COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

Recomendación y visualización de rutas turísticas personalizadas mediante aplicación de modelos de IA

Autor: David Tarrasa Puebla

Director: Pablo Sánchez Pérez

Madrid, Mayo 2026

Declaración de originalidad

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título **Recomendación y visualización de rutas turísticas personalizadas mediante aplicación de modelos de IA** en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico **2025/2026** es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Uso de Inteligencia Artificial[1]

Declaro bajo mi responsabilidad que (indicar la opción correcta):

☐ No he utilizado Inteligencia Artificial en la elaboración del presente documento.

☒ He utilizado Inteligencia Artificial en la elaboración del presente documento y/o del Anexo B siempre en las condiciones permitidas por la Universidad Pontificia Comillas, es decir, aplicando el Nivel 2 de la Escala de Evaluación de Perkins et al. (2024): *"La IA puede utilizarse para actividades previas a la tarea, como la lluvia de ideas, la descripción y la investigación inicial. Este nivel se centra en el uso de la IA para la planificación, las síntesis y la generación de ideas, pero las evaluaciones deben hacer hincapié en la capacidad de desarrollar y refinar estas ideas de forma independiente"*. En concreto, la Inteligencia Artificial ha sido empleada para:

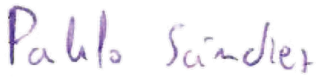
Se ha utilizado inteligencia artificial generativa como apoyo en tareas de planificación y revisión no sustantiva del documento, concretamente para lluvia de ideas, reorganización de apartados, síntesis inicial de información técnica y detección de inconsistencias internas del texto. La validación técnica del contenido, la verificación de los resultados experimentales, las decisiones de diseño del sistema, la elaboración de las figuras y tablas, y la redacción final del documento han sido realizadas por el autor.

[1]Esta declaración se refiere al uso de la Inteligencia Artificial generativa para realizar los documentos del Proyecto (Anexo B y Memoria). No aplica a Proyectos donde, por su naturaleza, deban emplear inteligencia artificial como parte de los mismos (aplicación de técnicas de aprendizaje automático, redes neuronales, análisis de datos...).

Firma del Autor

 (firma aquí)
Fdo: David Tarrasa Puebla
Fecha: 22/05/2026

Autorización para la entrega del Proyecto

El Director del Proyecto
 (firma aquí)
Fdo: Pablo Sánchez Pérez
Fecha: 22/05/2026

Agradecimientos

En primer lugar, quiero dar las gracias a mi tutor de TFG, Pablo Sánchez Pérez, por su disponibilidad, orientación y *feedback* continuo a lo largo del proyecto. Su experiencia en sistemas de recomendación y su ayuda para encauzar decisiones técnicas, especialmente en la parte de metodología y evaluación, han sido fundamentales para definir un trabajo sólido y bien planteado.

Quiero agradecer también a todo el profesorado del grado que me ha acompañado durante estos años de formación, ya que muchas de las herramientas y conceptos aplicados en este proyecto, como la ingeniería de datos, las bases de datos, el aprendizaje automático y la metodología científica, provienen directamente de lo aprendido en la carrera.

Por último, quiero dar las gracias a mi familia y a mis amigos por el apoyo constante, y a mis compañeros de promoción por haber compartido conmigo estos años de trabajo, proyectos y aprendizaje. Su ayuda y compañía han hecho esta etapa mucho más llevadera y motivadora.

UNIVERSIDAD PONTIFICIA COMILLAS
Escuela Técnica Superior de Ingeniería (ICAI)
Grado en Ingeniería Matemática e Inteligencia Artificial

Recomendación y visualización de rutas turísticas personalizadas mediante aplicación de modelos de IA

Autor: Tarrasa Puebla, David.
Director: Sánchez Pérez, Pablo.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este trabajo aborda el problema de la recomendación personalizada de rutas turísticas a partir de datos reales de visitas de usuarios y metadatos de puntos de interés (POIs, del inglés Points of Interest).

Palabras clave: sistemas de recomendación, rutas turísticas, puntos de interés, filtrado colaborativo, modelos secuenciales, evaluación offline, sistemas híbridos.

1. Introducción

La planificación de visitas en entornos urbanos constituye un problema relevante dentro del turismo inteligente y de los sistemas de recomendación contextuales. En este dominio, no basta con identificar lugares potencialmente interesantes para el usuario, sino que también es necesario organizarlos en una secuencia espacialmente razonable y útil desde un punto de vista práctico. Por ello, este TFG plantea una solución integral que combina recomendación de POIs, construcción de rutas y visualización interactiva.

2. Objetivos

El objetivo general del proyecto es desarrollar un recomendador de rutas turísticas capaz de generar propuestas útiles y personalizadas a partir de las señales disponibles en cada petición (historial de usuario, preferencias explícitas y contexto espacial). Para ello, se persiguen cuatro objetivos principales:

- Diseñar un modelo de datos que represente usuarios, trayectorias históricas, POIs y categorías asociadas.
- Implementar varios motores de recomendación complementarios que capturen señales semánticas, secuenciales y colaborativas.

- Extender la recomendación de POIs a la generación de rutas completas mediante mecanismos de selección, ordenación y control de coherencia espacial.
- Definir un protocolo de evaluación offline reproducible y desarrollar un prototipo funcional expuesto mediante API y aplicación web.

Además, el sistema distingue cuatro variantes principales de recomendación multi-ruta: `history` (historial previo del usuario), `inputs` (preferencias explícitas), `location` (proximidad al punto inicial) y `full` (combinación de todo lo disponible), lo que permite adaptar la salida a distintos escenarios de uso y niveles de información disponibles.

3. Descripción del modelo/sistema/herramienta

La solución propuesta se ha implementado como una arquitectura modular de extremo a extremo. En una primera capa, un proceso ETL prepara los datos y los carga en una base de datos PostgreSQL. Sobre esta base, el núcleo del sistema combina varios motores de recomendación. Modelos basados en contenido, co-visita, secuencia, embeddings y filtrado colaborativo implícito, junto con mecanismos de fusión y baselines de referencia. A partir de sus salidas, el sistema aplica un pipeline de scoring y reranking que incorpora restricciones de negocio y contexto, como preferencias explícitas, filtros de precio, diversidad por categoría y penalización por distancia. Posteriormente, los POIs seleccionados se transforman en una ruta ordenada mediante una heurística de *Nearest Neighbor*. La capa de producto se apoya en un backend FastAPI y un frontend ligero que permite configurar ciudad, preferencias, proximidad y usuario opcional, así como visualizar varias rutas en paralelo. De este modo, el proyecto no se limita a una validación experimental, sino que se concreta en un prototipo funcional y utilizable. Los datos de partida provienen del *Foursquare Semantic Trails Dataset* (2018), con aproximadamente 1,3 millones de check-ins reales distribuidos en tres ciudades: Osaka, Istanbul y Petaling Jaya (Selangor, Malasia), enriquecidos con metadatos de categoría, rating y coordenadas geográficas.

4. Resultados

La validación del sistema se ha llevado a cabo mediante evaluación offline y pruebas funcionales del prototipo. Para la evaluación de ranking se ha utilizado como protocolo principal `last_trail_user` (reserva el último trayecto de cada usuario como conjunto de prueba), bajo una configuración homogénea para todos los modelos. Las métricas consideradas incluyen *Hit@K*, *Precision@K*, *Recall@K* y *nDCG@K*; la evaluación se complementa con mediciones de *novelty* y *diversity*. Los resultados muestran que el baseline aleatorio obtiene el peor comportamiento, mientras que los modelos secuenciales, de co-visita e híbridos logran resultados claramente mejores. En particular, `hybrid` (fusión ponderada de modelos base) obtiene un *Hit@20* de 0.450 en Osaka y 0.424 en Petaling Jaya, frente a 0.000 del baseline aleatorio en ambas ciudades (resultado estadísticamente

esperable: el primer POI del trail se usa como semilla de entrada y queda excluido de la evaluación; el motor aleatorio sortea entre los miles de POIs no visitados por el usuario en entrenamiento, y con $k = 20$ la probabilidad de acertar alguno de los restantes ítems del trail es inferior al 1% por usuario). Los resultados sitúan a `hybrid` como el mejor motor global en *Hit@20* y *nDCG@20*, seguido muy de cerca por `rrf` (fusión de rankings mediante *Reciprocal Rank Fusion*). La evaluación de calidad de ruta confirma también que las rutas generadas mantienen una coherencia espacial razonable y una diversidad suficiente. El análisis por ciudad revela que en Istanbul, donde el volumen de datos es menor, el motor `markov` (modelo secuencial basado en transiciones entre POIs) supera al híbrido en varias métricas, ya que no depende de representaciones latentes y funciona mejor con datos escasos.

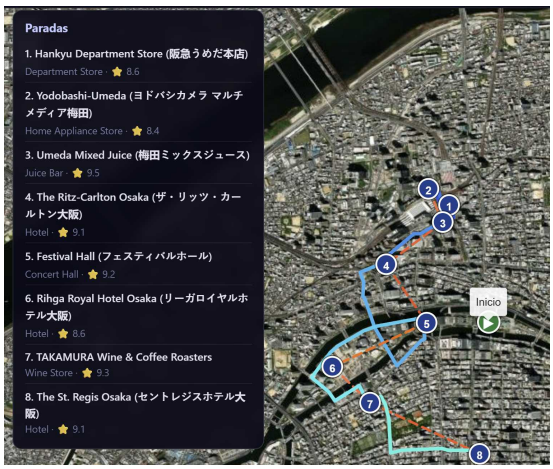


Figura 1: Ruta generada en Osaka sobre el prototipo web.

5. Conclusiones

Los resultados muestran que combinar señales heterogéneas dentro de una arquitectura híbrida ofrece el mejor comportamiento global en las ciudades con mayor cobertura de datos. La aportación central no es un algoritmo aislado, sino un sistema funcional completo: pipeline ETL, motores de recomendación, construcción de rutas, evaluación reproducible y prototipo web operativo sobre datos reales. El trabajo también deja ver que el verdadero reto de la recomendación turística no está solo en el ranking de POIs, sino en integrar esa señal dentro de un recorrido con sentido geográfico y temático. En ese frente, el sistema responde bien en los escenarios principales evaluados y deja abierta una base concreta sobre la que seguir iterando.

6. Referencias

Las referencias bibliográficas completas están en la bibliografía general de la memoria.

UNIVERSIDAD PONTIFICIA COMILLAS
Escuela Técnica Superior de Ingeniería (ICAI)
Grado en Ingeniería Matemática e Inteligencia Artificial

Recommendation and Visualization of Personalized Tourist Routes Using AI Models

Autor: Tarrasa Puebla, David.
Supervisor: Sánchez Pérez, Pablo.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This thesis addresses the problem of personalized tourism route recommendation from real user visit data and point-of-interest (POI) metadata.

Keywords: recommender systems, tourism routes, points of interest, collaborative filtering, sequential models, offline evaluation, hybrid systems.

1. Introduction

Urban visit planning is a relevant problem in intelligent tourism and context-aware recommender systems. In this domain, identifying potentially interesting places is not enough; they must also be organized into a spatially reasonable and practically useful sequence. Therefore, this thesis proposes an integrated solution that combines POI recommendation, route construction, and interactive visualization.

2. Objectives

The general objective of this thesis is to develop a tourism route recommender capable of generating useful and personalized suggestions from the signals available in each request (user history, explicit preferences, and spatial context). To achieve this, four main objectives are pursued:

- To design a data model representing users, historical trajectories, POIs, and associated categories.
- To implement complementary recommendation engines capturing semantic, sequential, and collaborative signals.
- To extend POI recommendation to full route generation through selection, ordering, and spatial coherence control mechanisms.
- To define a reproducible offline evaluation protocol and develop a functional prototype exposed through an API and a web application.

In addition, the system distinguishes four main multi-route variants: `history` (based on the user's prior visit history), `inputs` (driven by explicit user preferences), `location` (focused on proximity to the starting point), and `full` (combining all available signals), allowing adaptation to different usage scenarios and information levels.

3. Description of the proposed model/system/tool

The proposed solution is implemented as a modular end-to-end architecture. In a first layer, an ETL process prepares the data and loads it into PostgreSQL. On top of this, the system core combines several recommendation engines: content-based, co-visitation, sequential, embedding-based, and implicit collaborative filtering models, together with fusion mechanisms and reference baselines. From these outputs, the system applies a scoring and reranking pipeline that incorporates business and contextual constraints, such as explicit preferences, price filters, category diversity, and distance penalization. Selected POIs are then transformed into an ordered route using a *Nearest Neighbor* heuristic refined with *2-opt*, producing a final output that can be exported and visualized on a map. The product layer relies on a FastAPI backend and a lightweight frontend that allows users to configure city, preferences, proximity, and optional user identifier, and to visualize multiple routes in parallel. In this way, the project goes beyond experimental validation and materializes as a functional and usable prototype. The input data comes from the *Foursquare Semantic Trails Dataset* (2018), comprising approximately 1.3 million real check-ins across three cities: Osaka, Istanbul, and Petaling Jaya (Selangor, Malaysia), enriched with category metadata, ratings, and geographic coordinates.

4. Results

System validation was carried out through offline evaluation and functional prototype tests. Ranking evaluation uses `last_trail_user` (holds out each user's last trail as the test set) as the main protocol, under a homogeneous configuration for all models. Reported metrics include *Hit@K*, *Precision@K*, *Recall@K*, and *nDCG@K*, together with category-level measures, *novelty*, and *diversity*.

