significativa respecto al anterior al trabajar con dos líneas reales de autobús que conectan diversos barrios con la Estación de AVE en Segovia. El objetivo es mantener la simplicidad del entorno controlado, pero incorporando elementos más próximos a la aplicación real: múltiples líneas independientes, recorridos diferenciados y características adicionales por nodo. En este caso, cada línea se trata como un grafo independiente, y se entrena un modelo distinto para cada uno.

Diseño del grafo Se construyeron dos grafos dirigidos, uno por cada línea de autobús:

- **Línea 11:** Acueducto 3 → Plaza de Toros → Gerardo Diego → Estación AVE.

- **Línea 12:** Centro → Instituto Andrés Laguna → Ctra. S. Rafael → Estación AVE.

En ambos casos, los nodos representan las paradas de autobús y las aristas indican la secuencia de recorrido, acompañadas de un atributo adicional: el tiempo estimado de desplazamiento en minutos entre paradas. Esta información se almacena como atributo de arista (`edge_attr`) en el objeto `Data`.

Cada nodo contiene ahora dos características: el número de pasajeros que suben y el número de pasajeros que bajan en esa parada, reflejando una representación más rica del contexto de demanda. El nodo final, correspondiente a la estación, tiene como objetivo predecir el total de pasajeros que llegan, por lo que sus entradas en la matriz de características son cero (para subida) y el valor real total (para bajada), actuando como etiqueta supervisada del modelo.

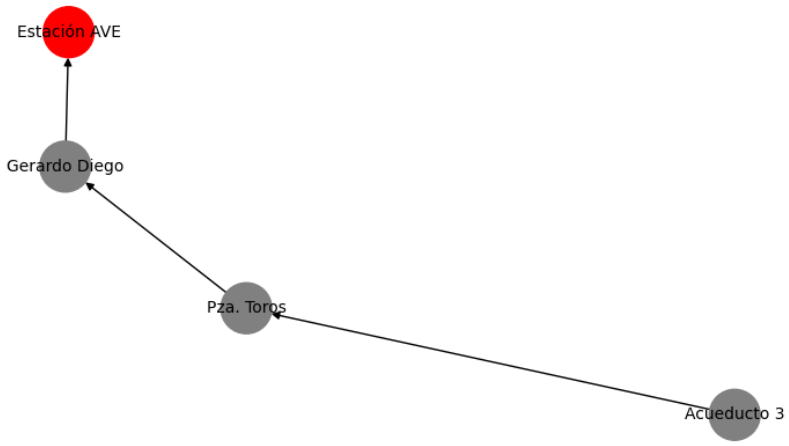


Figura 4: Grafo de la Línea 11.

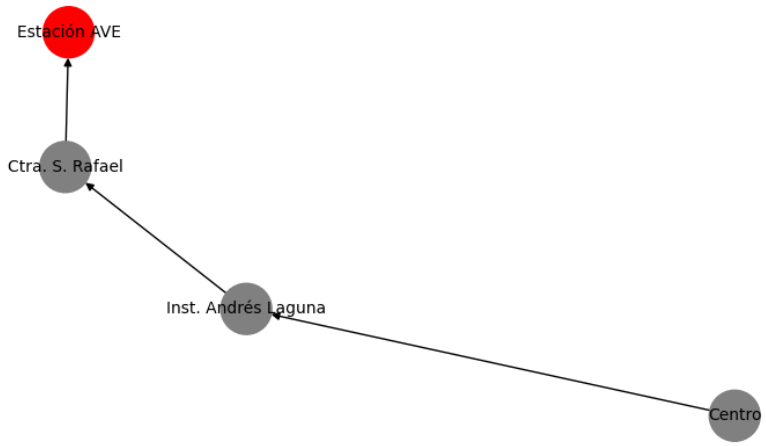


Figura 5: Grafo de la Línea 12.

Además, se construyó una visualización combinada que muestra ambas líneas, sus tiempos de recorrido y los nodos compartidos, facilitando la comprensión del trazado conjunto:

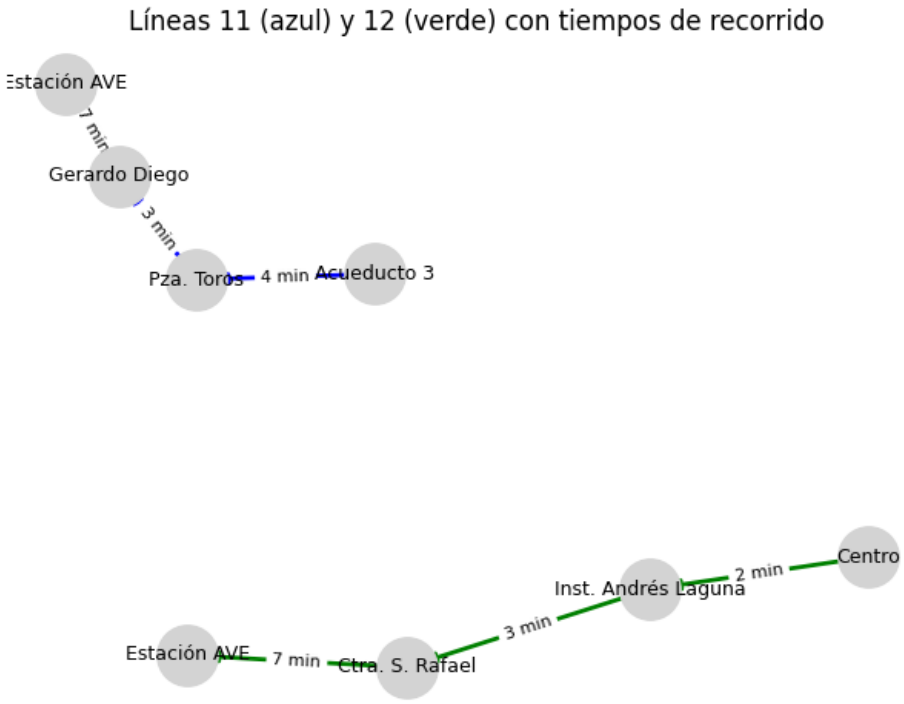


Figura 6: Representación conjunta de las líneas 11 (azul) y 12 (verde) con tiempos de recorrido.

Visualización geográfica en mapa interactivo Como complemento a la representación topológica de los grafos, se desarrolló un mapa interactivo utilizando la librería `folium`, que permite representar el recorrido geográfico real de ambas líneas sobre un mapa de Segovia. Para ello, se incorporaron las coordenadas de cada parada a partir de un archivo CSV `ubicaciones_paradas.csv` con información geoespacial.

El resultado es un archivo HTML exportado como `mapa_lineas_11_12.html`, que puede abrirse en el navegador. Al pasar el ratón por encima de los tramos de recorrido, se muestra un *tooltip* con la información del origen, destino y el tiempo estimado de viaje entre ambas paradas. Este tipo de representación facilita la validación de la estructura del grafo respecto al trazado real.

El archivo puede obtenerse ejecutando

```
python mapa_folium.py
```

desde la carpeta `src`.

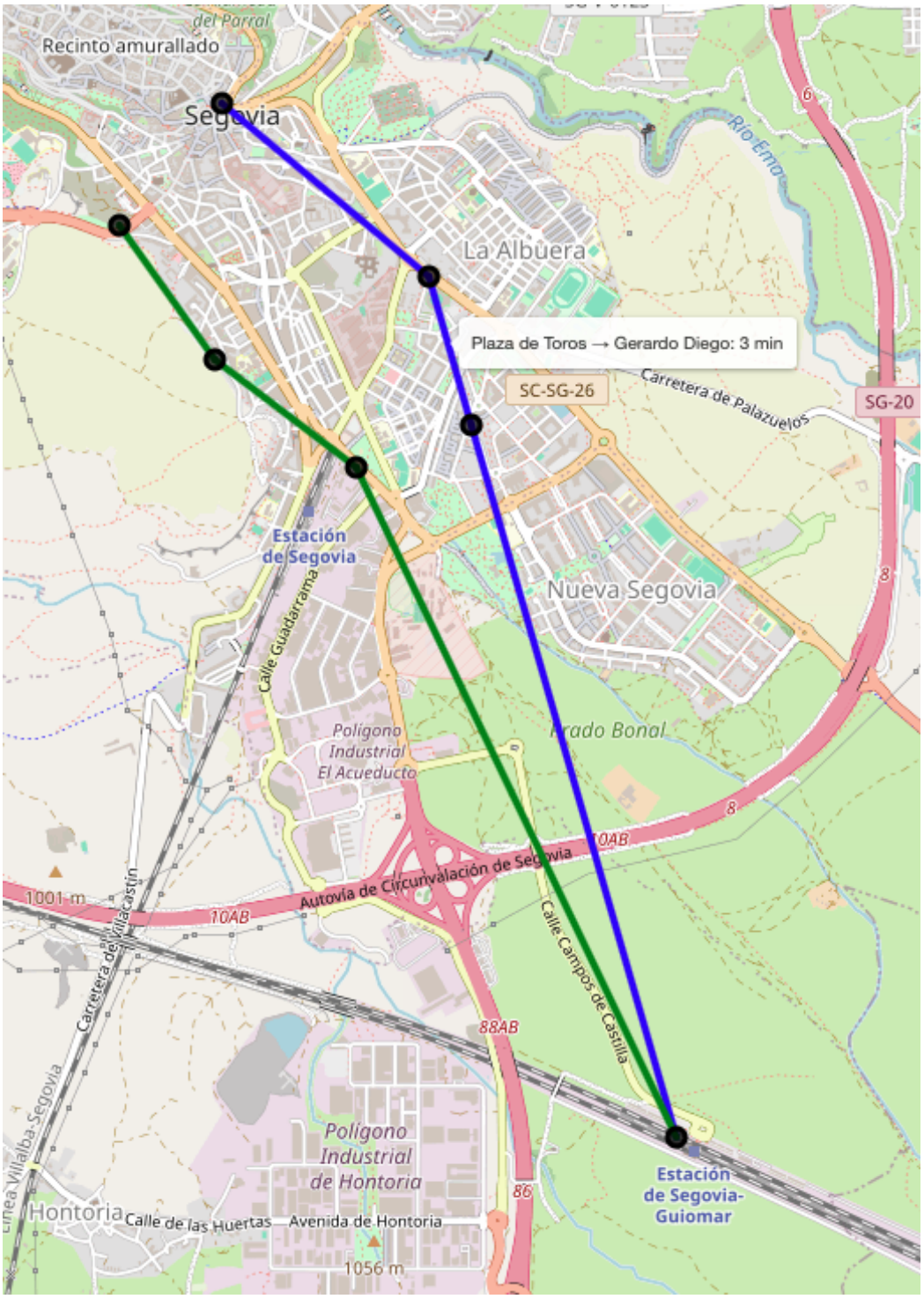


Figura 7: Visualización geográfica de las líneas 11 (azul) y 12 (verde) sobre mapa de Segovia, desarrollada con `folium`.

Modelo de red y entrenamiento Se utilizó nuevamente una arquitectura GCN de dos capas, adaptada a la nueva dimensionalidad de entrada (dos características por nodo). La primera capa proyecta el vector de entrada de dimensión 2 a un espacio latente de 8 dimensiones, seguido de una activación ReLU, y finalmente una segunda capa produce una predicción unidimensional por nodo.