Results show that simpler engines are useful as references but have clear limitations. The random baseline performs worst, while sequential, co-visitation, and hybrid models achieve clearly better results. In particular, `hybrid` (weighted fusion of base models) reaches *Hit@20* values of 0.450 in Osaka and 0.424 in Petaling Jaya, versus 0.000 for the random baseline in both cities (a statistically expected result: the first trail POI is used as a seed and excluded from evaluation; the random engine draws from the thousands of POIs unseen during training, and with $k = 20$ the probability of hitting any of the remaining actual trail items is below 1% per user). In addition, the average over Osaka and Petaling Jaya places `hybrid` as the best global engine in *Hit@20* and *nDCG@20*, closely followed by `rrf` (ranking fusion via *Reciprocal Rank Fusion*); this average is re-

ported separately because Istanbul has lower volume and higher *sparsity*, which distorts aggregated comparison. Route quality evaluation also confirms that generated routes keep reasonable spatial coherence and sufficient diversity. City-level analysis reveals that in Istanbul, where data volume is lower, the `markov` engine (sequential model based on POI transitions) outperforms the hybrid on several metrics, as it does not rely on latent representations and performs better under data sparsity.

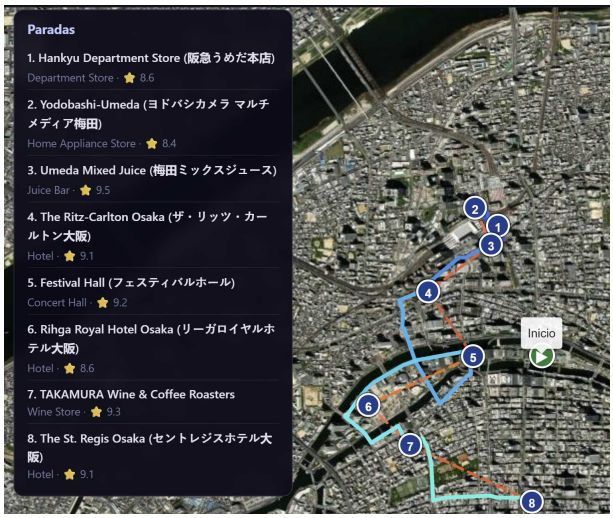


Figura 1: Route generated in Osaka on the web prototype.

5. Conclusions

Results indicate that combining heterogeneous signals within a hybrid architecture yields the best overall behaviour in cities with sufficient data coverage, while Istanbul illustrates how low interaction density reduces the effectiveness of latent models. The core contribution is not a standalone algorithm but a fully functional end-to-end system: ETL pipeline, recommendation engines, route construction, reproducible evaluation, and an operational web prototype built on real data. The work also makes clear that the real challenge in tourism recommendation lies not only in POI ranking but in integrating that signal into a geographically and thematically coherent itinerary. On that front, the system performs well across the main evaluated scenarios and leaves a concrete foundation for further development.

6. References

The complete bibliography is provided in the main references section of the thesis.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y motivación

La digitalización del turismo ha incrementado la disponibilidad de datos y servicios para apoyar la planificación de viajes y visitas urbanas. En este contexto, los sistemas de recomendación permiten reducir la sobrecarga de información y adaptar las sugerencias al perfil y al contexto del usuario [1, 2, 3].

En el ámbito turístico, la recomendación de puntos de interés (*Points of Interest*, POIs) es especialmente relevante, pero no suele ser suficiente por sí sola. El usuario no necesita solo una lista de lugares, sino una propuesta de visita que combine relevancia, coherencia espacial y variedad [3, 4, 5]. Por ello, la recomendación turística añade una dificultad adicional frente a otros dominios clásicos. Además de identificar lugares interesantes, debe organizarlos en una secuencia práctica y razonable dentro de una ciudad.

La motivación concreta nace de observar que la mayoría de las herramientas de recomendación turística se detienen en el listado: generan lugares potencialmente interesantes, pero no construyen un itinerario. El salto de *"aquí tienes diez POIs relevantes"* a *"aquí tienes un recorrido de cuatro horas que tiene sentido geográfico y temático"* no es trivial, y ese hueco es el que este trabajo intenta cubrir de principio a fin.

1.1.1 Problema abordado

En este contexto, el problema se define como la generación de rutas turísticas personalizadas a partir de historial de visitas y metadatos de puntos de interés. No se trata solo de seleccionar lugares relevantes, sino de ordenarlos en un itinerario que mantenga coherencia espacial, diversidad temática y utilidad práctica para el usuario final.

El problema combina varias dimensiones. Por un lado, exige estimar la relevancia de los puntos de interés para un usuario o contexto determinados. Por otro, requiere transformar esa recomendación en una secuencia de visita razonable, teniendo en cuenta factores como proximidad, diversidad temática y disponibilidad de información sobre el usuario. Por ello, este trabajo no se limita a un recomendador de ítems aislados, sino que plantea una solución orientada a la generación completa de rutas.

1.2 Objetivos del trabajo

1.2.1 Objetivo general

El objetivo general de este Trabajo Fin de Grado es diseñar, implementar y evaluar un sistema de recomendación de rutas turísticas personalizadas que, a partir de datos históricos de visitas y metadatos de puntos de interés, sea capaz de generar propuestas relevantes, coherentes desde el punto de vista espacial y útiles para distintos escenarios de uso [3, 4, 5].

1.2.2 Objetivos específicos

Para alcanzar este objetivo general, se plantean los siguientes objetivos específicos:

1. Diseñar un modelo de datos que permita representar usuarios, trayectorias históricas, puntos de interés y categorías asociadas de forma consistente.
2. Implementar varios motores de recomendación complementarios que capturen distintas señales del problema, incluyendo similitud de contenido, co-visita, patrones secuenciales y filtrado colaborativo implícito.
3. Desarrollar un enfoque híbrido capaz de combinar dichas señales de manera flexible según la información disponible en cada petición.
4. Extender la recomendación de puntos de interés a la construcción de rutas completas, incorporando mecanismos de selección, ordenación y control de coherencia espacial.
5. Definir un protocolo de evaluación offline homogéneo y reproducible que permita comparar los distintos modelos implementados con métricas de ranking y de calidad de ruta.
6. Exponer la funcionalidad del sistema mediante una API y una interfaz web que permitan explorar y visualizar las rutas generadas de forma interactiva.

1.2.3 Alcance y limitaciones del sistema

El proyecto cubre todo el flujo principal descrito en los objetivos anteriores: preparación y carga de datos, recomendación de puntos de interés, construcción y ordenación de rutas, evaluación offline y exposición mediante API y prototipo web. El sistema admite distintos escenarios de recomendación según la señal disponible: historial de usuario, preferencias explícitas, localización geográfica o combinación de ellas.

No obstante, el sistema presenta varias limitaciones. En primer lugar, opera actualmente sobre tres ciudades y depende de la cobertura y calidad de los datos disponibles, por lo

que los resultados no deben generalizarse automáticamente a otros entornos. En segundo lugar, la recomendación depende del nivel de información presente en cada petición, especialmente en casos de usuario frío (sin historial suficiente de interacciones) o con pocas señales. En tercer lugar, la construcción de rutas se basa en heurísticas prácticas y no en una optimización exacta bajo restricciones complejas como horarios, tráfico o meteorología. Por último, la validación realizada es offline y, aunque permite comparaciones rigurosas y reproducibles, no sustituye una evaluación con usuarios reales en un entorno de producción. El análisis detallado de estas y otras limitaciones identificadas durante el desarrollo se recoge en la Sección 5.2.

1.3 Alineación con los Objetivos de Desarrollo Sostenible (ODS)

El presente Trabajo Fin de Grado se alinea principalmente con varios de los Objetivos de Desarrollo Sostenible (ODS) definidos por Naciones Unidas, al proponer una solución tecnológica orientada a mejorar la recomendación y visualización de rutas turísticas personalizadas mediante técnicas de inteligencia artificial y análisis de datos.

En primer lugar, el trabajo se relaciona con el **ODS 9: Industria, innovación e infraestructura**, ya que desarrolla una arquitectura software que integra bases de datos, modelos de recomendación, evaluación reproducible y visualización interactiva. En este sentido, el proyecto constituye una aplicación de la innovación digital a un problema real del ámbito turístico.

En segundo lugar, el proyecto se vincula con el **ODS 11: Ciudades y comunidades sostenibles**. Aunque el sistema no actúa directamente sobre políticas urbanas, sí plantea una herramienta que puede favorecer una experiencia de visita más estructurada y contextualizada, facilitando recorridos más coherentes y potencialmente más eficientes dentro del entorno urbano.

Por último, el trabajo guarda relación con el **ODS 8: Trabajo decente y crecimiento económico**, dado que el turismo es un sector de gran relevancia económica en numerosas ciudades. Sistemas de recomendación más precisos y personalizados pueden contribuir indirectamente a mejorar la experiencia turística y a potenciar el descubrimiento de puntos de interés dentro de la oferta urbana.

En conjunto, la principal contribución del proyecto a los ODS se sitúa en el uso de la tecnología y la inteligencia artificial como herramientas para construir soluciones innovadoras, aplicables y potencialmente útiles en el contexto del turismo digital y de las ciudades inteligentes.

1.4 Estructura de la memoria

La memoria se organiza en seis capítulos principales, además de una sección de anexos.

En el **Capítulo 1** se presenta el contexto del problema, la motivación del trabajo, los objetivos perseguidos, su alineación con los Objetivos de Desarrollo Sostenible y la organización general de la memoria.

El **Capítulo 2** revisa el estado del arte en sistemas de recomendación, recomendación de puntos de interés, recomendación secuencial y generación de rutas turísticas, con especial atención a las técnicas, métricas y limitaciones más relevantes para el problema abordado.

En el **Capítulo 3** se describe el sistema desarrollado. Para ello se formaliza el problema, se presenta la arquitectura general de la solución, se detallan los datos utilizados, los motores de recomendación implementados, el proceso de construcción de rutas y los aspectos principales de la implementación. Asimismo, se recogen los elementos necesarios para la reproducibilidad y se describe el protocolo de evaluación empleado en la validación experimental.

El **Capítulo 4** presenta los resultados obtenidos. En él se analizan, en primer lugar, los resultados de ranking de los distintos motores de recomendación y, en segundo lugar, la calidad de las rutas generadas. Además, se incluyen análisis complementarios sobre escenarios *cold-start* y una comparación orientativa con trabajos de la literatura, junto con una síntesis crítica global de los resultados.

El **Capítulo 5** recoge las conclusiones del trabajo, sus principales aportaciones, las limitaciones identificadas durante el desarrollo y las líneas de trabajo futuro que se consideran más relevantes a partir de los resultados obtenidos.

El **Capítulo 6** corresponde a la bibliografía utilizada a lo largo de la memoria.

Finalmente, los **anexos** incluyen material complementario relacionado con la reproducibilidad, la configuración del sistema, la implementación, la API y resultados ampliados que, por extensión o nivel de detalle, no se incorporan en el cuerpo principal del documento.

2. Estado del arte

2.1 Formulación del problema y notación

En este trabajo, la recomendación de rutas turísticas se formula como un problema de ranking y selección secuencial sobre un conjunto de puntos de interés. Sean U el conjunto de usuarios y I el conjunto de puntos de interés (*Points of Interest*, POIs). Denotamos por $u \in U$ un usuario particular y por $i \in I$ un POI particular. Cada usuario u, cuando existe, dispone de un historial de visitas representado como una secuencia ordenada de POIs:

$$\mathbf{s}_u = (i_1, i_2, \ldots, i_T), \qquad i_t \in I.$$

El objetivo más básico del recomendador consiste en estimar una función de relevancia

$$r : U \times I \to \mathbb{R},$$

de forma que, para un usuario u, el sistema pueda producir un ranking de candidatos y seleccionar los K más relevantes:

$$\hat{\mathcal{R}}_u = \mathrm{TopK}_{i \in I \setminus I_u^{\mathrm{seen}}} r(u, i),$$

donde I_u^{seen} representa el conjunto de POIs ya visitados por el usuario.

Sin embargo, en el dominio turístico no basta con recomendar elementos relevantes de forma aislada. La salida deseada para el usuario u es una ruta ordenada de k POIs

$$\pi = (i_{\pi_1}, i_{\pi_2}, \ldots, i_{\pi_k}),$$

donde $\pi_j \in \{1, \ldots, |I|\}$ es el índice del j-ésimo POI en el catálogo e i_{π_j} el POI correspondiente; la ruta debe combinar relevancia, coherencia espacial y diversidad temática. De forma abstracta, este problema puede expresarse como una optimización sobre el conjunto de rutas factibles:

$$\pi^* = \arg\max_{\pi \in \Pi_k} \Big(\lambda_1 \,\mathrm{Rel}(\pi, u) - \lambda_2 \,\mathrm{Dist}(\pi) + \lambda_3 \,\mathrm{Div}(\pi) \Big),$$

donde Π_k es el conjunto de rutas candidatas de longitud k, $\mathrm{Rel}(\pi, u)$ mide la relevancia agregada de la ruta para el usuario, $\mathrm{Dist}(\pi)$ penaliza recorridos poco eficientes y $\mathrm{Div}(\pi)$ favorece variedad entre categorías.

Esta formulación permite entender el problema abordado en este TFG como una combinación de tres tareas complementarias: estimación de relevancia, modelado secuencial y construcción de rutas. Las secciones siguientes revisan las familias de modelos más relevantes para resolver cada una de estas dimensiones [3, 4, 5, 6].

2.2 Modelos de recomendación relevantes

Las familias de modelos más relevantes para este trabajo pueden entenderse como distintas formas de estimar la función de relevancia $r(u, i)$ introducida en la sección anterior. En términos generales, estas familias capturan señales complementarias del problema: similitud semántica entre POIs, patrones colectivos de visita, estructura secuencial de las trayectorias y afinidades latentes entre usuarios y lugares [3, 4, 5, 6].

Modelos basados en contenido. Los enfoques basados en contenido estiman la relevancia de un POI a partir de sus atributos descriptivos, como categorías, etiquetas o metadatos. Si $\mathbf{x}_i$ representa el vector de características del POI i y $\mathbf{p}_u$ el perfil del usuario, una formulación habitual consiste en medir la similitud coseno:

$$r_{\text{content}}(u, i) = \cos(\mathbf{p}_u, \mathbf{x}_i) = \frac{\mathbf{p}_u^\top \mathbf{x}_i}{\|\mathbf{p}_u\| \, \|\mathbf{x}_i\|}.$$

Este tipo de modelos resulta especialmente útil en escenarios con poca información colectiva o cuando la petición del usuario incluye preferencias explícitas. Su principal limitación es la sobreespecialización, ya que tienden a recomendar elementos muy parecidos a los ya conocidos [3, 7].

Modelos item-item y co-visita. Otra familia importante explota relaciones entre POIs a partir de su coocurrencia en trayectorias históricas. La idea es que dos lugares están relacionados si suelen aparecer juntos en visitas reales. Si $c(i, j)$ denota el número de trayectorias en las que aparecen conjuntamente i y j, una medida simple de asociación es

$$s(i, j) = \frac{c(i, j)}{\sqrt{c(i)\, c(j)}}.$$

Entonces, dado un conjunto de POIs ya observados por el usuario, la puntuación de un candidato puede obtenerse agregando sus similitudes con dichos elementos. Estos modelos capturan bien complementariedad de visita y relaciones funcionales entre lugares, incluso cuando no comparten categoría [3, 8].

Modelos secuenciales. En recomendación turística, el orden de visita contiene información relevante. Por ello, una parte importante de la literatura modela la probabilidad del siguiente POI condicionada por el contexto reciente. En su forma más simple, una cadena de Markov de primer orden asume:

$$P(i_{t+1} = j \mid i_t = i) = \frac{N(i \to j)}{\sum_{j'} N(i \to j')},$$

donde $N(i \to j)$ cuenta cuántas veces una transición entre i y j aparece en los datos. Este enfoque es interpretable y útil para capturar continuidad local, aunque simplifica el problema al depender solo del último estado [9, 10].

Una extensión natural consiste en aprender embeddings secuenciales sobre trayectorias, por ejemplo mediante Word2Vec [11]. Si una secuencia de POIs se interpreta como una "frase", el objetivo es maximizar la probabilidad de los contextos observados:

$$\max_{\theta} \sum_{t=1}^{T} \sum_{\substack{-w \le k \le w \\ k \neq 0}} \log p(i_{t+k} \mid i_t).$$

En esta expresión, T denota la longitud de la secuencia (trayectoria), w el tamaño de la ventana de contexto y k el desplazamiento relativo respecto al POI central i_t (con $k \neq 0$). De este modo, POIs que aparecen en contextos similares quedan próximos en el espacio vectorial, lo que permite capturar vecindad contextual más allá de las transiciones directas [3, 9, 11].

Filtrado colaborativo implícito. Dentro del filtrado colaborativo también existen enfoques basados en vecinos (k-NN), tanto usuario-usuario como ítem-ítem, que recomiendan por similitud directa en la matriz de interacciones [8]. Aunque son interpretables y útiles como referencia, suelen perder robustez en escenarios con alta dispersión o menor cobertura de datos.

Los modelos colaborativos buscan inferir afinidades usuario–POI a partir de patrones colectivos de interacción. En escenarios con feedback implícito, una formulación clásica consiste en aproximar la matriz usuario–ítem mediante factores latentes:

$$\hat{r}_{ui} = \mathbf{x}_u^\top \mathbf{y}_i,$$

donde $\mathbf{x}_u$ y $\mathbf{y}_i$ son las representaciones latentes del usuario y del POI (notación independiente de los vectores de características $\mathbf{x}_i$, $\mathbf{p}_u$ empleados en los modelos basados en contenido). En el caso de ALS (*Alternating Least Squares*, factorización matricial por mínimos cuadrados alternantes) para feedback implícito, el problema se expresa como

$$\min_{\mathbf{X},\mathbf{Y}} \sum_{u,i} c_{ui} \left(p_{ui} - \mathbf{x}_u^\top \mathbf{y}_i\right)^2 + \lambda \left(\sum_u \|\mathbf{x}_u\|^2 + \sum_i \|\mathbf{y}_i\|^2 \right),$$

donde p_{ui} representa preferencia binaria observada y c_{ui} el nivel de confianza asociado a esa observación. Estos modelos pueden ofrecer una personalización fuerte cuando existe suficiente historial, pero sufren en situaciones de *cold-start* o alta dispersión [3, 8, 12].

Modelos híbridos y fusión. Dado que ninguna familia por sí sola captura todas las dimensiones del problema, los enfoques híbridos combinan varias puntuaciones parciales. Una formulación genérica es

$$r_{\text{hyb}}(u,i) = \sum_{m=1}^{M} \alpha_m \, r_m(u,i), \qquad \sum_{m=1}^{M} \alpha_m = 1,$$

donde cada r_m corresponde a un motor distinto y α_m controla su peso relativo. En la práctica, este tipo de combinación permite integrar contenido, co-visita, secuencia, colaboración y contexto geográfico en una puntuación final más robusta [3, 5, 6].

En conjunto, estas familias de modelos muestran que la recomendación turística no puede abordarse de forma satisfactoria desde una única perspectiva. Por ello, los sistemas más competitivos en este dominio tienden a combinar varias señales y a complementar el ranking de POIs con fases posteriores de reranking y construcción de ruta [3, 4].

2.3 Recomendación de POIs, secuencias y rutas

La recomendación de puntos de interés presenta particularidades que la diferencian de otros problemas clásicos de recomendación. En este dominio, cada ítem está asociado a una localización física, una categoría semántica y, con frecuencia, a restricciones contextuales de uso. Por ello, la relevancia de un POI no depende solo de la afinidad usuario–ítem, sino también de factores como la proximidad, el contexto espacial y la relación con otros lugares visitados [3, 4, 5].

Componente geográfica. Una primera diferencia respecto a otros dominios es la presencia explícita de coordenadas geográficas. Si i y j son dos POIs con coordenadas $(\text{lat}_i, \text{lon}_i)$ y $(\text{lat}_j, \text{lon}_j)$, la distancia entre ambos puede modelarse mediante la fórmula de Haversine:

$$d(i,j) = \text{Haversine}\big((\text{lat}_i, \text{lon}_i), (\text{lat}_j, \text{lon}_j)\big).$$

Esta distancia permite introducir penalizaciones espaciales en la puntuación de recomendación. Por ejemplo, si ℓ representa una localización inicial conocida del usuario, una forma simple de incorporar proximidad es

$$r(u, i \mid \ell) = r_0(u,i) - \beta\, d(\ell, i),$$

donde $r_0(u,i)$ es la relevancia base estimada por el modelo y $\beta > 0$ controla el peso de la distancia. Este tipo de formulación resulta especialmente importante en escenarios *geo-first*, donde la utilidad práctica del POI depende de su cercanía [3, 4].

Componente secuencial. La recomendación turística tampoco puede entenderse solo como una tarea de ranking estático, ya que las visitas suelen formar trayectorias con un cierto orden. En consecuencia, una parte importante de la literatura modela secuencias de POIs en lugar de conjuntos no ordenados. Si la trayectoria de un usuario se representa como

$$\mathbf{s}_u = (i_1, i_2, \ldots, i_T),$$

el problema puede plantearse como una predicción del siguiente lugar, o bien como una recomendación condicionada por el contexto reciente:

$$r(u, i_t \mid i_1, \ldots, i_{t-1}).$$

Esta formulación justifica el uso de cadenas de Markov, embeddings secuenciales y modelos de recomendación secuencial, ya que el orden observado en los datos aporta información que no queda reflejada en modelos puramente agregados [3, 9, 10].

De ranking de POIs a construcción de rutas. Un aspecto central en turismo es que recuperar buenos POIs no garantiza obtener una buena ruta. Un conjunto de lugares puede ser relevante desde el punto de vista semántico y, aun así, generar un itinerario poco razonable si presenta saltos espaciales grandes o una secuencia mal organizada. Por ello, la literatura distingue entre dos problemas relacionados pero no equivalentes: la recomendación de POIs y la construcción de itinerarios [4, 5].

De forma simplificada, una ruta puede modelarse como una permutación ordenada de k POIs seleccionados:

$$\pi = (i_{\pi_1}, i_{\pi_2}, \ldots, i_{\pi_k}).$$

La calidad de dicha ruta puede depender simultáneamente de la relevancia de sus elementos, de la distancia total recorrida y de la diversidad entre categorías:

$$\pi^* = \arg\max_{\pi \in \Pi_k} \Big(\lambda_1 \,\mathrm{Rel}(\pi, u) - \lambda_2 \,\mathrm{Dist}(\pi) + \lambda_3 \,\mathrm{Div}(\pi) \Big).$$

Aquí, la longitud espacial de la ruta puede definirse como

$$\mathrm{Dist}(\pi) = \sum_{t=1}^{k-1} d(i_{\pi_t}, i_{\pi_{t+1}}),$$

mientras que una medida sencilla de diversidad categórica es

$$\mathrm{Div}(\pi) = \frac{|\{\mathrm{cat}(i) : i \in \pi\}|}{|\pi|}.$$

Estas expresiones no pretenden resolver exactamente el problema de optimización de rutas, pero sí muestran por qué la recomendación turística requiere integrar relevancia, estructura secuencial y coherencia geográfica en una misma formulación [3, 4, 5].

2.4 Evaluación y limitaciones del estado del arte

La evaluación en sistemas de recomendación se formula habitualmente como un problema de ranking. Dado un conjunto de elementos relevantes Rel_u para un usuario u y un ranking recomendado $Rec_u@K$, una de las métricas más utilizadas es

$$\text{Precision@K} = \frac{|Rel_u \cap Rec_u@K|}{K},$$

que mide la proporción de elementos relevantes dentro de las primeras K posiciones. De forma complementaria,

$$\text{Recall@K} = \frac{|Rel_u \cap Rec_u@K|}{|Rel_u|},$$

mide la fracción de elementos relevantes recuperados por el sistema. Cuando además interesa penalizar que los aciertos aparezcan en posiciones bajas del ranking, se emplea la familia DCG/nDCG. En particular,

$$\text{DCG@K} = \sum_{t=1}^{K} \frac{2^{rel_t} - 1}{\log_2(t+1)}, \qquad \text{nDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}},$$

donde IDCG@K representa el valor ideal de DCG@K. Estas métricas son especialmente adecuadas cuando el sistema produce una lista ordenada de candidatos y se desea evaluar no solo el acierto, sino también la calidad del ordenamiento [13, 14, 15].

En recomendación de puntos de interés, la coincidencia exacta con un POI concreto no siempre refleja por completo la utilidad de la recomendación. Distintos lugares de una misma categoría pueden resultar igualmente adecuados dentro de una experiencia turística similar. Por ello, la literatura propone complementar las métricas exactas con medidas semánticas a nivel de categoría, por ejemplo:

$$\text{CatPrecision@K} = \frac{|\text{Cat}(Rel_u) \cap \text{Cat}(Rec_u@K)|}{K},$$

donde $\text{Cat}(\cdot)$ denota el *conjunto* de categorías únicas presentes en los elementos relevantes o recomendados. Cada categoría coincidente se contabiliza como máximo una vez, independientemente de cuántos POIs de esa categoría aparezcan en la lista recomendada; el denominador K normaliza el valor respecto al total de posiciones disponibles en la recomendación. Este tipo de evaluación resulta especialmente útil en sistemas turísticos, donde parte del valor de la recomendación reside en acertar el tipo de experiencia más que un lugar exacto [3, 5].

Cuando el sistema produce rutas completas, la evaluación debe ir más allá del ranking de ítems. En este dominio también se consideran medidas agregadas relacionadas con el coste espacial del itinerario y la diversidad categórica, ya introducidas en la sección anterior. Estas medidas permiten valorar si la ruta final no solo contiene POIs relevantes, sino también si resulta recorrible y suficientemente variada [4, 5].

Desde una perspectiva crítica, el estado del arte muestra varias limitaciones recurrentes. En primer lugar, muchos trabajos se centran en una sola dimensión del problema, como la similitud semántica, la colaboración entre usuarios o la predicción secuencial, sin integrar de manera equilibrada relevancia, secuencia y coherencia geográfica. En segundo lugar, una parte importante de la literatura se orienta a recomendar el siguiente POI o a producir rankings de lugares, pero presta menos atención a la construcción de rutas completas realmente utilizables. En tercer lugar, la evaluación no siempre incorpora simultáneamente exactitud a nivel de POI, adecuación semántica y calidad del itinerario final [3, 4, 5, 13].

Estas limitaciones justifican el enfoque adoptado en este trabajo. En lugar de depender de un único modelo, resulta más adecuado combinar varias señales complementarias y añadir una fase específica de construcción de rutas. De este modo, la recomendación turística puede abordarse como un problema más amplio que la simple recuperación de ítems. Un problema en el que deben integrarse ranking, contexto secuencial, restricciones espaciales y criterios de utilidad práctica.