El entrenamiento se realiza de forma independiente para cada grafo, utilizando una función de pérdida MSE entre el valor predicho en el nodo de la estación y el valor real. A continuación, se presenta la salida de consola con la evolución del entrenamiento:

```
===== Entrenando modelo para Línea 11 =====
Data(x=[4, 2], edge_index=[2, 3], edge_attr=[3, 1], y=[1])
Epoch 0 - Loss: 183.5444
Epoch 20 - Loss: 58.4934
Epoch 40 - Loss: 2.8182
Epoch 60 - Loss: 0.9039
Epoch 80 - Loss: 0.0028
Epoch 100 - Loss: 0.0092
Epoch 120 - Loss: 0.0021
Epoch 140 - Loss: 0.0001
Epoch 160 - Loss: 0.0000
Epoch 180 - Loss: 0.0000
Predicción para la estación: 14.00
Valor real: 14.00

===== Entrenando modelo para Línea 12 =====
Data(x=[4, 2], edge_index=[2, 3], edge_attr=[3, 1], y=[1])
Epoch 0 - Loss: 516.4146
Epoch 20 - Loss: 367.7339
Epoch 40 - Loss: 241.3574
Epoch 60 - Loss: 116.9827
Epoch 80 - Loss: 26.5657
Epoch 100 - Loss: 0.1781
Epoch 120 - Loss: 0.3945
Epoch 140 - Loss: 0.0004
Epoch 160 - Loss: 0.0063
Epoch 180 - Loss: 0.0001
Predicción para la estación: 19.01
Valor real: 19.00
```

Resultados y análisis Los resultados muestran un ajuste prácticamente perfecto en ambos casos. El modelo consigue predecir con alta precisión el número total de pasajeros que llegan a la estación en cada línea, incluso partiendo de grafos con solo cuatro nodos. Estos resultados, si bien no son generalizables por su carácter sintético, validan que el modelo y la estructura del grafo son capaces de capturar relaciones útiles a pequeña escala y sientan las bases para futuras extensiones más realistas.

5.4.3 toy-model-3

Con el objetivo de aproximarse progresivamente a un entorno más realista, el tercer modelo desarrollado introduce una dimensión temporal en la predicción de demanda. Se mantiene la estructura

básica de líneas independientes y nodos con atributos de subida y bajada, pero se incorpora una nueva característica: la franja horaria en la que se produce el viaje. Este factor es especialmente relevante en entornos de movilidad urbana, donde la demanda varía significativamente a lo largo del día.

Diseño del grafo Cada línea (Línea 11 y Línea 12) se modela como un grafo dirigido compuesto por cuatro nodos y tres aristas, como en el modelo anterior. A diferencia de las versiones anteriores, los atributos de los nodos incluyen ahora tres dimensiones:

- Número de pasajeros que suben en la parada.
- Número de pasajeros que bajan.
- Codificación numérica de la franja horaria.

Se definen cuatro franjas horarias representativas: 10:00, 12:00, 16:00 y 20:00, codificadas como valores numéricos de 0,0 a 3,0. La estación AVE sigue actuando como nodo objetivo en cada línea, y se utiliza como etiqueta el número total de pasajeros que llegan a esta estación en cada franja.

Los datos de subida y bajada se generan de forma controlada mediante diccionarios codificados, y se introduce una ligera aleatoriedad en los valores de bajada en las paradas intermedias para simular escenarios más realistas. De este modo, el modelo debe aprender a interpretar tanto la estructura del grafo como el contexto horario en la estimación de la demanda.

Modelo de red y entrenamiento El modelo empleado es una red neuronal convolucional sobre grafos (GCN) con dos capas. La entrada está compuesta ahora por tres características por nodo. La primera capa transforma esta representación a un espacio latente de 8 dimensiones, seguido de una función de activación ReLU, y finalmente una segunda capa proyecta el resultado a una dimensión escalar por nodo.

El modelo se entrena por separado para cada línea y franja horaria, utilizando como función de pérdida el error cuadrático medio (MSE) entre la predicción para la estación y el valor real. A continuación se muestra una selección de la salida de consola correspondiente a los entrenamientos realizados:

```
===== Franja horaria: 10:00 (código 0.0) =====
...
Predicción para la estación: 15.00
Valor real: 15.00
...
Predicción para la estación: 20.00
Valor real: 20.00

===== Franja horaria: 12:00 (código 1.0) =====
...
Predicción para la estación: 8.00
Valor real: 8.00
...
Predicción para la estación: 9.00
```

```
Valor real: 9.00

===== Franja horaria: 16:00 (código 2.0) =====
...
Predicción para la estación: 18.00
Valor real: 18.00
...
Predicción para la estación: 23.00
Valor real: 23.00

===== Franja horaria: 20:00 (código 3.0) =====
...
Predicción para la estación: 6.00
Valor real: 6.00
...
Predicción para la estación: 6.00
Valor real: 6.00
```

Los resultados obtenidos se guardan en un archivo CSV que permite realizar análisis posteriores y alimentar herramientas de visualización.

Visualización animada por franja horaria Como complemento a la interpretación de resultados, se desarrolló una visualización interactiva en formato HTML con la librería `folium`, que permite representar la evolución de la demanda a lo largo del día mediante un mapa animado.

Para ello, se utilizaron las predicciones generadas para cada línea y franja horaria, ubicando un marcador sobre la estación AVE con un tamaño proporcional al valor estimado. El resultado es un *heatmap* animado que representa visualmente cómo varía la demanda en la estación en función del momento del día.

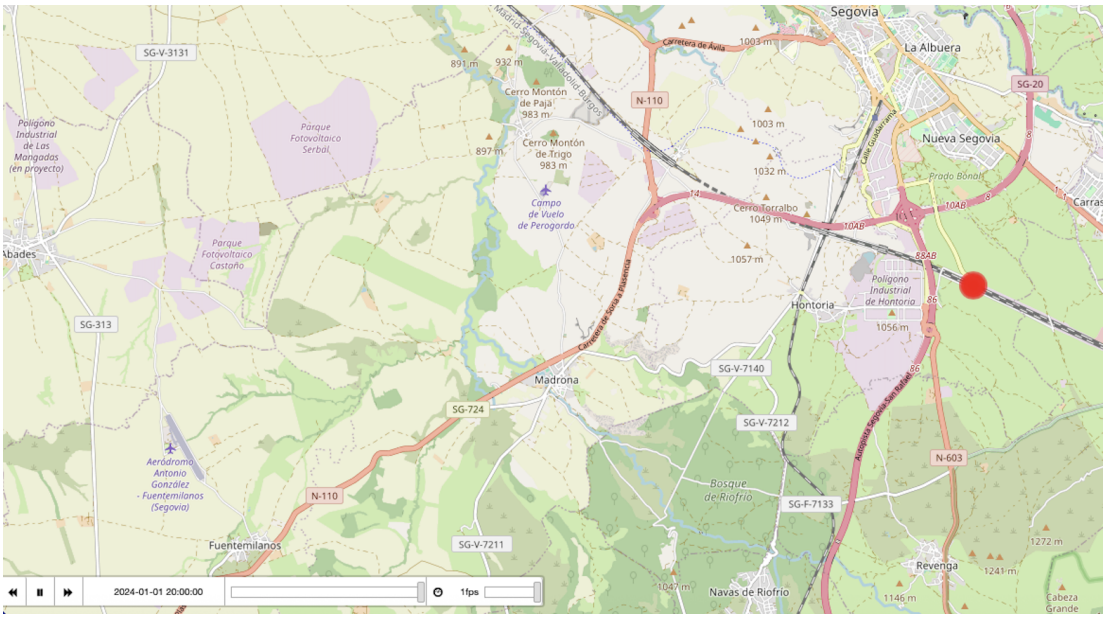


Figura 8: Heatmap animado generado con `folium`, mostrando la evolución horaria de la demanda en la Estación AVE.

El mapa generado se guarda como `mapa_heatmap_animado.html`, y puede reproducirse en cualquier navegador web. Permite visualizar la dinámica horaria de la estación mediante una interfaz que avanza automáticamente a lo largo de las franjas, representando el volumen de pasajeros en forma de círculos rojos de tamaño creciente.

Resultados y análisis El modelo es capaz de adaptarse correctamente a los patrones de cada franja horaria, con errores residuales mínimos incluso en contextos más variables. La introducción de la hora como tercera característica por nodo se demuestra efectiva, permitiendo al modelo diferenciar correctamente entre contextos temporales distintos con la misma topología de grafo.

Este modelo constituye un paso importante hacia la generación de representaciones dinámicas de la demanda de transporte en red, y avanza hacia desarrollos futuros que incluyan codificaciones más sofisticadas del tiempo, múltiples líneas simultáneas y datasets reales de movilidad urbana.

5.4.4 toy-model-4

En esta cuarta y última versión del modelo, se introduce una mejora significativa orientada a lograr una mayor capacidad de generalización y escalabilidad. A diferencia de las versiones anteriores, en las que cada modelo se entrenaba de forma independiente por franja horaria y línea, en este caso se adopta un enfoque *multigrafo*: se entrena un único modelo GCN sobre un conjunto de grafos diversos que representan distintas franjas horarias y líneas de autobús.

Diseño del grafo y codificación temporal La estructura de cada grafo se mantiene similar: cuatro nodos que representan las paradas de cada línea y tres aristas secuenciales que definen el recorrido. Sin embargo, el vector de características por nodo se amplía ahora a cuatro dimensiones:

- Pasajeros que suben.
- Pasajeros que bajan.
- Codificación seno del momento del día.
- Codificación coseno del momento del día.

La codificación temporal se calcula utilizando funciones seno y coseno sobre los minutos transcurridos desde medianoche, normalizados a una rotación completa de 24 horas. Esta técnica permite representar la hora como una variable circular continua, evitando los saltos artificiales entre valores horarios (por ejemplo, entre 23:00 y 00:00), y se ha demostrado eficaz en múltiples contextos de aprendizaje automático.

Modelo de red y entrenamiento multifranja El modelo utilizado sigue siendo una GCN de dos capas, ajustada ahora a la entrada de dimensión cuatro. La primera capa proyecta a un espacio latente de 8 dimensiones, seguida de una activación ReLU, y una segunda capa produce una predicción escalar por nodo.

Se generan ocho grafos en total, combinando cuatro franjas horarias (10:00, 12:00, 16:00 y 20:00) con dos líneas (11 y 12). Estos grafos se encapsulan en un único conjunto de entrenamiento, que se recorre utilizando un `DataLoader` de PyTorch Geometric. El modelo aprende así a inferir patrones compartidos entre distintas líneas y horarios, aumentando su robustez sin incrementar el coste computacional.

```
Epoch 0 - Loss: 793.1296
Epoch 20 - Loss: 272.3032
Epoch 40 - Loss: 49.9810
Epoch 60 - Loss: 2.0152
Epoch 80 - Loss: 0.1110
Epoch 100 - Loss: 0.0156
Epoch 120 - Loss: 0.0006
Epoch 140 - Loss: 0.0000
Epoch 160 - Loss: 0.0000
Epoch 180 - Loss: 0.0000
```

Evaluación y visualización de resultados Tras el entrenamiento, el modelo se evalúa sobre los mismos ocho grafos. Los resultados muestran una predicción prácticamente perfecta para cada combinación de franja y línea, con errores inferiores a 0.01 en todos los casos. Estos se recogen en un archivo CSV y se visualizan mediante un gráfico de barras que compara el valor real y la predicción.