3. Sistema desarrollado

3.1 Planteamiento del problema

3.1.1 Definición del problema y modelado

Tal y como se estableció en la Sección 1.1.1, el sistema tiene como objetivo seleccionar POIs relevantes para una ciudad concreta y organizarlos en una secuencia de visita coherente. Esta tarea combina dos subtareas: estimación de relevancia para el usuario (recomendación de POIs) y ordenación espacialmente razonable de los candidatos seleccionados (construcción de ruta).

Desde un punto de vista funcional, la entrada puede incluir una ciudad objetivo, un identificador de usuario opcional, preferencias temáticas o restricciones simples y, en ciertos casos, una localización inicial. La salida es una ruta compuesta por varios puntos de interés ordenados, con la información necesaria para representarla y visualizarla.

El sistema se apoya en tres entidades principales: usuarios, trayectorias históricas de visita y puntos de interés. Los usuarios se representan a partir de su historial de visitas observado en el dataset. Dicho historial no se modela como un conjunto plano de interacciones, sino como una colección de trayectorias o *trails*, que preservan la dimensión secuencial de las visitas. Los puntos de interés constituyen la unidad básica de recomendación. Cada POI dispone de metadatos asociados, como nombre, coordenadas geográficas, rating, categoría principal y otras categorías secundarias. Esta información se utiliza tanto en la recomendación como en los filtros y en la evaluación.

3.1.2 Señales disponibles y escenarios de recomendación

El sistema admite varias señales de entrada, cuya disponibilidad depende de la petición. La primera es el historial del usuario, cuando existe información previa suficiente en la ciudad seleccionada. La segunda son las preferencias o restricciones explícitas, que reflejan la intención actual del usuario. La tercera es la localización geográfica inicial, útil para priorizar cercanía y coherencia espacial. Además, el sistema explota señales derivadas de los propios datos, como co-visita, secuencia, popularidad o afinidades latentes entre usuarios y POIs.

A partir de estas señales, el sistema distingue cuatro escenarios principales de recomendación. El primero está guiado por historial de usuario (`history`). El segundo está guiado por inputs explícitos de la petición (`inputs`). El tercero está dominado por localización geográfica (`location`). El cuarto combina todas las señales disponibles (`full`). Esta separación permite adaptar la recomendación a distintos niveles de información y, al mismo tiempo, hacer más interpretable el comportamiento del sistema. La lógica de activación se resume en la Figura 3.1.

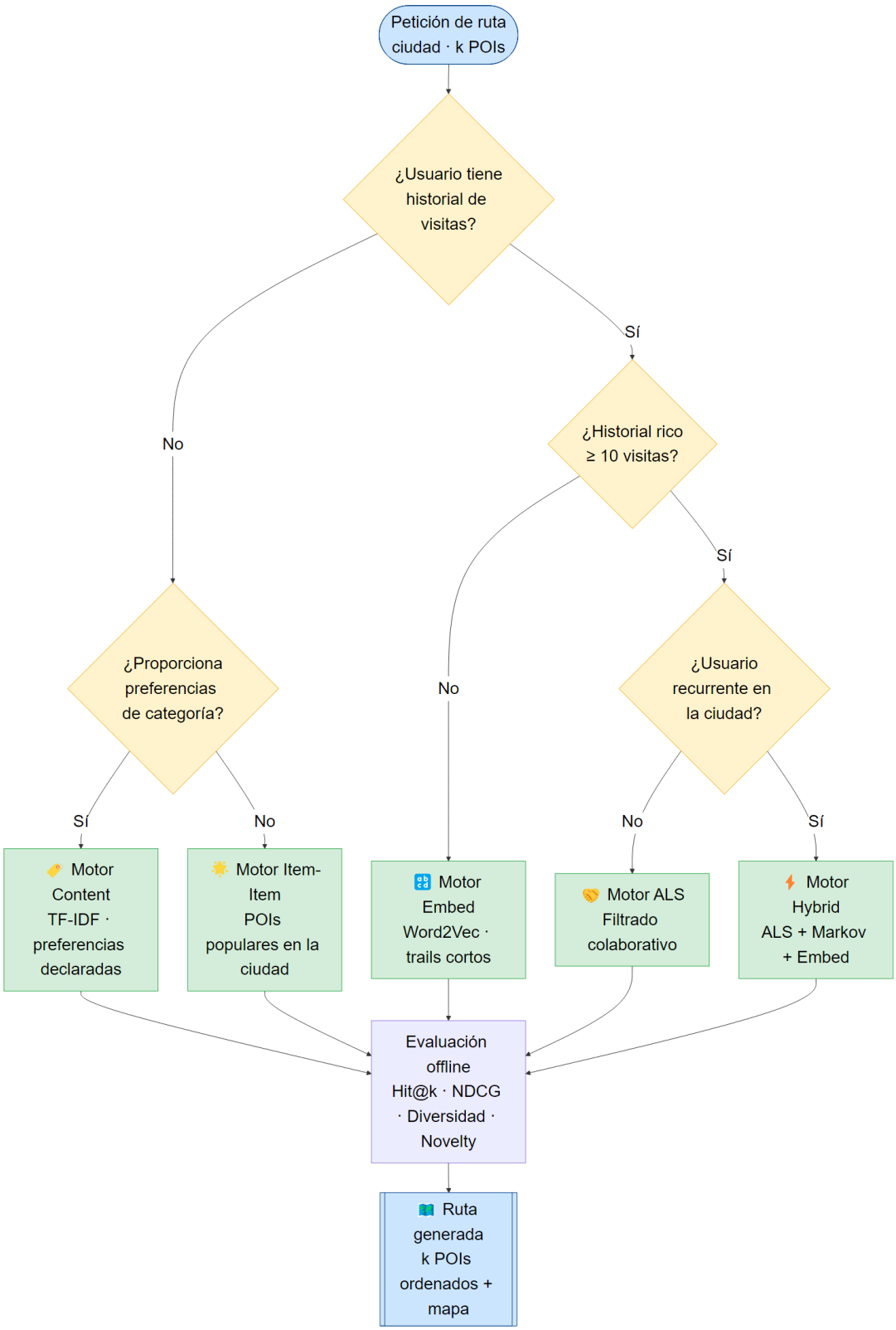


Figura 3.1: Árbol de decisión del sistema por tipo de usuario según las señales de entrada disponibles.

3.1.3 Restricciones del sistema y criterios de evaluación

El sistema debe cumplir varias restricciones prácticas. En primer lugar, la recomendación se limita a la ciudad seleccionada. En segundo lugar, debe respetar filtros asociados a precio, gratuidad y preferencias temáticas cuando estas señales están presentes. En tercer lugar, debe evitar, en la medida de lo posible, recomendaciones redundantes o espacialmente poco razonables. Además, cuando existe historial de usuario, la recomendación debe priorizar la novedad y evitar repetir lugares ya visitados, salvo en contextos concretos de evaluación. Finalmente, el sistema debe mantener un diseño reproducible, de forma que configuraciones, artefactos y resultados experimentales puedan relanzarse bajo condiciones controladas.

La validación del sistema se plantea en dos niveles. El primero corresponde a la evaluación de ranking de los puntos de interés recomendados, utilizando un protocolo offline reproducible y métricas como *Hit@K*, *Precision@K*, *Recall@K* y *nDCG@K*, junto con sus variantes a nivel de categoría. El segundo nivel corresponde a la evaluación de calidad de ruta, donde se analizan propiedades como distancia total, longitud media de tramos, diversidad temática y coherencia del itinerario generado. De esta forma, la evaluación no se limita a medir si el sistema recomienda POIs relevantes, sino también si es capaz de convertirlos en rutas razonables desde una perspectiva práctica.

3.2 Diseño de la solución

3.2.1 Arquitectura general

La solución propuesta se ha diseñado como un sistema integral de recomendación de rutas turísticas. En lugar de apoyarse en un único modelo, el sistema combina varios motores complementarios para capturar distintas señales del problema, como similitud semántica entre POIs, co-visita, secuencia de trayectorias y afinidades latentes entre usuarios y lugares. El enfoque adoptado responde a la naturaleza multi-señal del problema. En algunos casos, la recomendación debe apoyarse principalmente en el historial del usuario; en otros, en las preferencias explícitas o en la localización geográfica. Por ello, el sistema se ha planteado como una arquitectura flexible, capaz de adaptar su comportamiento a la información disponible en cada petición.

Desde un punto de vista funcional, la solución se organiza en tres pasos principales. En primer lugar, se estiman puntuaciones de relevancia para los POIs candidatos mediante distintos motores de recomendación. En segundo lugar, esas puntuaciones se combinan y refinan mediante un pipeline de scoring y reranking que incorpora restricciones semánticas y espaciales. En tercer lugar, los POIs seleccionados se transforman en una ruta ordenada y visualizable. Además, el sistema incorpora un esquema de respuesta multi-ruta (contrato de la API) que permite generar variantes guiadas por historial, inputs explícitos, localización o combinación de señales.

La arquitectura global del sistema se concibe como una estructura modular en capas, con

el objetivo de separar responsabilidades y facilitar tanto la reproducibilidad experimental como la evolución del proyecto. La primera capa corresponde a la preparación y carga de datos. La segunda contiene el núcleo del recomendador, formado por los motores de recomendación, el pipeline de scoring y reranking, y los módulos de construcción de rutas. La tercera está dedicada a la evaluación offline y benchmarking. La cuarta corresponde a la exposición del sistema mediante producto, a través de una API basada en FastAPI y un frontend web ligero. Esta organización modular permite desacoplar datos, inferencia, evaluación y presentación, y constituye una base adecuada para un sistema experimental que además debe demostrarse como prototipo funcional. El flujo de extremo a extremo se muestra en la Figura 3.2.

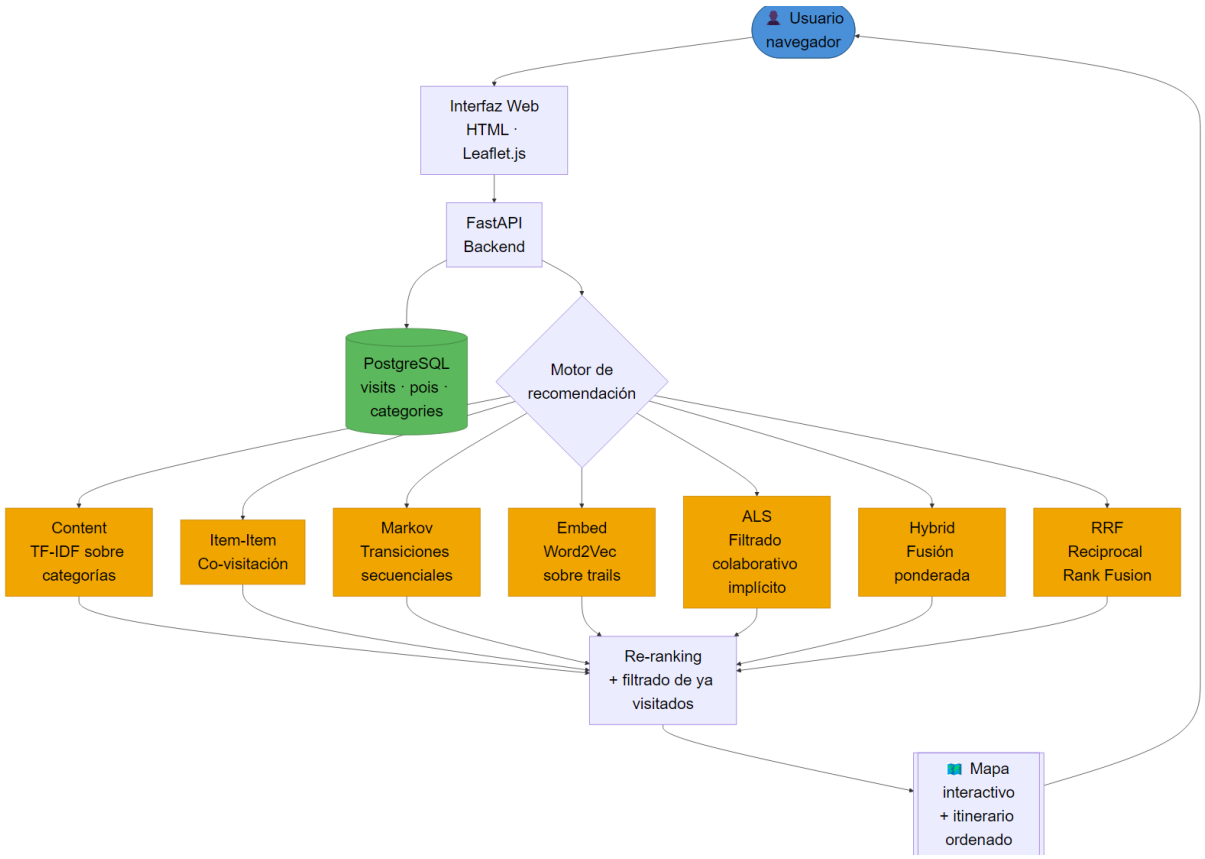


Figura 3.2: Flujo de una petición de recomendación a través de los componentes del sistema, desde la interfaz web hasta la devolución de la ruta generada.

3.2.2 Capa de datos

La capa de datos constituye la base del sistema. El proyecto parte de un conjunto de visitas históricas de usuarios y de un catálogo enriquecido de puntos de interés, que posteriormente se estructuran y cargan en PostgreSQL.