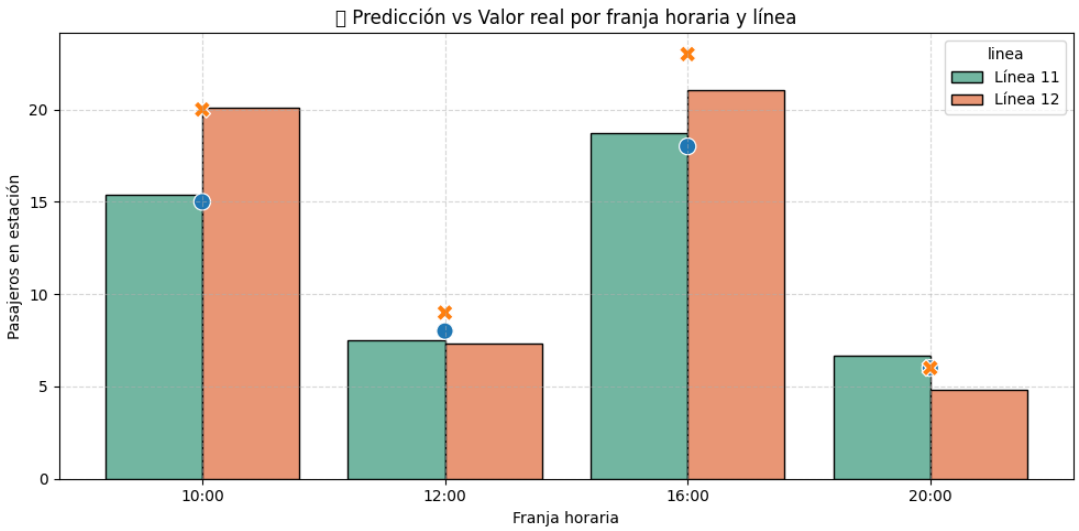


Figura 9: Predicción vs valor real para cada franja horaria y línea en el modelo multifranja.

Este tipo de visualización facilita la comparación entre líneas y horas, confirmando que el modelo ha aprendido a diferenciar correctamente contextos de entrada incluso cuando la topología del grafo es similar.

Conclusiones del modelo El modelo `toy-model-4` representa un avance conceptual respecto a las versiones anteriores, al introducir codificación temporal continua y entrenamiento conjunto sobre múltiples contextos. Esto permite:

- Reutilizar un único modelo entrenado para diferentes escenarios sin necesidad de duplicar recursos.
- Capturar relaciones temporales sutiles mediante codificación seno-coseno.
- Preparar el terreno para su aplicación a grafos dinámicos o secuenciales.

La capacidad de generalización lograda con este enfoque lo convierte en una base sólida para futuros modelos reales basados en datos de movilidad urbana a gran escala.

5.5. Modelos aplicados a datos reales: Buses de Brasil

5.5.1 Preparación del dataset y limpieza

Con el objetivo de aplicar los modelos desarrollados a un caso real, se utilizó un conjunto de datos de viajes de autobuses en Brasil. Este dataset fue proporcionado como alternativa al de Segovia, que inicialmente se consideró para el proyecto pero que finalmente no estuvo disponible. La información se distribuye en varios ficheros `.csv`, cada uno correspondiente a un mes distinto, conteniendo millones de registros sobre billetes vendidos y trayectos realizados en el sistema de transporte público.

Descripción general del dataset Los ficheros contenían información heterogénea, escrita originalmente en portugués, sobre los viajes realizados por autobuses urbanos. Entre las columnas más relevantes se encontraban:

- `ponto_origem_viagem` y `ponto_destino_viagem`: paradas de origen y destino del trayecto.
- `nu_linha`: número de línea del autobús.
- `data_viagem`: fecha del viaje.
- `hora_viagem`: hora del inicio del viaje.

Además, se contaban con columnas adicionales como el identificador del billete, el tipo de billete o la tarifa, que fueron descartadas en una etapa posterior por no ser relevantes para el modelado mediante GNNs.

Traducción y normalización de campos Dado que los nombres de las variables estaban en portugués, se elaboró un diccionario de traducción de variables al español con el objetivo de facilitar la comprensión y trazabilidad durante las etapas de análisis y desarrollo. Este diccionario se almacenó como un documento adicional en el repositorio.

Posteriormente, se unificaron todos los archivos mensuales en un único `DataFrame` utilizando `pandas`. En esta etapa también se homogeneizaron formatos de fechas y horas, convirtiendo los campos `data_viagem` y `hora_viagem` en objetos datetime estándar de Python, lo cual permitió extraer variables derivadas como el día del mes o la hora del día.

Limpieza de duplicados y valores faltantes Tras la carga y unificación de los datos, se detectó la presencia de registros duplicados, especialmente en combinaciones idénticas de origen-destino-línea-hora. Se procedió a eliminar estos duplicados mediante la función `drop_duplicates()`, asegurando que cada trayecto observado solo se contase una vez en el conjunto base sobre el que se construiría el grafo.

También se identificaron algunas filas con valores nulos o mal formateados en los campos críticos (como la hora del viaje). Estos registros fueron descartados ya que representaban una proporción ínfima del total y su imputación hubiera introducido ruido adicional.

Agregación para tareas de predicción Para la tarea de predicción de demanda, se transformó el conjunto de datos en un grafo dirigido donde cada arista representa una conexión directa observada entre un par de paradas (origen y destino), en una línea y franja horaria determinadas. Se agrupó el conjunto de datos original mediante una operación `groupby()` con las claves:

`ponto_origem_viagem`, `ponto_destino_viagem`, `nu_linha`, `hora`

Y se calculó el número total de viajes observados para cada combinación, guardado como la variable `num_viajes`, que posteriormente se usaría como variable objetivo (target) en la tarea de regresión.

Codificación y escalado Para modelar el grafo con `PyTorch Geometric`, fue necesario codificar los nodos (paradas de autobús) como enteros consecutivos. Se utilizó `LabelEncoder` de `sklearn` sobre el conjunto unificado de nodos (orígenes y destinos).

Por otro lado, se aplicó `StandardScaler` a las variables de entrada asociadas a las aristas (línea y hora) para normalizarlas y evitar que escalas distintas dificultaran el entrenamiento del modelo. También se aplicó una transformación logarítmica `log1p()` a la variable `num_viajes` para suavizar la distribución, que presentaba colas muy largas y valores atípicos.

Detección y eliminación de outliers Dado que algunos pares origen-destino presentaban volúmenes de viajes anormalmente altos (con valores superiores a 10 000 viajes en una hora concreta), se realizó una inspección de outliers. Se eliminó el 1 % superior de observaciones en cuanto a demanda, por considerarse que no representaban patrones habituales y podían sesgar el aprendizaje del modelo.

División en conjuntos de entrenamiento y test Para evaluar el rendimiento real del modelo, el conjunto de datos agregado y preprocesado se dividió en dos subconjuntos:

- **Entrenamiento**: 80 % de las observaciones, utilizado para ajustar los pesos del modelo.
- **Test**: 20 % restante, reservado para validar la capacidad del modelo para generalizar.

Esta división se hizo de forma aleatoria utilizando la función `train_test_split()` de `sklearn`, fijando una semilla aleatoria para garantizar reproducibilidad.

En resumen, esta etapa de preparación fue esencial para garantizar que los datos se encontraban en un formato correcto, coherente y representativo del problema real a modelar. La limpieza y preprocesamiento adecuados permitieron extraer información relevante del dataset original, reduciendo ruido y facilitando la posterior construcción del grafo y entrenamiento de modelos GNN.

5.5.2 Modelo 1: Predicción de enlaces

El primer modelo funcional desarrollado para abordar el problema de predicción en redes de transporte fue un modelo de **predicción de enlaces** (*link prediction*) utilizando una Red Neuronal de Grafos (GNN). El objetivo principal de este modelo es predecir si, dada una parada de origen y otra de destino, existe una conexión real (una arista) entre ambas en el grafo observado.

Planteamiento del problema Formalmente, se trata de un problema de clasificación binaria sobre aristas del grafo. Dado un conjunto de pares de nodos, algunos conectados y otros no, el modelo debe aprender una función que, a partir de la información estructural del grafo, prediga la probabilidad de que una arista real exista entre ellos.

Este enfoque resulta útil en contextos donde se dispone de datos parciales y se desea inferir rutas potenciales no observadas, o completar información faltante en redes incompletas.

Construcción del grafo Para construir el grafo sobre el cual se entrenó el modelo, se utilizó el conjunto `viajes_limpios.csv`, previamente depurado. Se generaron los siguientes componentes:

- `x`: vector de características de nodos. Dado que no se disponía de atributos numéricos asociados a cada parada, se optó por una codificación de identidad: un índice entero para cada nodo.

- `edge_index`: matriz de adyacencia en formato COO, con pares (origen, destino) observados en el dataset.
- `edge_label`: vector binario indicando si la arista corresponde a una conexión real (1) o negativa generada artificialmente (0).

Para equilibrar el dataset, se generaron ejemplos negativos mediante muestreo aleatorio de pares de nodos que no estaban conectados. El número de aristas positivas y negativas fue igualado para garantizar un aprendizaje balanceado.

Arquitectura del modelo El modelo implementado sigue la estructura clásica de una *Graph Convolutional Network* (GCN), adaptada para tareas de clasificación sobre enlaces. El flujo general fue:

1. Aplicación de dos capas `GCNConv` para generar **embeddings vectoriales** para cada nodo.
2. Cálculo de la similitud entre nodos mediante el *producto escalar* entre los vectores de los nodos origen y destino.
3. Interpretación del resultado como un **logit**, aplicando la función `sigmoid` para obtener una probabilidad.

Entrenamiento y evaluación El modelo se entrenó durante 100 épocas con la función de pérdida `Binary Cross Entropy with Logits` y el optimizador Adam. El conjunto de datos contenía más de 21.000 aristas positivas y otras tantas negativas.

La evolución de la pérdida mostró una disminución progresiva:

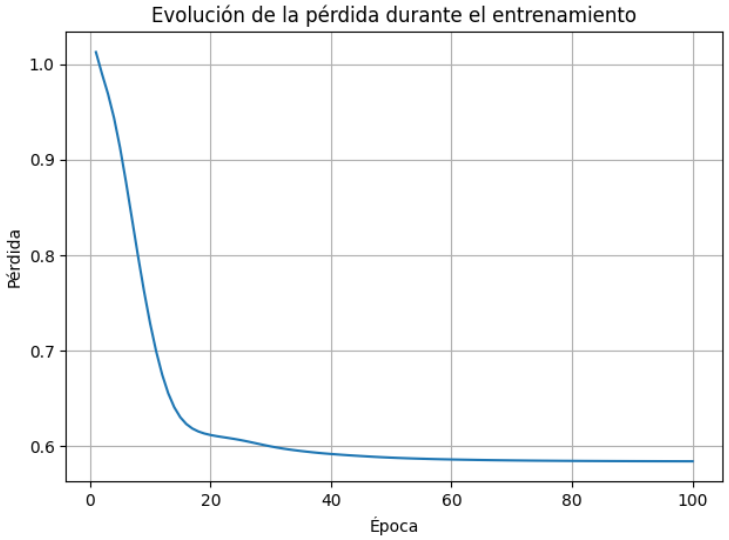


Figura 10: Evolución de la pérdida durante el entrenamiento (experimento 1).

En una segunda ejecución del modelo, los resultados de entrenamiento fueron consistentes:

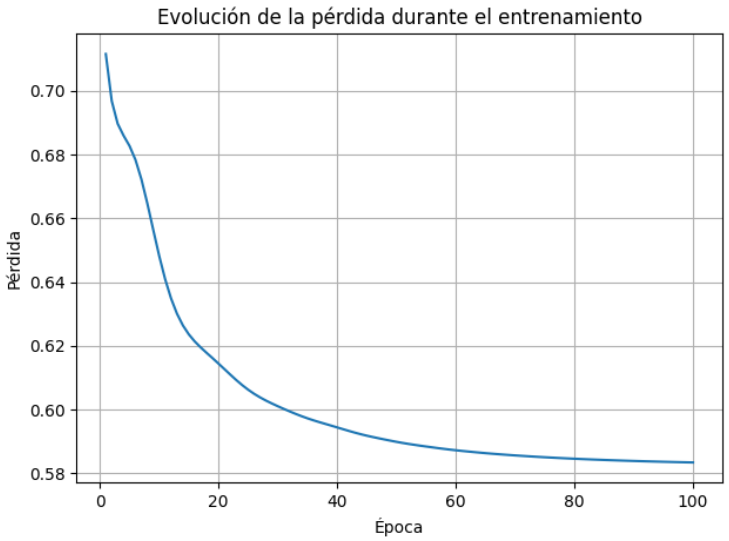


Figura 11: Evolución de la pérdida durante el entrenamiento (experimento 2).