Los datos de partida provienen del *Foursquare Semantic Trails Dataset* (2018) [16], un conjunto de check-ins reales de usuarios en puntos de interés obtenidos a través de la

plataforma Foursquare[1]. El dataset cubre las tres ciudades del sistema (Osaka, Istanbul y Petaling Jaya) e incluye aproximadamente 1,3 millones de registros de visita. Cada check-in se enriquece con metadatos de categoría, rating, nombre y coordenadas geográficas procedentes de la Foursquare Places API. Este conjunto de datos aporta señales de movilidad real muy valiosas, aunque su cobertura no es homogénea. Osaka concentra la mayor densidad de registros, mientras que Istanbul y Petaling Jaya presentan una distribución más dispersa, lo que tiene implicaciones directas en el rendimiento de los motores de recomendación. La Figura 3.3 muestra el tamaño de cada burbuja proporcional a la raíz cuadrada del número de check-ins. Osaka concentra el mayor volumen, lo que explica que sea la ciudad de referencia en la evaluación experimental. La distribución espacial de POIs por ciudad se observa en la Figura 3.4, y el contraste entre oferta y demanda turística en Osaka en la Figura 3.5, que muestra las zonas de mayor densidad de visitas reales concentran principalmente POIs de Food, Transport y Shopping, lo que refleja el patrón turístico predominante en la ciudad.

Tabla 3.1: Estadísticas estructurales del *dataset* por ciudad (*Foursquare Semantic Trails Dataset*, 2018).

Ciudad	QID	Visitas	Usuarios	Trails	POIs	Trails/usuario	Densidad
Osaka	Q35765	675,807	25,467	97,697	16,481	3.84	0.0016
Istanbul	Q406	161,977	15,853	35,648	4,629	2.25	0.0022
Petaling Jaya	Q864965	460,584	18,346	82,951	18,616	4.52	0.0013

Nota: *POIs* indica el número de POIs únicos visitados por ciudad. La densidad se calcula como Visitas/(Usuarios × POIs).

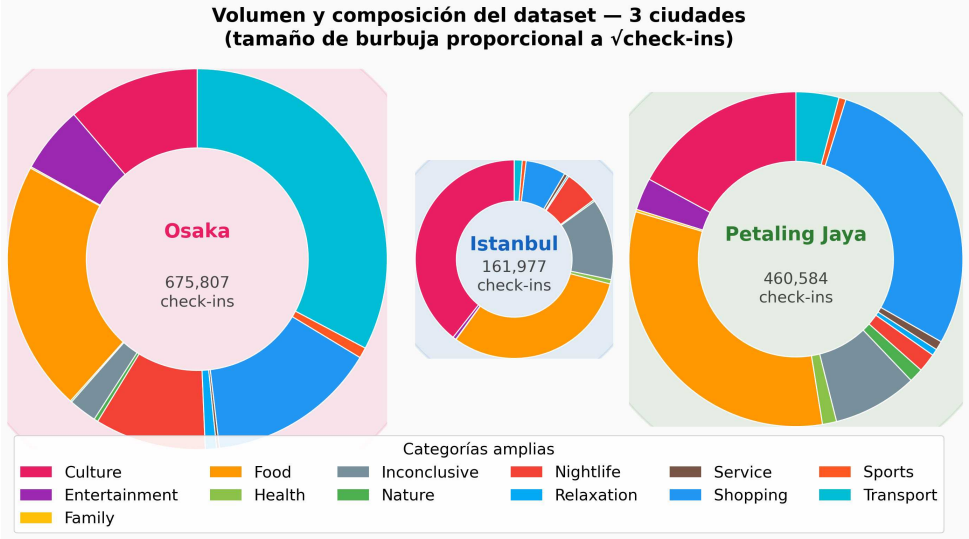


Figura 3.3: Comparación de volumen de datos y composición temática por categoría por ciudad.

[1] `https://developer.foursquare.com/`

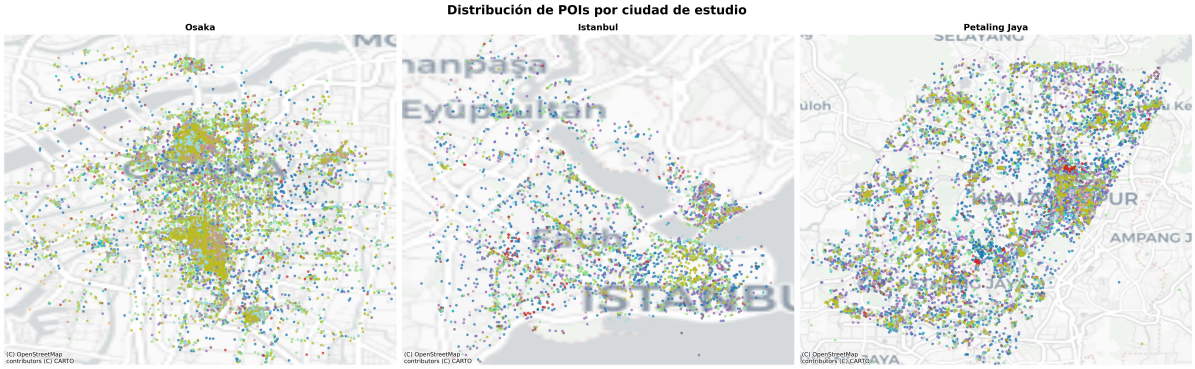
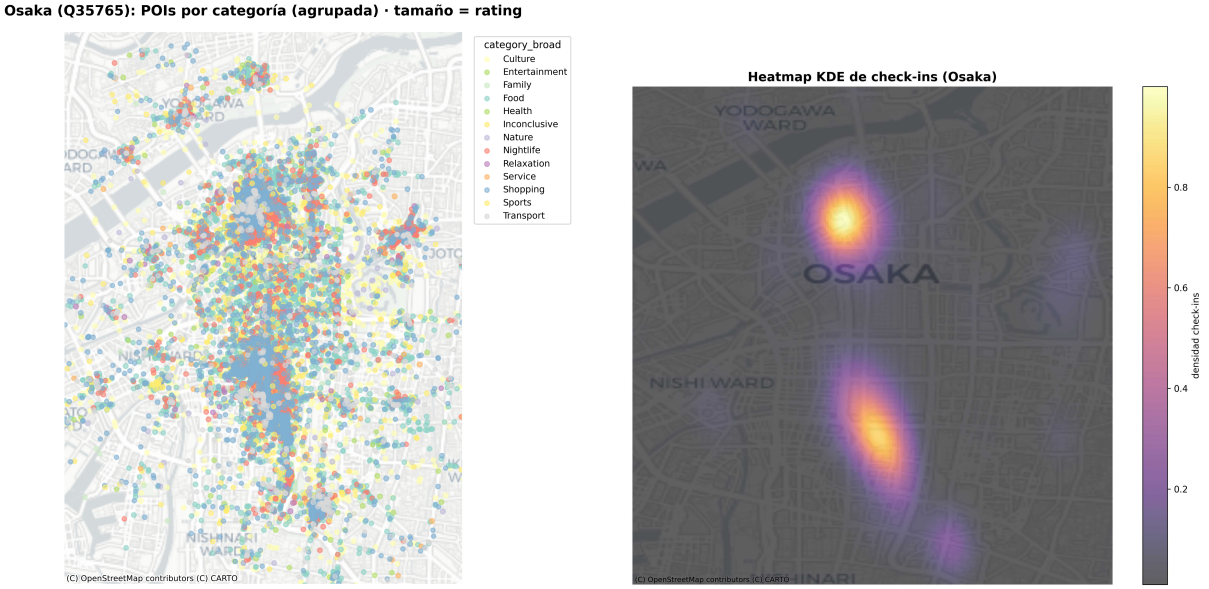


Figura 3.4: Distribución geográfica de los puntos de interés en las tres ciudades del sistema: Osaka, Istanbul y Petaling Jaya.



(a) Oferta: POIs de Osaka por categoría y rating.

(b) Demanda: densidad de check-ins reales en Osaka.

Figura 3.5: Oferta y demanda turística en Osaka.

El modelo de datos se apoya en cuatro tablas: `visits`, que recoge los eventos históricos de visita; `pois`, que almacena los metadatos principales de cada punto de interés; `poi_categories`, que contiene la taxonomía completa de categorías asociadas a cada POI; y `saved_routes`, que persiste las rutas generadas desde la capa de producto. Una decisión en esta capa es el uso de `city_qid` como identificador de ciudad, lo que permite filtrar los datos de forma robusta y mantener configuraciones, artefactos y evaluaciones separadas por ciudad. La preparación de esta información se apoya en un pipeline ETL que limpia, transforma y carga los datos en PostgreSQL. Su implementación concreta se describe en la Sección 3.3.2. El esquema relacional completo, con las relaciones entre tablas y sus claves se muestra en la Figura 3.6, y el flujo ETL en la Figura 3.7.

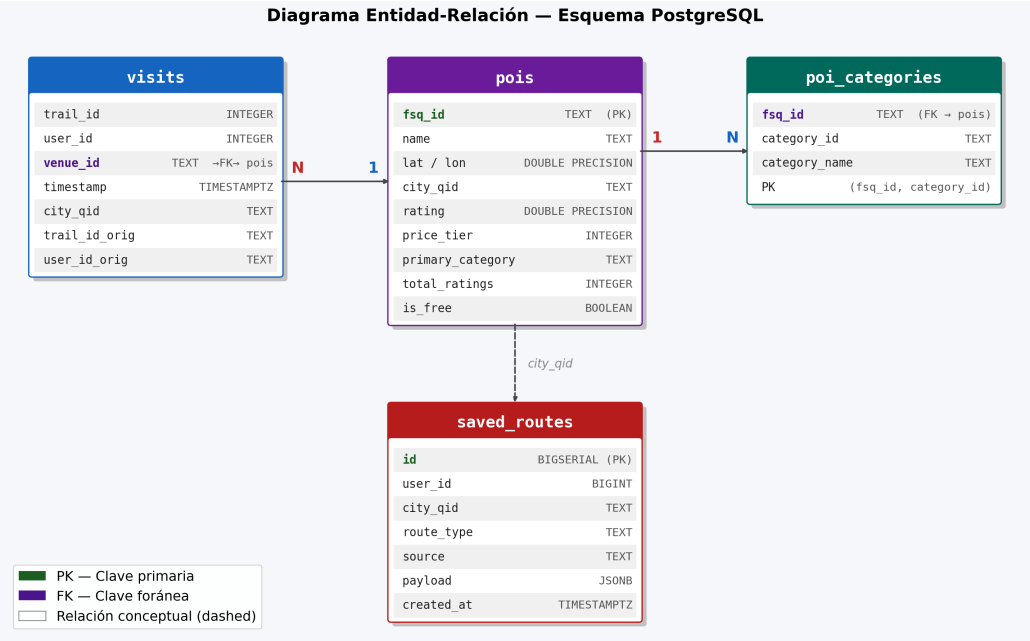


Figura 3.6: Diagrama entidad-relación del modelo de datos utilizado en el sistema.

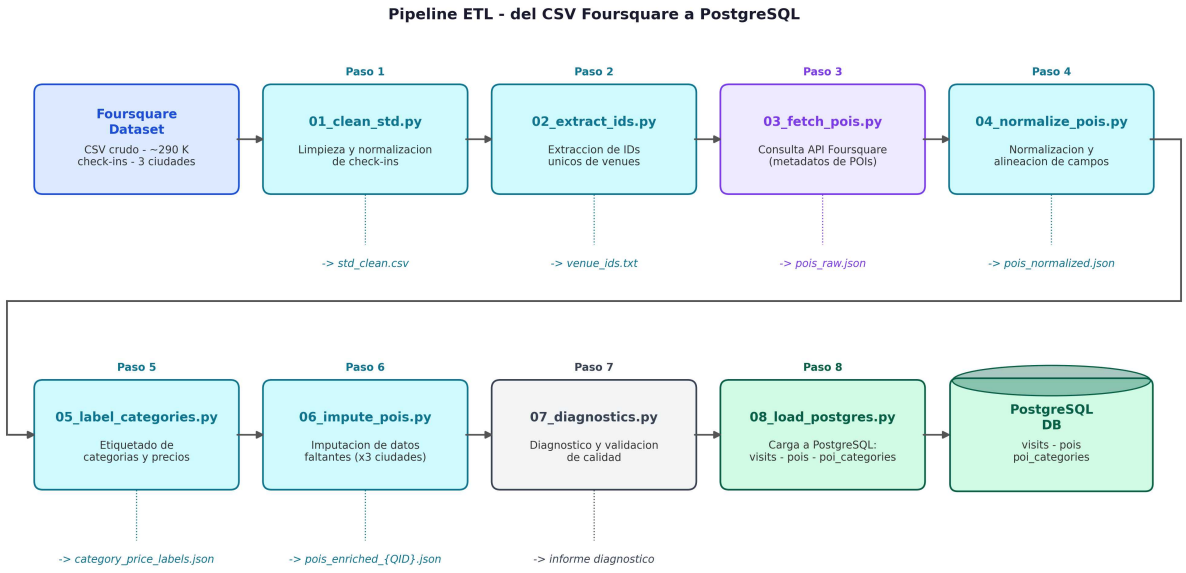


Figura 3.7: Pipeline ETL desde los datos procesados hasta la carga final en PostgreSQL.

El desglose conceptual de cada una de estas etapas se desarrolla en la Sección 3.3.2.

3.2.3 Motores de recomendación

El núcleo del sistema está formado por varios motores de recomendación complementarios. Esta decisión responde a la naturaleza heterogénea del problema. Distintas señales aportan información útil según el contexto, el tipo de usuario y la información disponible en cada petición. El motor `content` se basa en similitud semántica entre categorías y metadatos de los POIs. El motor `item` explota relaciones de co-visita entre lugares observadas en trayectorias históricas. El motor `markov` modela transiciones secuenciales entre puntos de interés mediante cadenas de Markov de orden 2. Dada la secuencia de visita (p_{t-1}, p_t), la probabilidad estimada del siguiente POI p_{t+1} se obtiene como:

$$\hat{P}(p_{t+1} \mid p_{t-1}, p_t) = (1-b)\, P_2(p_{t+1} \mid p_{t-1}, p_t) + b\, P_1(p_{t+1} \mid p_t),$$

donde P_2 son las probabilidades de transición de orden 2 observadas en los datos, P_1 las de orden 1 y b es el parámetro de *backoff* (ajustado por ciudad: $b = 0.30$ en Osaka, $b = 0.25$ en Istanbul y Petaling Jaya). Este parámetro se seleccionó por ciudad mediante barrido discreto en validación offline con protocolo `last_trail_user`, maximizando *nDCG@20* (con *MRR* como criterio de desempate) en `tune_markov.py`; el detalle completo se recoge en el Apéndice A.2. Cuando no existen transiciones observadas entre POIs individuales, el motor retrocede a una cadena de categorías. Recupera la distribución de categorías destino del POI actual y pondera cada candidato por la probabilidad de su categoría multiplicada por su rating normalizado. El motor `embed` utiliza embeddings secuenciales aprendidos sobre *trails*. Para calcular la puntuación de un candidato, no se limita al POI actual, sino que emplea los últimos N POIs visitados como contexto (parámetro `context_n`, ajustado por ciudad: 2 en Osaka, 1 en Istanbul, 3 en Petaling Jaya). Este valor se seleccionó mediante barrido conjunto con `topn_score` y `hub_alpha` en `tune_embeddings_scoring.py`, usando el mismo criterio de validación por ciudad. Esta elección proporciona una señal secuencial más rica y reduce la sensibilidad a errores en un único punto de referencia. El motor `als` implementa filtrado colaborativo implícito mediante factorización matricial siguiendo el enfoque de Hu et al. [12]. En lugar de valoraciones explícitas, trata la frecuencia de visita como señal de confianza y aprende representaciones latentes de usuarios y POIs que capturan afinidades difíciles de modelar con señales directas. Esto lo convierte en el motor de personalización más potente cuando existe suficiente historial, aunque su rendimiento decae en entornos con señal escasa, como Istanbul, donde la sparsidad de interacciones dificulta el aprendizaje de representaciones fiables. Sobre estos motores base, el sistema incorpora `hybrid`, que combina las puntuaciones de varios motores mediante una suma ponderada configurable por ciudad:

$$r_{\text{hyb}}(u,i) = \sum_m \alpha_m \cdot r_m(u,i), \quad \sum_m \alpha_m = 1 \tag{3.2.1}$$

donde $r_m(u,i)$ es la puntuación del motor m para el usuario u y el POI i. Antes de la combinación, cada motor normaliza sus puntuaciones al rango $[0,1]$ dividiéndolas por el máximo de la señal, lo que garantiza que ningún motor domine por escala y que los pesos sean comparables entre sí:

$$\hat{r}_m(u,i) = \frac{r_m(u,i)}{\max_j r_m(u,j)} \tag{3.2.2}$$

Los pesos α_m no son fijos. El sistema los selecciona automáticamente en función de las señales disponibles en cada petición. La Tabla 3.2 recoge los cinco escenarios contemplados, con el vector $[\alpha_{\text{content}}, \alpha_{\text{item}}, \alpha_{\text{markov}}, \alpha_{\text{embed}}, \alpha_{\text{als}}]$ correspondiente. Los pesos de `embed` y `als` se fijan a 0 cuando no está disponible su modelo entrenado por ciudad (es decir, los ficheros `word2vec_<city_qid>.joblib` y `als_<city_qid>.joblib`). En el escenario de *cold-start*, el motor de contenido recibe el mayor peso porque es el único capaz de operar sin historial ni POI de referencia.

Tabla 3.2: Pesos del motor híbrido por escenario de señales disponibles. El escenario `trail_current` (última fila) corresponde a construcción de ruta paso a paso; pesos por ciudad y detalles en la Sección B.8 del apéndice. Los escenarios restantes corresponden al modo `trail_user`, donde el motor agrega señales de historial completo, POI actual, modelos latentes o una combinación de ellas según el caso; en ausencia de toda señal se aplica la distribución de popularidad global (*cold-start*). Los valores mostrados corresponden a la configuración global por defecto; los ajustes específicos por ciudad se recogen en la Sección B.8 y en la Tabla A.4.

Escenario	**content**	**item**	**markov**	**embed**	**als**
Historial + POI actual	0.08	0.30	0.27	0.15	0.20
Solo historial de usuario	0.08	0.30	0.27	0.15	0.20
Solo POI actual (*current-only*)	0.10	0.15	0.35	0.05	0.35
Solo modelos latentes	0.15	0.15	0.10	0.15	0.45
Cold-start (sin señal)	0.60	0.20	0.20	0.00	0.00
Construcción de ruta (paso a paso)	0.00	0.29	0.39	0.20	0.13

Tanto el motor `embed` como el motor `markov` incorporan un parámetro configurable de penalización de hubs (`hub_alpha`) que, cuando está activo, reduce la puntuación de los POIs más frecuentados de forma proporcional a su popularidad global, favoreciendo recomendaciones menos predecibles. En la configuración de producción actual este parámetro está desactivado (`hub_alpha = 0`) en todos los entornos evaluados, pero puede ajustarse por ciudad si se detecta concentración excesiva en los POIs más visitados.

El motor `rrf` fusiona en cambio los rankings de posición mediante *Reciprocal Rank Fusion* [17], sin requerir calibración de escalas entre motores. Dado un conjunto de M rankings parciales, la puntuación de un candidato i es:

$$\text{RRF}(i) = \sum_{m=1}^{M} \frac{1}{k + \text{rank}_m(i)} \tag{3.2.3}$$

donde $\text{rank}_m(i)$ es la posición (1-indexada) de i en el ranking del motor m y $k = 30$ es la constante de suavizado empleada en el sistema, que atenúa el impacto excesivo de las primeras posiciones. Los candidatos ausentes en algún motor reciben una penalización equivalente a una posición muy baja. Además, se incluyen dos referencias experimentales simples: `popular`, basada en frecuencia global de visitas, y `random`, como baseline aleatorio. Cada uno de estos motores responde mejor a un escenario distinto. Los modelos basados en contenido son especialmente útiles cuando predominan las preferencias

explícitas. Los modelos secuenciales capturan mejor la continuidad entre visitas. Los modelos colaborativos ofrecen una personalización más fuerte cuando existe suficiente historial. Los enfoques híbridos, por su parte, permiten combinar estas señales y adaptarse mejor a escenarios mixtos.

En conjunto, esta familia de motores constituye la base algorítmica del sistema. A partir de sus salidas, el sistema aplica posteriormente un pipeline común de scoring, reranking y construcción de rutas para transformar puntuaciones parciales en una recomendación final interpretable y utilizable.

Las dinámicas secuenciales del motor Markov se ilustran en las Figuras 3.8 y 3.9. En el grafo de transiciones por categorías (Figura 3.8), Food, Transport y Shopping concentran el mayor número de conexiones y las probabilidades de transición más altas; Food presenta además la mayor probabilidad de auto-transición, reflejo del hábito turístico de encadenar varias visitas gastronómicas. El motor aprende y reproduce estos patrones directamente del histórico de visitas, lo que explica su coherencia secuencial en las rutas generadas.

La comparativa con rutas reales (Figura 3.9) refuerza esta lectura. La coincidencia geográfica entre los arcos del modelo y los flujos turísticos observados apoya la interpretación de que el motor captura los patrones de movilidad del dataset y los traslada fielmente a las recomendaciones. La proyección geográfica del grafo para las tres ciudades se muestra en la Figura 3.10. El tamaño de cada nodo es proporcional al número de visitas y el color refleja la categoría amplia. En esta visualización, Osaka presenta una red de transiciones claramente más densa que Istanbul, coherente con las diferencias de volumen de datos y rendimiento observadas en la evaluación.

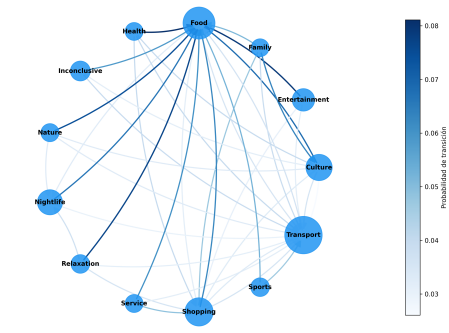


Figura 3.8: Grafo de transiciones Markov por categorías en Osaka.

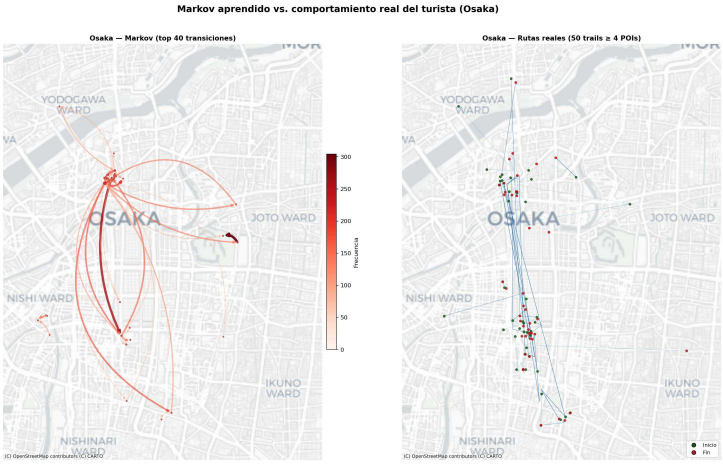


Figura 3.9: Transiciones del modelo Markov (izq.) frente a rutas reales del *dataset* (dcha.) en Osaka.

Figura 3.10: Grafo Markov geográfico para las tres ciudades del sistema.

3.2.4 Pipeline de inferencia y esquema de respuesta multi-ruta

Una vez definidos los motores de recomendación, el sistema integra sus salidas mediante un pipeline común de inferencia. Este pipeline construye primero el conjunto de candidatos dentro de la ciudad seleccionada, activa los motores disponibles según las señales presentes y los artefactos entrenados, y combina posteriormente sus puntuaciones en una representación unificada.

Sobre esta base, el sistema aplica una fase de scoring y reranking orientada a introducir restricciones prácticas del dominio. Entre ellas se incluyen distancia, filtros de precio o gratuidad, preferencias explícitas y controles de diversidad por categoría. Además, la capa de *intents* permite trabajar tanto en modo *soft*, reforzando preferencias, como en modo *strict*, filtrando candidatos por coincidencia semántica. El resultado de esta fase es una lista ordenada de POIs con puntuación y metadatos enriquecidos. Esta lista constituye la salida del scoring, pero no la ruta final. Para convertirla en un itinerario utilizable, el sistema aplica posteriormente una fase de selección y ordenación de ruta.

Uno de los elementos diferenciales del sistema es la incorporación de un esquema de respuesta multi-ruta (contrato de la API), diseñado para generar varias variantes de recomendación dentro de una misma petición. Esta decisión mejora la interpretabilidad del sistema y permite adaptar la salida a distintas señales de entrada disponibles. Este esquema de respuesta contempla cuatro variantes principales. La variante `history` se activa cuando el usuario dispone de historial en la ciudad seleccionada. La variante `inputs` se construye a partir de preferencias o filtros explícitos de la petición y es independiente del historial. La variante `location` se activa cuando se dispone de latitud y longitud. A diferencia de las demás variantes, aplica una penalización geográfica directa sobre las puntuaciones normalizadas del modelo. Cada candidato i recibe una puntuación que combina la señal del recomendador con la distancia al punto inicial:

$$s_{\text{loc}}(i) \;=\; \hat{s}(i) \cdot \left(1 - w_d \cdot \frac{d_{\text{anc}}(i)}{d_{\max}}\right) + \delta_{\text{cat}} \tag{3.2.4}$$

donde $\hat{s}(i) = s(i)/\max_j s(j)$ es la puntuación del modelo normalizada al rango $[0, 1]$, $d_{\text{anc}}(i)$ es la distancia en km al punto inicial, $d_{\max}$ es la distancia máxima observada entre el ancla y cualquier candidato, w_d es el peso de la penalización de distancia (configurable por ciudad), y δ_{cat} es una bonificación de diversidad que se aplica cuando el candidato aporta una categoría aún no presente en la ruta. La ruta se construye de forma greedy, añadiendo en cada paso el candidato con mayor s_{loc} entre los no seleccionados. Finalmente, la variante `full` combina las señales disponibles y produce una recomendación más completa y equilibrada. Adicionalmente, el sistema incorpora un mecanismo opcional de inyección de un POI de sorpresa suave (*soft surprise*) en la variante `full`. Con una probabilidad configurable (por defecto del 3 %), el sistema puede incluir en el itinerario final un candidato menos obvio que, siendo compatible con el contexto geográfico y semántico de la ruta, no habría sido seleccionado por puntuación directa. Para ser elegible, el POI sorpresa debe alcanzar al menos el 35 % de la puntuación del mejor candidato y estar a menos de 3 km del recorrido principal. Este mecanismo es intencionadamente de baja probabilidad y no interviene en la evaluación offline. Además, el sistema aplica una regla de validación clara. Si el usuario no tiene historial y la petición no incluye ni inputs ni localización, no existe señal suficiente para construir la recomendación y la solicitud debe fallar. Esta lógica refuerza la consistencia del sistema y evita respuestas arbitrarias.