El modelo alcanzó un AUC ROC de aproximadamente 0.90, lo que indica una buena capacidad de separar enlaces reales de los generados aleatoriamente. No obstante, la métrica de `accuracy` se mantuvo en torno a 0.5 debido al umbral fijo aplicado a los logits (p. ej., predicción positiva si score ¿0), lo que sugiere que podría mejorarse con una calibración más precisa del umbral de decisión.

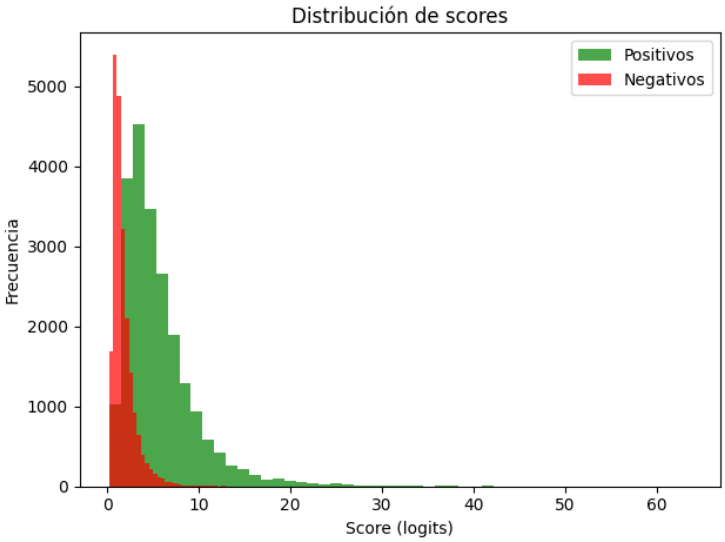


Figura 12: Distribución de scores para aristas positivas y negativas (experimento 1).

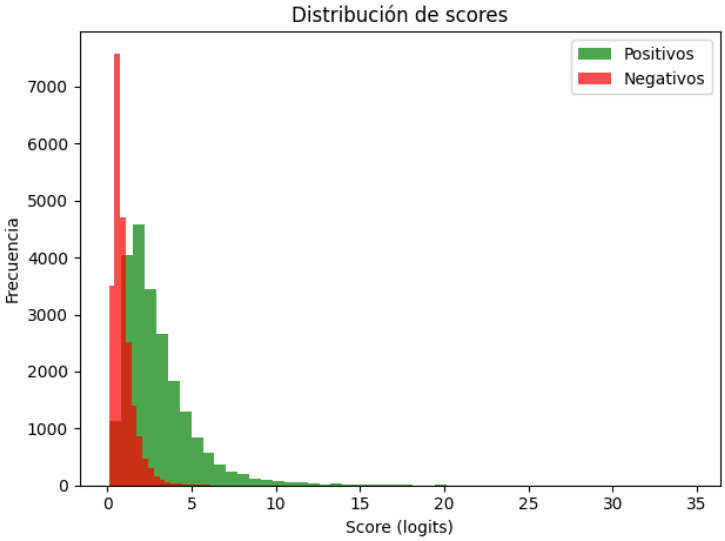


Figura 13: Distribución de scores para aristas positivas y negativas (experimento 2).

Visualización de predicciones Como paso adicional, se generó un subgrafo aleatorio de 100 nodos con el objetivo de visualizar las predicciones del modelo sobre conexiones reales y artificiales. En este grafo se representaron:

- Enlace **verde**: conexión real correctamente predicha como existente (verdadero positivo).
- Enlace **rojo**: conexión real que el modelo no detectó (falso negativo).
- Enlace **azul**: conexión negativa predicha como positiva (falso positivo).

Se añadieron etiquetas reales a los nodos utilizando el `LabelEncoder` inverso para facilitar la interpretación semántica. No obstante, dado el elevado número de nodos en el grafo original y la alta densidad de conexiones, la legibilidad de los nombres queda limitada, especialmente en la región central del grafo.

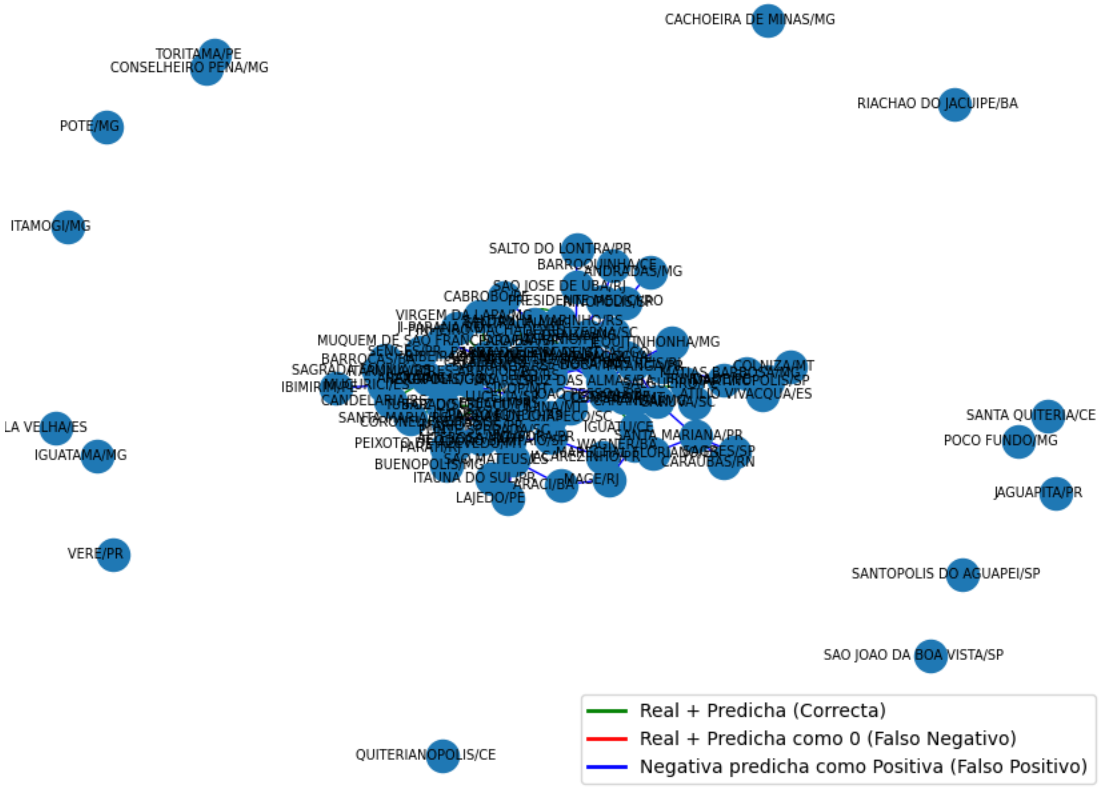


Figura 14: Subgrafo aleatorio con visualización de predicciones de enlace. La densidad de conexiones reduce la legibilidad de nombres, pero permite comprobar el procedimiento de inspección estructural.

Limitaciones del modelo Las principales limitaciones detectadas en este primer enfoque fueron:

- No permite cuantificar el flujo de pasajeros entre nodos, solo su existencia.
- No incorpora información contextual como la línea del viaje o la hora.
- Los embeddings generados son estáticos y homogéneos para todos los contextos.
- El conjunto de entrenamiento requiere generación artificial de ejemplos negativos.

Conclusión Este modelo de predicción de enlaces se considera una base útil para entender la estructura de la red y explorar el uso de GNNs en problemas de transporte. Sin embargo, su capacidad práctica es limitada en tareas de análisis de demanda o planificación de rutas. Por ello, se desarrollaron posteriormente modelos orientados a la predicción de cantidades y flujos, como se describe en los siguientes apartados.

5.5.3 Modelo 1: Link prediction (Mejoras)

1. Problemas detectados en el modelo anterior Aunque el primer modelo de predicción de enlaces permitió una aproximación funcional al problema de predicción binaria entre pares de paradas, presentó varias carencias estructurales y técnicas que limitaron su rendimiento y capacidad de generalización:

- **Representación pobre de los nodos:** los nodos solo contenían un ID numérico sin ningún tipo de información adicional, lo que impedía a la GNN aprender patrones significativos basados en características.
- **Negativos triviales:** los pares negativos se generaban aleatoriamente entre nodos no conectados, lo que daba lugar a ejemplos demasiado obvios y artificiales. Esto provocaba que el modelo tuviese una alta AUC pero baja capacidad real de discriminación.
- **Solapamiento en scores:** el histograma de puntuaciones mostraba una fuerte superposición entre las puntuaciones asignadas a enlaces reales y falsas conexiones, con una `accuracy` inferior al 50 %, muy cercana al azar.
- **Sobreajuste a hubs:** existía el riesgo de que el modelo simplemente aprendiera que ciertos nodos (e.g., `SAO PAULO/SP`, `RIO DE JANEIRO/RJ`) estaban presentes en muchas conexiones, sin entender patrones más generales.

Ante estas limitaciones, se implementó una versión completamente rediseñada y mejorada del modelo, como se detalla a continuación.

2. Mejoras implementadas La nueva versión del modelo de predicción de enlaces introduce una serie de mejoras sustanciales en la representación de los nodos, en la arquitectura del modelo y en la estrategia de generación de ejemplos negativos:

- **Features enriquecidos para los nodos:** cada nodo cuenta ahora con un vector de características que incluye:
  - Hora media de salida desde la parada (extraída desde los datos de viaje).
  - Estado geográfico (sufijo del nombre del nodo), codificado mediante `one-hot`.
  - Línea más frecuente observada en dicha parada.
- **Normalización de features:** se aplicó `StandardScaler` para que todas las variables tuviesen magnitudes comparables.
- **Arquitectura GCN mejorada:**
  - Dos capas `GCNConv` para capturar relaciones de segundo orden.

- Capas `BatchNorm1d` para estabilizar la activación.
- Capa `Dropout (0.2)` para mitigar el sobreajuste.

- **Negativos inteligentes:** en lugar de usar pares aleatorios, los negativos se seleccionan de forma más informativa, generando *hard negatives* (parejas plausibles pero no observadas).
- **Función de predicción por lote:** se añadió una función `predict_links_batch` que permite evaluar múltiples pares de nodos, proporcionando el score y la probabilidad estimada.

3. Análisis de resultados El nuevo modelo fue entrenado durante 100 épocas con una función de pérdida `binary cross-entropy`. La evolución de la pérdida mostró un descenso constante, estabilizándose alrededor de 0.0746.

- **Métricas finales:**
  - `ROC AUC: 0.9859`, indicando una excelente capacidad de discriminación.
  - `Accuracy: 0.9743`, reflejando una clara mejora frente a la versión inicial.
- **Distribución de scores:** la figura 15 muestra una separación clara entre los scores asignados a enlaces positivos (verde) y negativos (rojo), con escaso solapamiento entre ambos.
- **Curva de pérdida:** la figura 16 muestra cómo la pérdida decrece de forma estable, sin señales de sobreajuste.
- **Predicciones individuales:** al consultar conexiones conocidas como `SAO PAULO/SP { RIO DE JANEIRO/RJ`, el modelo devuelve puntuaciones superiores a 60 y una probabilidad del 100 %, lo que confirma su capacidad para identificar relaciones reales.