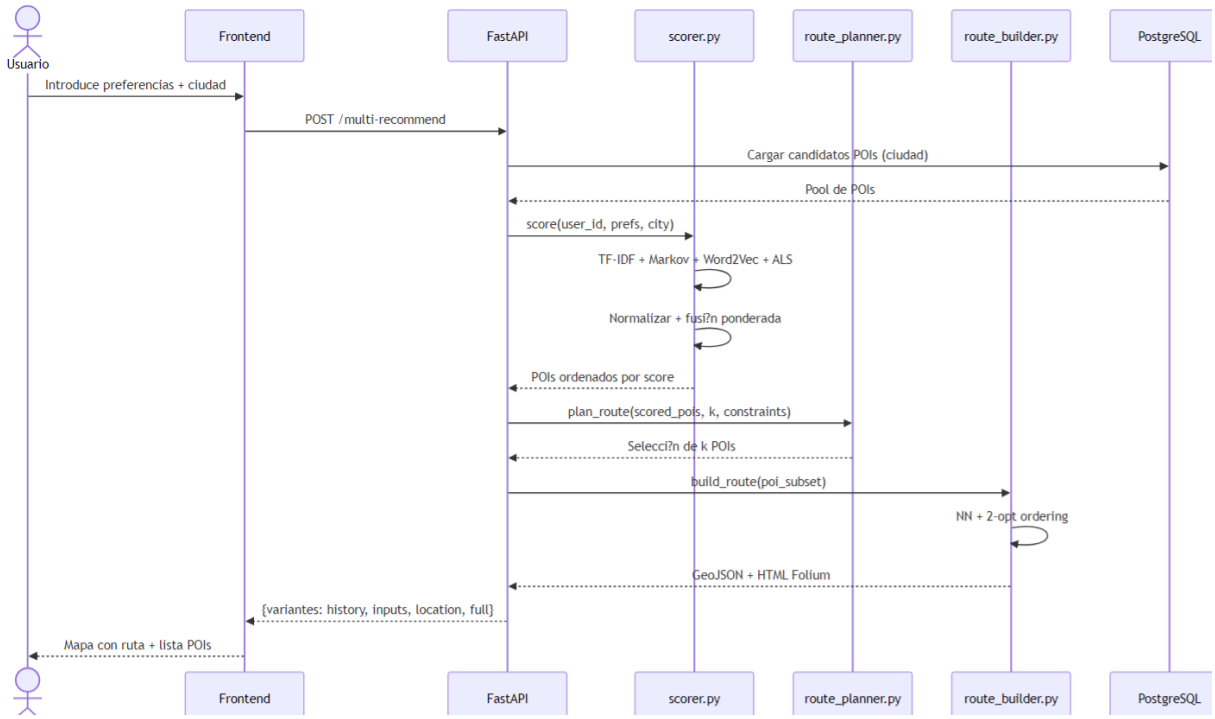


Figura 3.11: Diagrama de secuencia de una petición de recomendación: el usuario configura ciudad y preferencias en el frontend, que invoca `POST /multi-recommend`; la API carga candidatos desde PostgreSQL, activa `scorer.py` (fusión ponderada de motores), delega en `route_planner.py` para seleccionar el subconjunto coherente y en `route_builder.py` para ordenarlo (heurística NN + 2-opt), y devuelve las cuatro variantes de ruta.

En conjunto, este pipeline permite pasar de una colección heterogénea de señales a una salida estructurada, interpretable y adaptable a distintos escenarios de uso. El detalle del flujo se muestra en la Figura 3.11, donde pueden distinguirse los actores principales implicados (frontend, API, módulos `scorer.py`, `route_planner.py` y `route_builder.py`, y base de datos PostgreSQL) y la secuencia de llamadas que se producen ante una petición de recomendación.

3.2.5 Construcción y visualización de rutas

Una vez obtenida la lista de puntos de interés recomendados, el sistema debe transformarla en una ruta utilizable. Para ello se aplica una fase específica de planificación y ordenación, separada del scoring de recomendación.

La construcción de rutas se organiza en dos pasos. En primer lugar, se selecciona un subconjunto de POIs que mantenga una cierta coherencia espacial y diversidad temática. En segundo lugar, ese subconjunto se ordena en una secuencia de visita razonable.

Para la ordenación final, el sistema utiliza una heurística inicial basada en *Nearest Neighbor*, seguida de una mejora local mediante *2-opt*. Esta estrategia permite obtener recorridos suficientemente buenos para el tamaño de ruta considerado en el proyecto, sin necesidad de resolver un problema exacto de optimización combinatoria.

El resultado de esta fase es una ruta ordenada con información asociada sobre distancia y estructura del itinerario, preparada para su exportación y visualización posterior.

La visualización cartográfica constituye la última capa funcional del sistema y permite representar las rutas generadas de forma clara e interpretable. En este dominio, la salida no puede limitarse a una lista textual de POIs, ya que la utilidad práctica de la recomendación depende también de su comprensión espacial.

El sistema genera representaciones estructuradas de la ruta, incluyendo GeoJSON y salidas HTML orientadas a inspección visual. Sobre estas salidas se muestran los puntos de interés ordenados, la relación entre paradas y la geometría general del recorrido.

La representación cartográfica combina dos niveles. Por un lado, se muestran conexiones rectas de referencia entre puntos consecutivos. Por otro, cuando es posible, se superpone una geometría aproximada por calles, lo que mejora la interpretabilidad del itinerario.

Además, la visualización incorpora elementos de apoyo como numeración de paradas, colores por tramo, leyenda y cambio de mapa base. De este modo, la ruta final no solo puede evaluarse numéricamente, sino también inspeccionarse de forma cualitativa desde una perspectiva de uso real.

3.3 Implementación

3.3.1 Stack tecnológico y organización del repositorio

La implementación del sistema se ha realizado principalmente en Python, al tratarse de un entorno adecuado para tareas de ingeniería de datos, recomendación, evaluación experimental y desarrollo rápido de prototipos.

Como sistema de persistencia se ha utilizado PostgreSQL, que actúa como base estructurada para almacenar visitas históricas, metadatos de puntos de interés, categorías y rutas guardadas. En la capa de producto, el backend se ha implementado con FastAPI, mientras que la interfaz de usuario se ha desarrollado como una aplicación web ligera basada en HTML, CSS y JavaScript. Para la visualización cartográfica se utiliza Leaflet, que permite representar puntos de interés, rutas y distintas capas base de mapa.

El sistema incorpora además artefactos entrenados por ciudad para embeddings y ALS, ficheros de configuración TOML y scripts específicos para entrenamiento, evaluación, benchmark y generación de figuras. En conjunto, este stack tecnológico permite cubrir de forma integrada la preparación de datos, la inferencia, la validación experimental y la interacción final con el usuario.

Desde el punto de vista hardware, el proyecto está orientado a un entorno de desarrollo local, suficiente para levantar la base de datos, ejecutar los modelos y lanzar evaluaciones y pruebas funcionales sobre las ciudades de estudio.

El repositorio se organiza de forma modular, separando datos, lógica de recomendación, evaluación y capa de producto. La carpeta `src/etl/` contiene el pipeline de limpieza, transformación y carga de datos en PostgreSQL. La carpeta `sql/` recoge el esquema relacional utilizado por la base de datos. El núcleo del sistema se encuentra en `src/recommender/`, donde se sitúan los modelos, el scoring, la construcción de rutas, la API, el entrenamiento y el tuning.

Dentro de este bloque destacan especialmente varios módulos: `scorer.py`, responsable de la integración de motores y del reranking; `route_planner.py`, orientado a la selección de subconjuntos coherentes de POIs; `route_builder.py`, encargado de la ordenación final y exportación de rutas; `multi_route_service.py`, que implementa el esquema de respuesta multi-ruta (contrato de la API); y `api.py`, que expone la funcionalidad del sistema mediante FastAPI.

La evaluación se mantiene separada en `src/recommender/eval/`, donde se encuentran los scripts de evaluación de ranking y calidad de ruta. La carpeta `frontend/` contiene la interfaz web. Por su parte, `configs/` almacena la configuración global y los overrides por ciudad, mientras que `data/` agrupa datos procesados, artefactos, rutas exportadas, reportes y figuras.

Esta organización modular facilita la mantenibilidad del proyecto, mejora la reproducibilidad y permite evolucionar componentes concretos sin afectar directamente al resto del sistema.

Tabla 3.3: Bloques principales del repositorio y su función

Bloque	Función principal
`src/etl/`	Limpieza, transformación y carga de datos en PostgreSQL.
`src/recommender/`	Núcleo del sistema: motores, scoring, rutas, API, entrenamiento y tuning.
`src/recommender/eval/`	Evaluación offline de ranking y calidad de ruta.
`frontend/`	Interfaz web y recursos estáticos de visualización.
`configs/`	Configuración global y por ciudad mediante archivos TOML.
`sql/`	Esquema de base de datos.
`data/`	Datos procesados, artefactos, reportes, rutas y figuras.
`docs/`	Documentación técnica y operativa del proyecto.

3.3.2 Persistencia de datos y pipeline ETL

La capa de persistencia del sistema se apoya en PostgreSQL como base de datos principal. Esta elección permite mantener una estructura clara y consistente para las visitas históricas, los metadatos de los puntos de interés, la taxonomía de categorías y las rutas guardadas.

El esquema de persistencia se organiza principalmente en cuatro tablas. La tabla `visits` almacena los eventos históricos de visita, incluyendo usuario, *trail*, punto de interés y ciudad. La tabla `pois` contiene los metadatos principales de cada POI, como nombre, coordenadas, *rating* o categoría principal. La tabla `poi_categories` amplía la información semántica con la taxonomía completa de categorías asociadas a cada lugar. Finalmente, `saved_routes` se utiliza para almacenar rutas guardadas desde la capa de producto.

Una decisión importante en el diseño de esta capa es el uso de `city_qid` como identificador de ciudad. Esto permite segmentar datos, configuraciones y artefactos por ciudad de forma robusta y evita inconsistencias derivadas de nombres textuales.

El sistema incorpora un pipeline ETL de ocho pasos que transforma el CSV de partida en una base de datos estructurada y enriquecida (Figura 3.7). El diseño responde a una decisión de separación de responsabilidades. Cada etapa resuelve un problema acotado, puede ejecutarse o relanzarse de forma independiente y deja un artefacto intermedio en disco, lo que facilita la depuración y la reproducibilidad sin necesidad de reejecutar toda la cadena.

Las primeras etapas, limpieza y extracción de identificadores, estabilizan el conjunto de datos. Filtran registros irrelevantes, normalizan formatos y enumeran los venues únicos que serán consultados. La consulta a la API Foursquare Places se aísla en un paso propio porque es el único con dependencia externa (requiere clave de API y realiza llamadas de

red con coste) y porque su resultado puede reutilizarse en reprocesados posteriores sin volver a consumir cuota. La normalización y el etiquetado que siguen operan sobre el JSON crudo de la API para aplanar jerarquías de categorías y estimar franjas de precio cuando Foursquare no las proporciona explícitamente; esta separación evita mezclar lógica de acceso a datos con lógica de transformación.

La imputación se ubica al final del bloque de transformaciones porque depende de la taxonomía de categorías ya estabilizada. Las coordenadas faltantes se recuperan por K-NN espacial y las categorías ausentes por *forward-fill*, evitando que los modelos operen con metadatos incompletos. El paso de diagnóstico precede a la carga para que cualquier anomalía de cobertura o distribución se detecte antes de que los datos lleguen a PostgreSQL. Finalmente, la carga se realiza mediante *upsert*, lo que permite actualizar el catálogo sin perder datos existentes. El desglose técnico de cada script se recoge en la Tabla A.5 del apéndice.

El proyecto mantiene los artefactos intermedios en disco, lo que permite relanzar la cadena desde cualquier punto sin reejecutar pasos costosos como la consulta a la API. En conjunto, la base de datos y el pipeline ETL proporcionan la infraestructura sobre la que se apoyan los modelos de recomendación, la evaluación y la capa de producto.

3.3.3 Artefactos entrenados y configuración

El sistema entrena artefactos específicos por ciudad para aquellos motores que no pueden operar únicamente con reglas o cálculos directos sobre la base de datos. En el estado actual del proyecto, esto afecta principalmente a los modelos de embeddings secuenciales y al modelo ALS de filtrado colaborativo implícito.

Esta estrategia de entrenamiento independiente por ciudad se justifica porque los patrones de movilidad, la cobertura de puntos de interés y la densidad de interacciones varían entre entornos urbanos. Por ello, el proyecto mantiene artefactos diferenciados para cada ciudad, almacenados en caché y reutilizados posteriormente durante inferencia y evaluación. El artefacto resultante de los embeddings se guarda con la forma `word2vec_<city_qid>.joblib`. Y el modelo ALS persiste como `als_<city_qid>.joblib`. Esta organización facilita la trazabilidad experimental y evita mezclar modelos entrenados sobre contextos urbanos distintos.

La configuración del sistema se ha diseñado de forma centralizada y desacoplada del código fuente. Esta decisión permite ajustar pesos, hiperparámetros y restricciones sin modificar directamente la lógica implementada, lo que mejora tanto la mantenibilidad como la reproducibilidad.