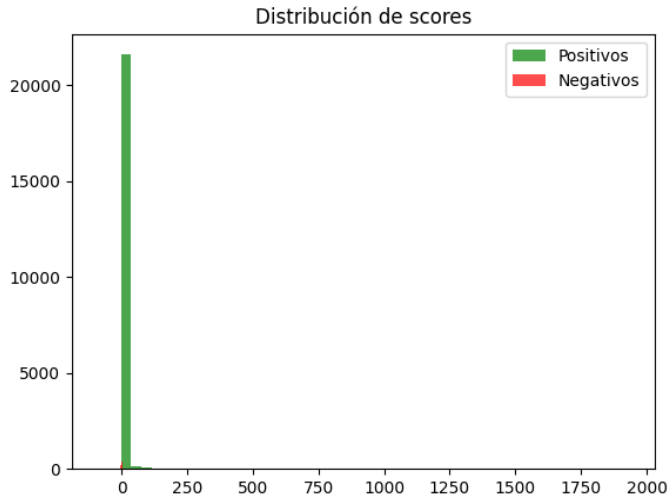


Figura 15: Distribución de scores para enlaces reales y negativos.

Figura 16: Evolución de la pérdida durante el entrenamiento del modelo.

4. Limitaciones y mejoras futuras A pesar del rendimiento alcanzado, se identifican posibles líneas de mejora:

- **Normalización adicional de activaciones:** algunos scores son excesivamente altos, por lo que sería recomendable aplicar técnicas como `LayerNorm` o añadir una capa de salida con activación sigmoide.
- **Nuevas arquitecturas:** explorar modelos como `GraphSAGE`, `GAT` o `GIN` podría aportar mayor flexibilidad para capturar relaciones complejas.
- **Visualización de embeddings:** aplicar `UMAP` o `PCA` sobre los embeddings generados permitiría visualizar la estructura del grafo y detectar clústeres.
- **Predicción masiva exportable:** sería útil exportar los resultados del modelo a CSV para su análisis en dashboards u otros sistemas externos.
- **Ampliación de atributos nodales:** podrían añadirse variables geoespaciales, demográficas o de conectividad para enriquecer aún más la representación de cada nodo.

En conjunto, estas mejoras permitirían consolidar el modelo actual como un sistema fiable de predicción de enlaces, aplicable tanto para inferencia de rutas faltantes como para sistemas de recomendación o planificación en redes de transporte.

5.5.4 Modelo 2: Predicción de demanda (regresión)

El segundo modelo desarrollado aborda una tarea más ambiciosa: la **predicción de la demanda** entre pares de puntos de la red de transporte interestadual brasileña. El objetivo es estimar cuántos viajes se realizan entre un nodo origen y un nodo destino en una franja horaria determinada, utilizando una arquitectura basada en *Graph Neural Networks* adaptada para regresión en aristas.

Formulación del problema A diferencia del modelo anterior, centrado en clasificación binaria, este modelo se enmarca dentro de una tarea de **regresión supervisada**. El modelo aprende a predecir un valor continuo —el número de viajes— para cada arista del grafo, utilizando tanto información estructural como atributos asociados a cada conexión (por ejemplo, la línea de autobús, la hora del día y el día de la semana).

Preparación del grafo A partir del conjunto limpio `viajes_limpios.csv`, se construyó un nuevo grafo dirigido. Cada nodo representa una parada de autobús, y cada arista representa un flujo de viajes entre un par origen–destino para una combinación de línea, hora y día. El preprocesamiento consistió en:

1. Conversión de la hora a formato entero (0–23) y extracción del día de la semana.
2. Agrupación por origen, destino, línea, hora y día para obtener el número de viajes reales.
3. Eliminación de valores atípicos por encima del percentil 99 de demanda.
4. Codificación de nodos con `LabelEncoder`.
5. Construcción de `edge_index`, `edge_attr` (línea, hora, día) y `edge_label` (número de viajes).
6. Aplicación de `StandardScaler` sobre los atributos de arista y transformación logarítmica `log1p` sobre el número de viajes para estabilizar la varianza.

División de los datos El conjunto de datos se dividió aleatoriamente en entrenamiento (80 %) y test (20 %), preservando una muestra variada de combinaciones origen–destino y franjas horarias. A diferencia de la división temporal realizada en toy-models, en este caso se buscaba maximizar la diversidad interna para facilitar el aprendizaje del modelo y su evaluación general.

Arquitectura del modelo El modelo implementado, denominado `GCNEdgeRegressor`, combina capas convolucionales con una red `MLP` final para la estimación de demanda. Su funcionamiento se basa en:

- Dos capas `GCNConv` que generan **embeddings** para cada nodo a partir de su posición en el grafo.
- Para cada arista, se concatenan los embeddings del nodo origen y destino junto a los atributos normalizados de la arista (línea, hora y día).
- Esta concatenación se pasa por una red densa con activación `ReLU` que devuelve un valor escalar estimado: el número de viajes.

Entrenamiento y métricas El modelo fue entrenado durante 100 épocas utilizando la función de pérdida `MSELoss` y el optimizador `Adam` con tasa de aprendizaje 0.01. Tras el entrenamiento, se evaluó sobre el conjunto de test mediante dos métricas estándar:

- **MSE (Mean Squared Error)**: cuantifica el error medio cuadrático entre valores reales y predichos.

- **R^2 (Coeficiente de determinación)**: mide la proporción de varianza explicada por el modelo.

Los resultados obtenidos fueron:

- `MSE`: 3886.10
- R^2: –0.37

Visualización de resultados Se generó una gráfica de dispersión que compara los valores reales con las predicciones sobre el conjunto de test. La línea roja representa la recta identidad ($y = x$). Puede observarse cómo la mayoría de predicciones tienden a agruparse en valores bajos, lo que refleja el predominio de bajas demandas en el conjunto de datos. Sin embargo, el modelo presenta una alta variabilidad en las predicciones, especialmente en valores intermedios, lo que genera una pérdida de precisión en los casos con mayor número de viajes.

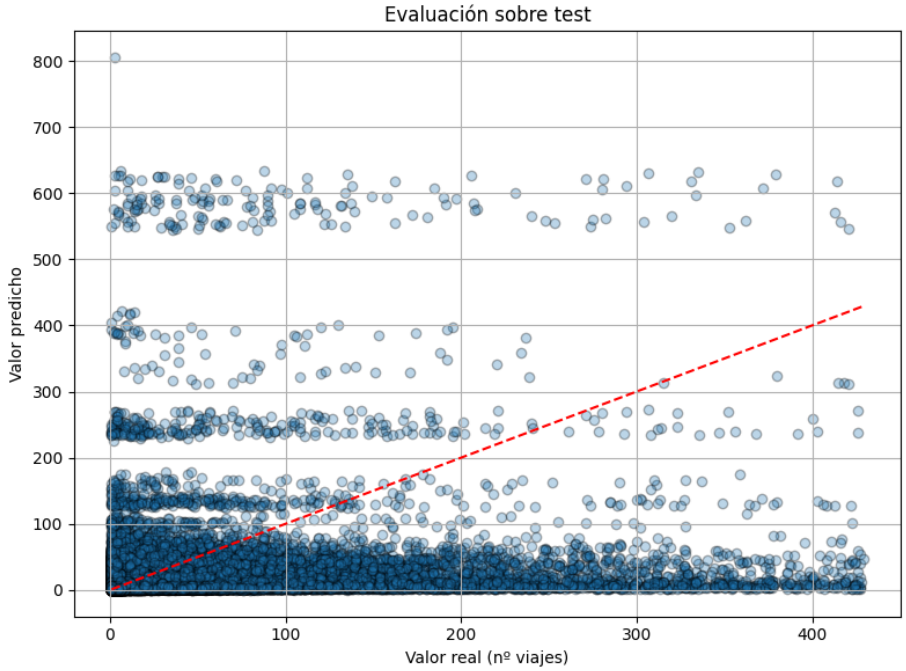


Figura 17: Evaluación sobre el conjunto de test: valores predichos vs reales. El modelo tiende a sobrestimar las demandas bajas e infraestimar las altas.

Limitaciones y observaciones A pesar de una arquitectura razonablemente adaptada al problema, los resultados obtenidos no alcanzan valores satisfactorios en términos de precisión. Algunas de las posibles causas identificadas son:

- **Features insuficientes**: únicamente se consideraron línea, hora y día de la semana. Incluir variables como tipo de servicio, festividad, mes, clima o información histórica podría mejorar el rendimiento.
- **Desbalance severo**: la distribución de demanda es altamente asimétrica, con una gran mayoría de aristas con demanda baja y muy pocas con valores altos, dificultando el aprendizaje.
- **Arquitectura limitada**: se utilizó un modelo simple de tipo GCN. Alternativas como `GraphSAGE`, `GAT` o arquitecturas con atención contextual podrían capturar patrones más complejos.

Este modelo demuestra la viabilidad del enfoque y la utilidad de trabajar directamente sobre aristas en lugar de nodos, pero también pone de manifiesto la necesidad de mejoras adicionales para capturar con mayor fidelidad la dinámica de la demanda en redes reales de transporte.

5.5.5 Modelo 2: Predicción de demanda (regresión) (Mejoras)

1. Problemas del modelo anterior El modelo inicial para la tarea de predicción de demanda se concibió como una regresión sobre aristas en un grafo dirigido, en el que cada arista representa un flujo de pasajeros entre un punto de origen y un punto de destino, asociado a una línea de autobús, una hora y un día determinados. El objetivo era estimar el número de viajes entre esos puntos en función de las características estructurales y temporales.

Sin embargo, el rendimiento del modelo inicial fue muy limitado, como demuestran las métricas obtenidas tras el entrenamiento:

- Un **error cuadrático medio (MSE)** extremadamente alto: 4.382.152, lo que sugiere una gran discrepancia entre los valores predichos y los reales.
- Un **coeficiente de determinación (R^2)** de -1542.76, indicando que el modelo era incapaz de explicar la varianza de los datos, comportándose incluso peor que una simple media.

Además, el gráfico de evaluación sobre el conjunto de test mostró un patrón preocupante: la mayoría de las predicciones estaban agrupadas en valores bajos, independientemente del valor real de la demanda. Las líneas con una alta carga de viajes actuaban como *outliers*, distorsionando el aprendizaje. La nube de puntos se acumulaba sistemáticamente por debajo de la recta $y = x$, lo que evidenciaba una clara infraestimación del modelo.

Entre los principales factores que limitaron el rendimiento del modelo se identificaron:

- **Representación de variables temporales inadecuada:** tanto la hora del día como el día de la semana se trataron como variables numéricas discretas, lo que dificultaba que la red capturara la naturaleza cíclica de estos atributos.
- **Distribución extremadamente sesgada de la variable objetivo:** un reducido número de rutas concentraba la mayoría de los viajes, generando un fuerte desbalance que favorecía la predicción de valores bajos.
- **Arquitectura simplificada:** la GNN utilizada presentaba una estructura básica sin mecanismos de regularización ni normalización, lo que podía inducir sobreajuste a patrones espurios o escasa capacidad de generalización.

- **Falta de features topológicos:** el modelo ignoraba propiedades relevantes como el grado del nodo o su conectividad, que podrían actuar como indicadores indirectos del volumen de tráfico.

Por tanto, se hizo necesario rediseñar el modelo, incorporando mejoras tanto en la construcción del dataset como en la arquitectura de la red neuronal, con el objetivo de abordar mejor la complejidad del problema y mitigar los efectos del sesgo en los datos.

2. Cambios implementados A partir del diagnóstico del modelo anterior, se desarrolló una versión significativamente mejorada, tanto en términos de ingeniería de características como de arquitectura del modelo, con el fin de dotarlo de mayor expresividad y capacidad de generalización.

Las principales mejoras introducidas fueron las siguientes:

- **Separación por franjas de demanda:** se implementó una estrategia de segmentación del dataset en tres subconjuntos —rutas de demanda *baja* (≤ 100 viajes), *media* (101–1000 viajes) y *alta* (> 1000 viajes)— y se entrenó un modelo independiente para cada franja. Esto permitió ajustar la capacidad del modelo a la escala y varianza específica de cada grupo, evitando que las rutas muy frecuentes distorsionasen el aprendizaje global.
- **Codificación cíclica de variables temporales:** para representar correctamente la naturaleza repetitiva de la hora y el día de la semana, se incorporaron las transformaciones seno y coseno (`sin/cos`) sobre ambas variables. Esta codificación permite al modelo capturar transiciones suaves entre, por ejemplo, la hora 23 y la hora 0, evitando disrupciones artificiales en el espacio de características.
- **Eliminación parcial de outliers:** se filtraron aquellas observaciones con demanda extrema (por encima del percentil 99) dentro de cada subconjunto, para evitar que afectasen de forma desproporcionada a la función de pérdida. Aunque no se eliminó completamente el sesgo, esta medida ayudó a estabilizar el entrenamiento y mejorar la calidad del ajuste, especialmente en la franja alta.
- **Normalización de atributos:** se aplicó `StandardScaler` tanto a las características de las aristas como a los atributos estructurales de los nodos, lo que mejoró el comportamiento numérico del modelo durante el entrenamiento.
- **Incorporación del grado del nodo como feature:** se añadió una nueva característica para cada nodo representando su grado (número de conexiones salientes), normalizado y utilizado como input al modelo. Este valor puede reflejar la importancia relativa de cada parada dentro de la red.
- **Arquitectura mejorada:** el modelo se estructuró como una Red Neuronal de Grafos con dos capas `GCNConv` para calcular los embeddings de nodo, seguidas de una red densa (`MLP`) que concatena las representaciones de los nodos origen y destino junto con los atributos de la arista. Esta red incluye regularización mediante `Dropout (0.2)` y normalización mediante `BatchNorm1d`, mejorando la estabilidad del entrenamiento y reduciendo el riesgo de sobreajuste.
- **Transformación logarítmica de la variable objetivo:** se mantuvo la transformación $\log(1+x)$ sobre el número de viajes, con el objetivo de suavizar la distribución de la demanda y reducir la influencia de valores extremos. La evaluación se realiza revirtiendo la transformación con $\exp(x) - 1$.

- **Curvas de pérdida y regresión por franja:** se implementó una visualización independiente para cada modelo, mostrando la evolución de la pérdida durante las 100 épocas de entrenamiento y un gráfico de dispersión predicho vs real. Estas gráficas permiten evaluar de forma más precisa cómo se comporta el modelo en cada franja, ayudando a detectar sesgos o regiones con alta varianza.

Estas mejoras buscaban dotar al modelo de una mayor capacidad para capturar patrones temporales y estructurales relevantes, controlar el efecto de los valores atípicos, y proporcionar una estimación más robusta del volumen de viajes entre pares de paradas, adaptada al nivel de tráfico observado en cada contexto.

Demanda baja El modelo entrenado sobre rutas de baja demanda mostró un comportamiento estable y una pérdida progresivamente decreciente:

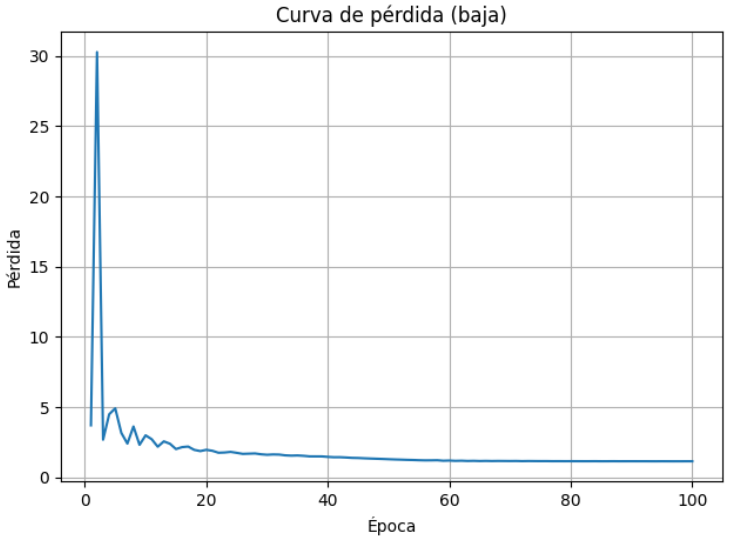


Figura 18: Curva de pérdida durante el entrenamiento – demanda baja.

En el conjunto de test, obtuvo:

- **MSE**: 432.97
- **MAE**: 12.04
- $\mathbf{R^2}$: –0.06

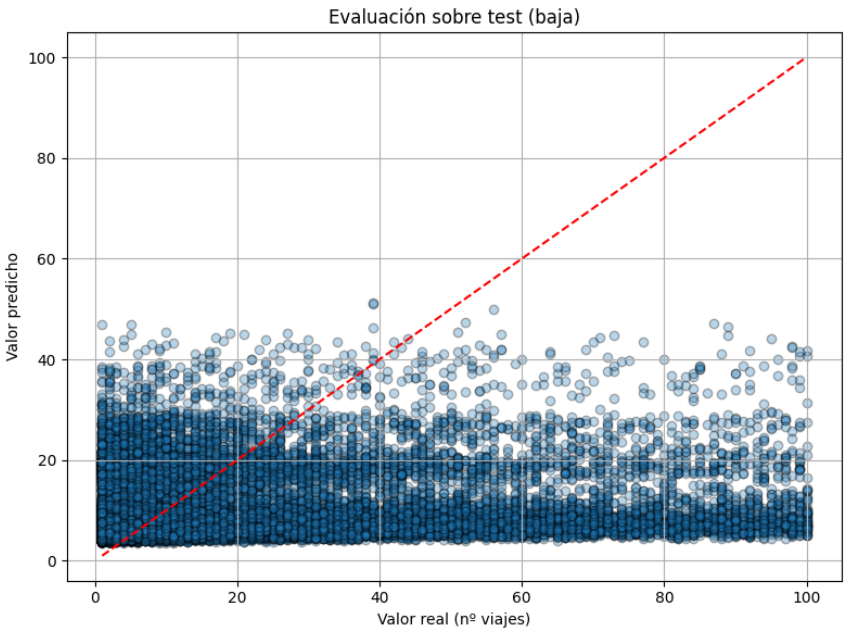


Figura 19: Evaluación sobre test – demanda baja.

Las predicciones son consistentes, aunque tienden a una ligera sobreestimación de los valores bajos, lo cual es razonable dada la escala del problema.

Demanda media La curva de pérdida del modelo de demanda media mostró más oscilaciones, señal de inestabilidad o sobreajuste parcial:

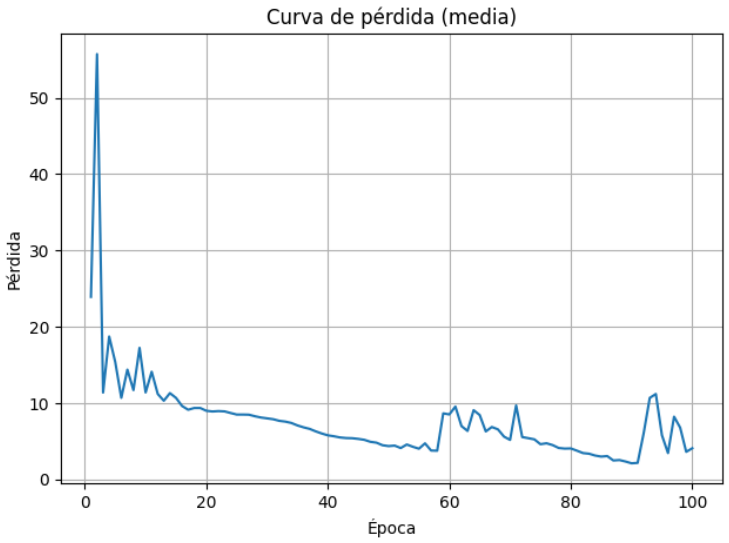


Figura 20: Curva de pérdida durante el entrenamiento – demanda media.

Métricas obtenidas sobre test:

- **MSE**: 66 283.71
- **MAE**: 203.13
- $\mathbf{R^2}$: –1.47

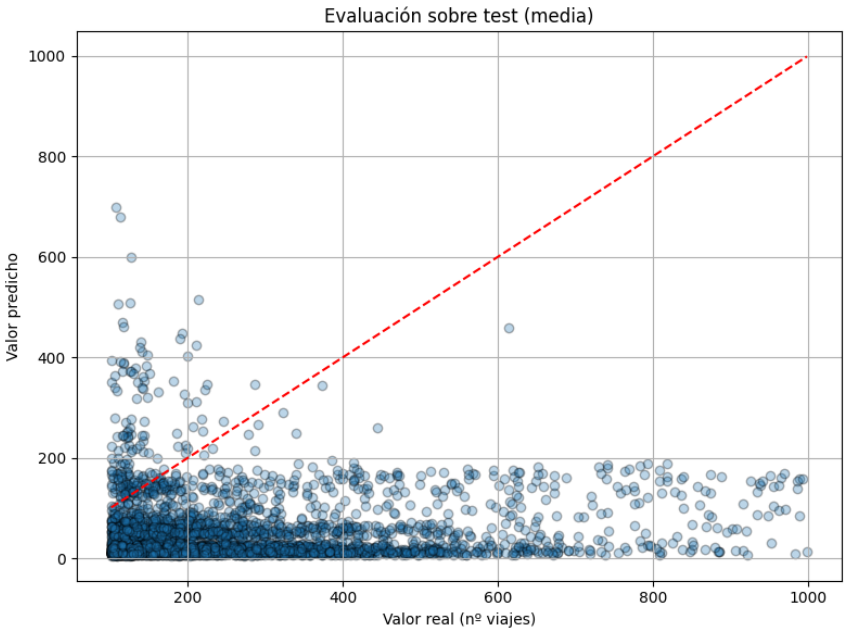


Figura 21: Evaluación sobre test – demanda media.

El modelo tiene dificultades para generalizar en este rango, con predicciones que divergen de los valores reales. La dispersión y varianza aumentan notablemente respecto a la franja baja.

Demanda alta El entrenamiento para la franja de alta demanda se comportó de forma más suave que en la media:

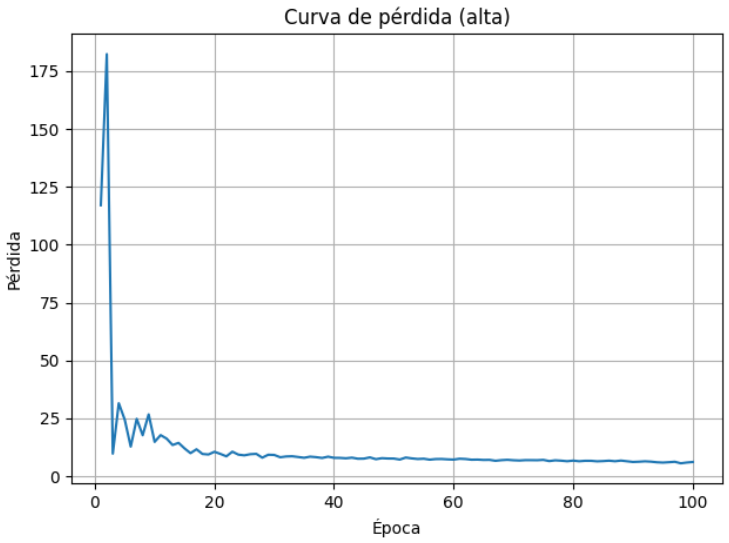


Figura 22: Curva de pérdida durante el entrenamiento – demanda alta.

Sin embargo, los resultados en test reflejan una baja capacidad de generalización:

- **MSE**: 732 515.19
- **MAE**: 723.16
- $\mathbf{R^2}$: –9.54

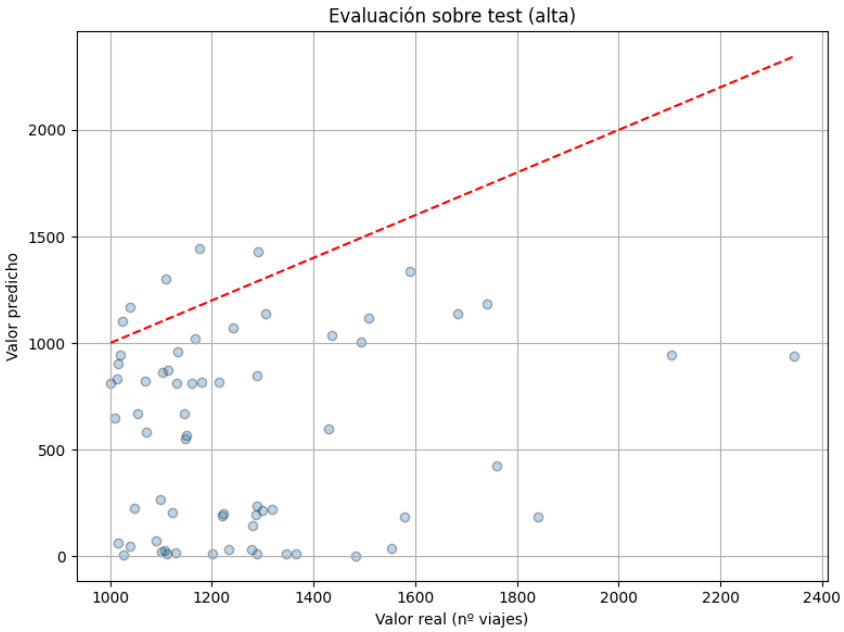


Figura 23: Evaluación sobre test – demanda alta.

Se observa una clara tendencia a la infraestimación en valores muy altos, así como una nube de predicciones aglutinadas en la zona baja. Esto indica que, incluso tras la transformación logarítmica, los valores extremos siguen afectando al aprendizaje.

Resumen general Para facilitar la comparación global, se generaron también las gráficas agregadas de pérdida y evaluación sobre todo el conjunto de test (sin separación por franja):

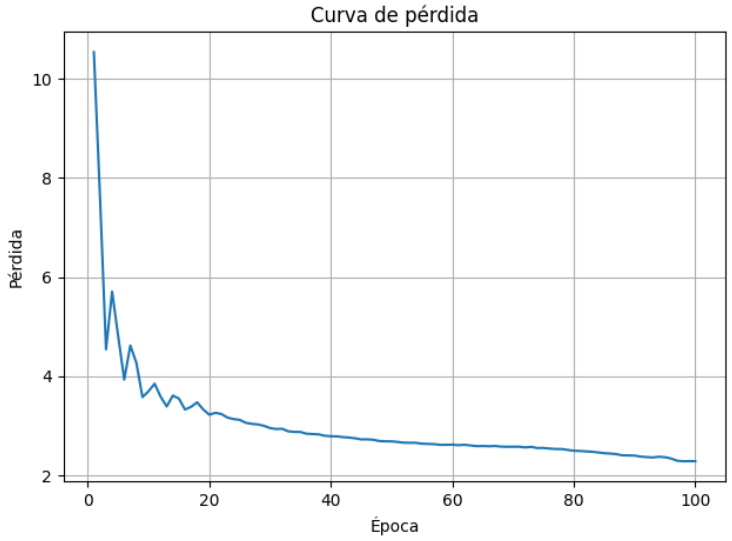


Figura 24: Curva de pérdida total (modelo general).

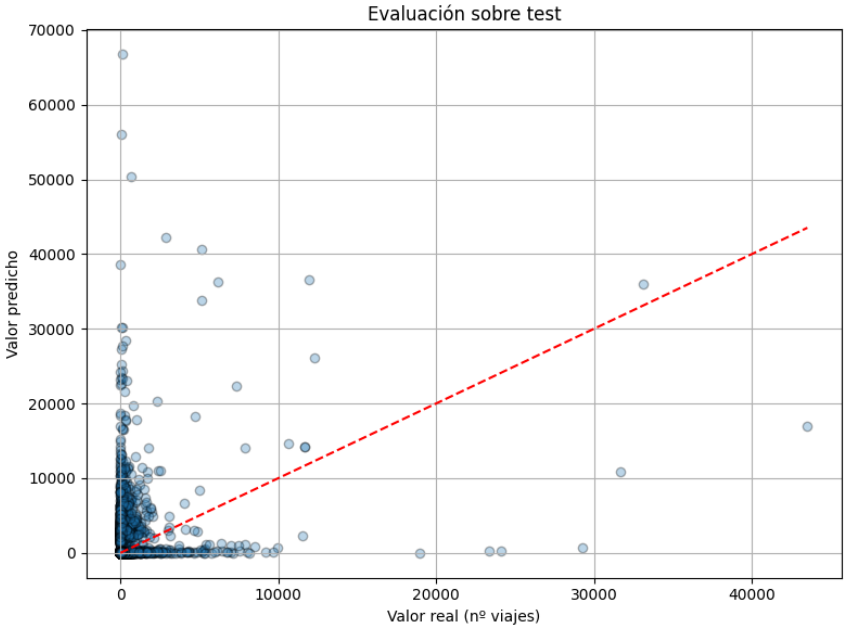


Figura 25: Evaluación sobre test – modelo general.

Como se puede observar, las predicciones del modelo general presentan un fuerte sesgo hacia los

valores bajos, incapaz de capturar la complejidad de la distribución de demanda real en los rangos altos. Este comportamiento refuerza la utilidad de aplicar modelos segmentados y con arquitecturas adaptadas a la escala del problema.

Conclusión.
El modelo de predicción de demanda presenta una mayor dificultad que el modelo de predicción de enlaces. El problema de regresión es más sensible al desbalanceo, a la estacionalidad horaria, y a la falta de features adicionales (como día de la semana, zona geográfica, etc.). Aunque el rendimiento no fue satisfactorio, se establecieron las bases para explorar arquitecturas más complejas (como GATs o MPNNs) y enriquecer las representaciones mediante codificaciones adicionales o embeddings preentrenados para líneas y puntos.

6. Conclusiones y líneas futuras

6.1. Conclusiones del trabajo

El presente Trabajo de Fin de Grado ha tenido como objetivo explorar el uso de redes neuronales sobre grafos (GNN) para tareas de predicción sobre datos reales de movilidad, aplicados al caso específico de trayectos de autobuses interurbanos en Brasil. A través de un enfoque iterativo y modular, se ha conseguido transformar un conjunto de datos en bruto, de gran volumen y escasa documentación, en una base estructurada y lista para aplicar técnicas avanzadas de modelado.

En una primera fase, se desarrollaron modelos de prueba (*toy models*) que permitieron familiarizarse con las arquitecturas GNN, los entornos de desarrollo y las librerías especializadas como `PyTorch Geometric`. Estos modelos sirvieron como base para comprender cómo representar y visualizar estructuras de red, así como para explorar tareas básicas como la clasificación de nodos o la predicción de enlaces en grafos sintéticos.

A continuación, se abordó el trabajo con datos reales, partiendo de ficheros de ventas de billetes de autobús en diferentes meses. La fase de limpieza e ingeniería de datos fue clave para seleccionar únicamente los registros válidos (viajes interestatales y regulares), así como para transformar los datos textuales en representaciones codificadas aptas para modelado. Se construyeron dos modelos principales:

- Un modelo de **predicción de enlaces**, que aprende a inferir si existe una conexión entre dos puntos dados, utilizando como entrada únicamente el grafo histórico de trayectos observados.
- Un modelo de **predicción de demanda**, que estima cuántos viajes se realizan entre un origen y un destino en una franja horaria concreta, utilizando una GNN de regresión sobre aristas.

Ambos modelos lograron completar su ciclo de desarrollo: construcción del grafo, entrenamiento del modelo, evaluación de resultados y visualización de predicciones. El modelo de predicción de enlaces obtuvo resultados sólidos, con valores de ROC AUC cercanos a 0.90, mostrando que la estructura del grafo contiene información útil para predecir trayectos no observados. En cambio, el modelo de predicción de demanda presentó una mayor dificultad, con errores de predicción significativos y bajo ajuste (R^2 cercano a 0), lo cual se ha interpretado como consecuencia de una alta dispersión en los datos y un conjunto de atributos poco expresivo.

En resumen, este trabajo ha demostrado que las GNN pueden aplicarse de forma efectiva a problemas de movilidad, y que incluso con una estructura de datos mínima es posible obtener resultados significativos en tareas de predicción estructural. Además, ha permitido experimentar con flujos

completos de procesamiento de datos reales, desde la limpieza hasta la visualización avanzada de resultados, consolidando tanto los conocimientos técnicos en inteligencia artificial como la capacidad de análisis crítico sobre los resultados obtenidos.

6.2. Posibles mejoras y extensiones

A pesar del progreso alcanzado en este trabajo, existen múltiples líneas de mejora y extensión que permitirían aumentar tanto la calidad de los modelos como su aplicabilidad práctica. A continuación se detallan las más relevantes:

- **Incorporación de más atributos como input del modelo:** actualmente, tanto los nodos como las aristas del grafo tienen una representación muy limitada. Se podrían añadir características adicionales como el día de la semana, si el viaje corresponde a una festividad o fin de semana, el operador de la línea, o incluso atributos climáticos, que podrían tener un impacto significativo en la demanda de viajes.
- **Enriquecimiento de los nodos:** una limitación clara es que los nodos (puntos de origen y destino) carecen de embeddings ricos o semántica más allá de su identificación numérica. Incorporar embeddings geográficos, categorizaciones urbanas (terminal, parada rural, etc.) o agrupaciones en zonas podría mejorar notablemente la capacidad predictiva del modelo.
- **Refinamiento del modelo de regresión de demanda:** debido al pobre desempeño del modelo actual, sería interesante explorar arquitecturas alternativas (por ejemplo, Graph Attention Networks), aplicar técnicas de balanceo, o incluso transformar el problema en una clasificación por rangos de demanda para evitar el sesgo de valores extremos.
- **Evaluación en más escenarios temporales:** actualmente se ha utilizado una muestra mensual de datos. Evaluar los modelos en diferentes meses o realizar un entrenamiento en un conjunto temporal y prueba en otro permitiría validar mejor la capacidad generalizadora del enfoque.
- **Integración con herramientas de Business Intelligence (BI):** una evolución natural del trabajo consistiría en exponer los resultados de los modelos (predicciones de enlaces o estimaciones de demanda) a través de plataformas como Looker Studio o Power BI. Esta integración permitiría a usuarios no técnicos interpretar los patrones de movilidad de forma visual e interactiva.
- **Implementación como servicio en tiempo real:** a medio plazo, se podría desarrollar una API REST que reciba una consulta de origen, destino, línea y hora, y devuelva la predicción del modelo. Esto permitiría integrar el sistema en aplicaciones reales de planificación de rutas o dimensionamiento de flotas.
- **Escalabilidad y uso de librerías más eficientes:** dado que los datos originales tienen millones de registros, una futura extensión podría trabajar directamente con ellos haciendo uso de bases de datos orientadas a grafos (como Neo4j) o frameworks distribuidos (como DGL o PyTorch Geometric Temporal).

- **Validación con datasets oficiales de movilidad:** por último, se podría repetir el enfoque propuesto usando datos abiertos de transporte público de ciudades europeas o latinoamericanas que cuentan con APIs GTFS y datos validados por organismos oficiales, aumentando la confiabilidad del estudio.

Estas mejoras no solo permitirían incrementar la precisión de los modelos desarrollados, sino también posicionar este trabajo como una base sólida para futuras investigaciones, aplicaciones reales en el sector transporte o incluso colaboraciones con organismos públicos y empresas de movilidad.

Parte III - Apéndices

7. Manual de uso y ejecución

7.1. Instalación del entorno virtual

Para garantizar la correcta ejecución del código desarrollado en este proyecto y evitar conflictos de versiones con otras instalaciones de Python, se ha utilizado un entorno virtual específico, creado y gestionado con la herramienta estándar `venv`. Este entorno proporciona un espacio aislado en el que se pueden instalar todas las dependencias necesarias sin afectar al sistema global.

A continuación se detallan los pasos necesarios para obtener el código fuente del proyecto, crear y activar el entorno, e instalar las librerías requeridas:

1. Clonación del repositorio El primer paso consiste en descargar el código del proyecto desde el repositorio de GitHub. Para ello, se debe utilizar el siguiente comando en la terminal:

```
git clone https://github.com/ruperame/gnntoymodel
```

Este comando creará una nueva carpeta con el nombre del repositorio en el directorio actual. A continuación, se debe acceder a dicha carpeta:

```
cd repositorio
```

2. Requisitos previos Es necesario tener instalada una versión reciente de Python 3 (preferiblemente Python 3.9 o superior). Para comprobar la instalación se puede utilizar el siguiente comando:

```
python --version
```

En caso de disponer de varias versiones de Python en el sistema, puede utilizarse el comando `python3` en su lugar.

3. Creación del entorno virtual Desde la carpeta raíz del proyecto, ejecutar el siguiente comando para crear un entorno virtual:

```
python -m venv gnn-tfg
```

Esto generará una carpeta llamada `gnn-tfg` que contendrá un entorno Python aislado con su propio intérprete y librerías.

4. Activación del entorno La activación del entorno varía según el sistema operativo:

- En sistemas Unix/Linux o macOS:

  ```
  source gnn-tfg/bin/activate
  ```

- En sistemas Windows (con CMD):

  ```
  gnn-tfg\Scripts\activate
  ```

- En sistemas Windows (con PowerShell):

```
.\gnn-tfg\Scripts\Activate.ps1
```

Una vez activado, el nombre del entorno aparecerá al inicio de la línea de comandos, indicando que se está trabajando dentro del entorno virtual.

5. Instalación de dependencias Con el entorno virtual activado, se deben instalar las dependencias necesarias mediante el archivo `requirements.txt`:

```
pip install -r requirements.txt
```

Entre las principales librerías incluidas en este archivo se encuentran:

- `networkx` para la creación y manipulación de grafos.
- `torch` y `torch-geometric` para el desarrollo y entrenamiento de Graph Neural Networks.
- `matplotlib` y `seaborn` para la generación de gráficos.
- `numpy` y `pandas` para el tratamiento eficiente de datos.

6. Verificación Para comprobar que todo está correctamente configurado, se puede ejecutar un script principal del proyecto, por ejemplo:

```
python src/toy_graph.py
```

Si la instalación ha sido exitosa, el script se ejecutará sin errores y comenzará a generar los resultados definidos en la configuración del sistema.

7. Desactivación del entorno Una vez finalizada la sesión de trabajo, se puede salir del entorno virtual con el comando:

```
deactivate
```

Este procedimiento asegura que el proyecto sea fácilmente reproducible en otros equipos o entornos, y que el entorno de desarrollo se mantenga limpio y controlado.

7.2. Ejecución del código y scripts

Una vez instalado y activado el entorno virtual del proyecto, el sistema puede ejecutarse mediante una serie de scripts desarrollados en Python, organizados en módulos específicos. Estos scripts permiten reproducir de forma completa tanto los experimentos realizados con *toy models* como los modelos aplicados sobre datos reales del sistema de autobuses de Brasil. La estructura modular facilita la experimentación y la ampliación del proyecto.

Modelos disponibles:

- `toy_graph.py`: script principal para generación y análisis de modelos sintéticos (*toy models*).

- `link_prediction.py`: predicción de enlaces entre paradas de autobús (modelo real).
- `demand_prediction.py`: predicción de carga de viajes entre pares origen-destino (modelo real).

Cada uno de estos scripts puede ejecutarse de forma independiente desde la terminal o integrarse en notebooks para su uso exploratorio. A continuación se describe su funcionamiento.

1. Toy Models: `src/toy_graph.py` Este script encapsula el flujo completo de trabajo sobre modelos sintéticos, permitiendo configurar múltiples franjas horarias y versiones del modelo para simular distintos escenarios de movilidad urbana. Las etapas clave son:

- Definición de parámetros globales: número de nodos, horas de simulación, versión del toy model.
- Generación del grafo y asignación de atributos (como codificación seno-coseno de la hora).
- Construcción del objeto `Data` para PyTorch Geometric.
- Entrenamiento del modelo GNN.
- Exportación de resultados y visualizaciones a `data/`.

Ejecución desde terminal:

```
python src/toy_graph.py
```

2. Modelo real 1: `gnn_models/link_prediction/link_prediction.py` Este script implementa un sistema de predicción de enlaces entre paradas reales. El objetivo es aprender si, dado un nodo de origen y uno de destino, existe (o debería existir) una conexión directa.

El flujo consiste en:

- Carga y limpieza de datos de autobuses de Brasil.
- Codificación de nodos mediante `LabelEncoder`.
- Generación del grafo de conexiones y muestreo de ejemplos negativos.
- Definición del modelo GCN, entrenamiento y evaluación.
- Visualización del subgrafo de predicciones con colores y etiquetas reales.

Para ejecutarlo:

```
python gnn_models/link_prediction/link_prediction.py
```

3. Modelo real 2: `gnn_models/demand_prediction/demand_prediction.py` Este script permite estimar cuántos viajes existen entre cada par origen-destino en una determinada hora y línea de autobús. Se trata de una tarea de regresión sobre aristas, útil para el análisis de demanda agregada.

Pasos principales:

- Preprocesamiento de datos: transformación de hora y fecha, agregación por número de viajes.
- Eliminación de valores atípicos extremos.
- Separación entre conjunto de entrenamiento y test.
- Normalización de atributos de arista (hora, línea).
- Entrenamiento y evaluación de un modelo GCN con MLP sobre aristas.
- Visualización dispersión real vs predicción.

Ejecución:

```
python gnn_models/demand_prediction/demand_prediction.py
```

4. Personalización y modificación de scripts Todos los scripts contienen bloques de configuración modificables directamente en el código:

- Número de épocas de entrenamiento.
- Capacidad del modelo (`hidden_channels`).
- Parámetros de visualización.
- Configuración de franjas horarias o modelos toy activos.

Además, cada modelo guarda sus resultados de forma organizada:

- `data/`: grafos `.pt`, encoders, y archivos auxiliares.
- `models/`: pesos entrenados de los modelos GNN.
- Visualizaciones interactivas o estáticas generadas con `matplotlib` y `networkx`.

5. Consideraciones adicionales Aunque la ejecución se ha pensado para entorno local, todos los scripts pueden ser adaptados para ejecución en servidores remotos o integrados en entornos como Google Colab o plataformas de orquestación como Apache Airflow. También es viable la futura implementación de una interfaz de línea de comandos para seleccionar modelo y parámetros sin necesidad de modificar el código.

En conjunto, el sistema desarrollado permite replicar todos los experimentos realizados en este trabajo fin de grado, facilitando además su modificación y extensión para nuevos contextos o datasets.

7.3. Visualización de resultados

7.4. Organización del proyecto y control de versiones

Durante el desarrollo del presente trabajo se ha seguido una estructura modular y organizada del repositorio, lo que ha permitido mantener una separación clara entre los diferentes bloques de código, datos y documentación. El proyecto se ha alojado en una plataforma de control de versiones (GitHub), lo cual ha sido esencial para gestionar cambios progresivos, experimentar con distintas variantes del modelo y documentar las decisiones tomadas.

Repositorio principal:

- URL: https://github.com/ruperame/gnntoymodel
- El repositorio contiene el código completo, datos y visualizaciones organizados en distintas ramas según el modelo y la versión utilizada.

Ramas disponibles:

- `main`: contiene el **toy model 1**, primer modelo funcional simple.
- `toy-model-2`, `toy-model-3`, `toy-model-4`: versiones mejoradas del modelo sintético con distintas franjas horarias, codificaciones temporales, o evaluaciones agregadas.
- `brasilBus`: modelo real sobre datos de autobuses de Brasil.
- `brasilBusMejoras`: versión mejorada del modelo anterior con entrenamiento por franjas de demanda y mayor robustez.

Nota: Puede aparecer un aviso como "`ignoring ref with broken name refs/heads/brasilBus 2`". Este mensaje se debe ignorar, ya que no afecta al uso del repositorio ni a las ramas relevantes.

Estructura de carpetas principal:

- `data/`: contiene los archivos de entrada y salida de los modelos, incluyendo los grafos en formato `.pt`, los `LabelEncoder` serializados, y el dataset limpio generado tras la fase de preprocesamiento.
- `gnn_models/`: incluye las implementaciones de los modelos desarrollados. Se divide en subcarpetas:
  - `link_prediction/`: contiene el modelo de predicción de enlaces y su `README`.
  - `demand_prediction/`: contiene el modelo de regresión de demanda y su `README`.
- `notebooks/`: se incluyen los notebooks exploratorios utilizados en la limpieza, visualización y análisis previo de los datos.
- `models/`: directorio de salida para guardar los pesos de los modelos entrenados.
- `README.md`: archivo de documentación principal del proyecto.

Control de versiones:
Se ha utilizado `Git` como sistema de control de versiones, con ramas específicas para cada bloque funcional del proyecto. Se ha seguido una estrategia de commits frecuentes y descriptivos para facilitar el seguimiento de cambios.
Algunos comandos utilizados durante el desarrollo han sido:

- `git clone https://github.com/ruperame/gnntoymodel` para descargar el repositorio.
- `git checkout <nombre_rama>` para cambiar entre modelos (toy-model-2, brasilBus, etc.).
- `git status`, `git add`, `git commit`, `git push` para versionado y actualización de cambios.
- `git rm / git restore` para revertir cambios o eliminar archivos del repositorio.

También se abordó un problema relacionado con archivos demasiado grandes (por ejemplo, ficheros `.csv` de más de 100 MB) que impedían realizar un `push` a GitHub. Para solucionarlo, se utilizó la herramienta `git filter-repo` para eliminar estos archivos del historial del repositorio, reduciendo el peso total y permitiendo la sincronización con el repositorio remoto.

Conclusión:
El uso riguroso de un sistema de control de versiones ha sido fundamental tanto para la trazabilidad como para el trabajo iterativo y reproducible del proyecto. Gracias a ello, fue posible integrar múltiples pruebas experimentales sin comprometer la estabilidad del repositorio principal, facilitando la evolución progresiva del trabajo.

Referencias

[1] Khemani, B., Patil, S., Kotecha, K. et al. (2024). *A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions*. J Big Data, 11, 18. https://doi.org/10.1186/s40537-023-00876-4

[2] Ward, I. R., Joyner, J., Lickfold, C., Guo, Y., Bennamoun, M. (2021). *A Practical Tutorial on Graph Neural Networks*. ACM Computing Surveys (CSUR), 54(10s), Article 205, 1–35. https://doi.org/10.1145/35030

[3] Wu, L., Cui, P., Pei, J., Zhao, L. (2022). *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer.

[4] Liu, Z., Zhou, J. (2020). *Introduction to Graph Neural Networks*. Morgan & Claypool Publishers. DOI: 10.2200/S00980ED1V01Y202001AIM045.

[5] Amal Menzli (2025). *Graph Neural Network and Some of GNN Applications: Everything You Need to Know*. https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications

[6] Adrien Payong (2023). *Graph Neural Networks: Fundamentals, Implementation, and Practical Uses*. https://blog.paperspace.com/graph-neural-networks-fundamentals-implementation-and-practical-uses/