El proyecto utiliza un fichero de configuración global, `configs/recommender.toml`, y un *override* específico por ciudad, `configs/recommender_<city_qid>.toml`. De este modo, el sistema puede mantener valores comunes por defecto y, al mismo tiempo, adaptar ciertos parámetros al comportamiento de cada ciudad.

Entre los elementos controlados por configuración se incluyen los pesos del modelo híbrido,

parámetros de embeddings y ALS, restricciones del planificador de rutas, semillas por defecto para evaluación y políticas de filtrado. Esta separación entre código y configuración refuerza la trazabilidad experimental del proyecto y facilita comparar variantes del sistema bajo condiciones controladas.

3.3.4 Modos de ejecución e interfaz de usuario

El sistema admite dos modos principales de ejecución. Por un lado, un modo orientado a investigación y depuración mediante línea de comandos. Por otro, un modo de producto, basado en API y frontend web. Esta separación permite mantener un entorno reproducible para experimentación y, al mismo tiempo, exponer el sistema como una aplicación interactiva. De forma general, el flujo operativo del proyecto sigue esta secuencia: preparación y carga de datos en PostgreSQL, entrenamiento de artefactos cuando procede, ejecución de inferencia o evaluación y, finalmente, visualización y análisis de resultados.

En el modo de investigación, la línea de comandos constituye la vía principal para entrenamiento, depuración, *tuning* y evaluación reproducible. A través de la CLI pueden ejecutarse recomendaciones simples, recomendaciones multi-ruta, entrenamiento de embeddings y ALS, evaluación offline y *benchmark* multi-ciudad. Este modo resulta especialmente útil porque permite fijar de forma explícita parámetros como ciudad, usuario, modo de recomendación, número de paradas, uso de artefactos entrenados, protocolo experimental y rutas de salida. El segundo modo de ejecución corresponde a la capa de producto, formada por un backend FastAPI y una interfaz web ligera. En este modo, el backend recibe las peticiones, consulta la base de datos, activa el pipeline de recomendación y devuelve la respuesta estructurada. El frontend permite al usuario seleccionar ciudad, preferencias, número de paradas y localización, y visualizar después las rutas generadas sobre mapa.

La interfaz web constituye la capa visible del sistema y la principal vía de interacción en modo producto. Se ha diseñado como una aplicación ligera de una sola página, conectada al backend real del recomendador. Desde ella, el usuario puede seleccionar la ciudad, introducir preferencias, fijar restricciones simples de presupuesto o gratuidad, aportar opcionalmente un identificador de usuario y seleccionar una localización inicial sobre mapa. Una vez recibida la respuesta, la interfaz permite alternar entre variantes de ruta y explorar visualmente la recomendación.

La Figura 3.12 muestra el prototipo web funcional sobre Osaka. En ella se observa el panel de configuración de la petición (ciudad, número de paradas, preferencias, presupuesto, proximidad y usuario opcional), el mapa central con la ruta numerada y la sección inferior con variantes de recomendación y lista de POIs con categoría, rating y distancia.

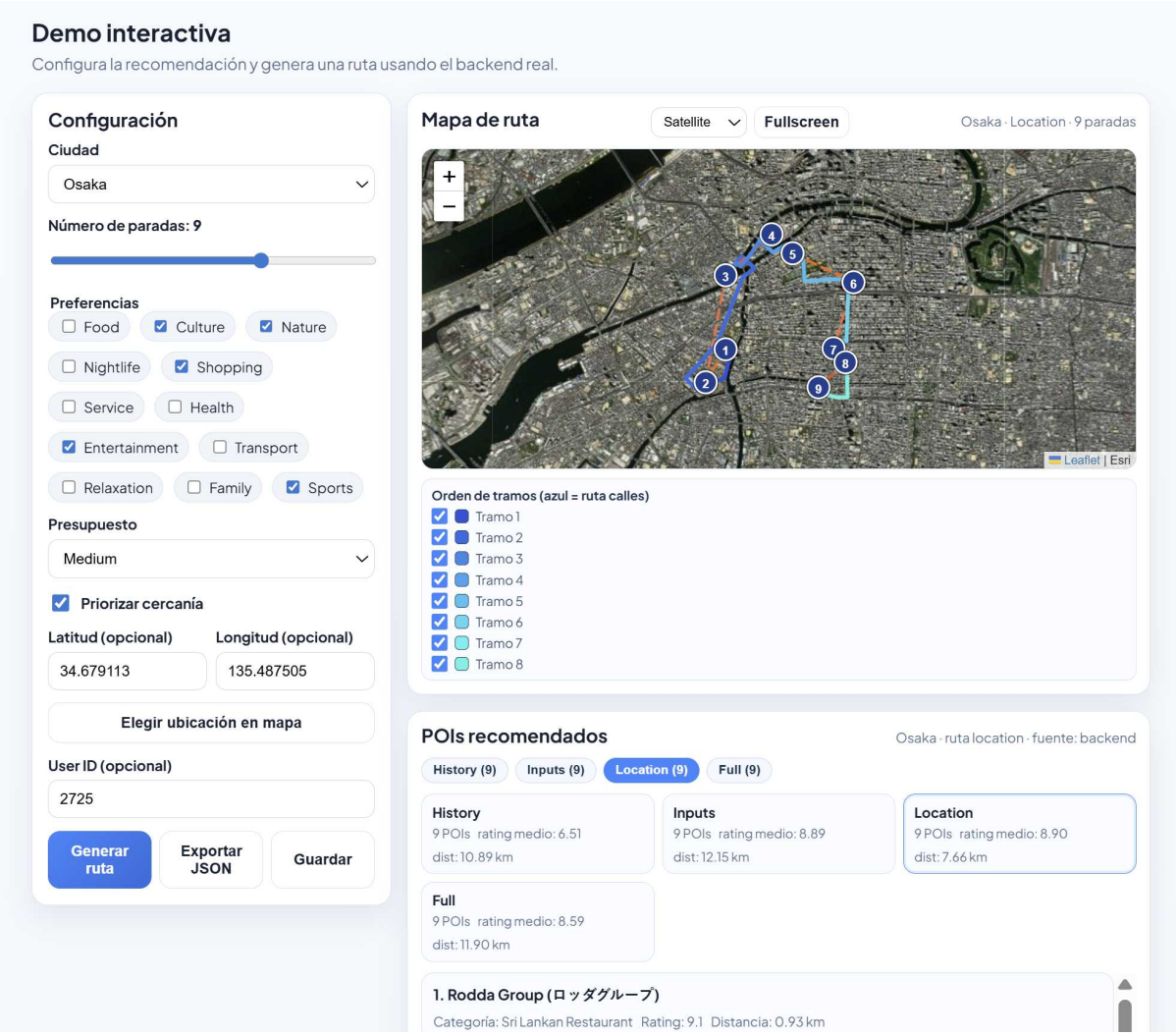


Figura 3.12: Prototipo web funcional del sistema sobre Osaka.

La representación cartográfica se realiza con Leaflet y, según la capa seleccionada, utiliza mapas base de OpenStreetMap y Esri; el cálculo de rutas se apoya en OSRM. La visualización incluye puntos numerados, segmentos por tramo, cambio de mapa base y visualización de rutas guardadas. Además, el frontend muestra información de variantes omitidas o advertencias devueltas por el backend, lo que mejora la interpretabilidad del sistema.

3.3.5 Reproducibilidad experimental

La reproducibilidad constituye un principio central del proyecto. El comportamiento del sistema se controla mediante configuración centralizada, artefactos entrenados por ciudad y protocolos de evaluación fijos, lo que permite relanzar experimentos bajo condiciones homogéneas.

El proyecto incorpora además scripts específicos para *benchmark* y automatización experi-

mental. Estos scripts permiten ejecutar de forma sistemática entrenamiento, evaluación de ranking y evaluación de calidad de ruta sobre las ciudades de estudio, reduciendo errores manuales y mejorando la consistencia metodológica. Asimismo, las salidas experimentales se almacenan de forma estructurada en forma de JSON, reportes y figuras, lo que facilita tanto la auditoría posterior como la regeneración de tablas y gráficos de la memoria. En conjunto, esta orientación a reproducibilidad refuerza la validez experimental del sistema y la trazabilidad de los resultados obtenidos.

3.4 Protocolo de evaluación

3.4.1 Objetivo y protocolo experimental

La validación del sistema se ha diseñado con una doble finalidad. Por un lado, evaluar si los motores de recomendación son capaces de recuperar y ordenar correctamente puntos de interés relevantes. Por otro, comprobar si esos puntos recomendados pueden transformarse en rutas razonables desde una perspectiva práctica, es decir, recorridos suficientemente compactos, coherentes y útiles para un usuario final.

El protocolo principal utilizado en este trabajo es `last_trail_user`. Para cada usuario evaluable, el último *trail* se reserva como prueba (*test*) y los *trails* anteriores se usan como entrenamiento (*train*). Dentro de este protocolo, el primer POI del *trail* de *test* actúa como semilla y el resto de POIs constituye la verdad de referencia. Además, los modelos que dependen de entrenamiento (`embed` y `als`) se reentrenan exclusivamente sobre *train*, evitando fugas de información entre entrenamiento y prueba. Esta configuración aplica por igual a todos los modelos evaluados, incluyendo los baselines.

El resultado próximo a cero del motor aleatorio no es un error, sino una consecuencia directa del protocolo: con $k = 20$ candidatos sorteados entre los miles de POIs no visitados en entrenamiento, la probabilidad de acertar alguno de los 2–5 ítems de verdad es inferior al 1 % por usuario. Esta formulación resulta adecuada, ya que preserva la estructura secuencial de las trayectorias y aproxima la evaluación a un escenario realista de recomendación de ruta. La evaluación se ejecuta en modo `--fair`. La Figura 3.13 sintetiza visualmente el protocolo completo, desde la partición de datos hasta la evaluación de los nueve motores.

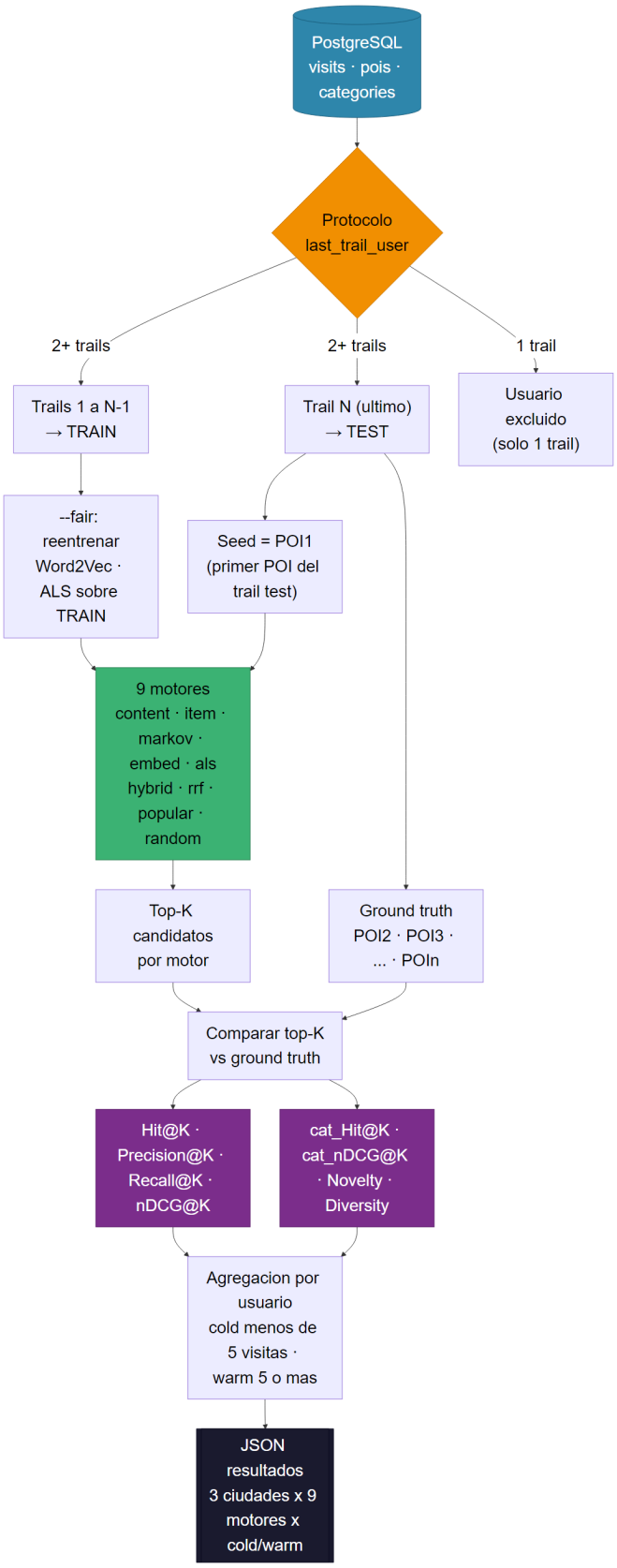


Figura 3.13: Diagrama del protocolo de evaluación.

3.4.2 Configuración experimental

La evaluación se ha realizado sobre las tres ciudades operativas del sistema. En ranking, la configuración principal utiliza $k = 20$, `seed=42`, `max_users=300`, `min_train=2` y `min_test_pois=4`. En evaluación de rutas, la configuración principal utiliza $k = 8$, `max_cases=200` y las mismas restricciones de elegibilidad para entrenamiento y prueba. Estas elecciones permiten equilibrar coste computacional, cobertura de usuarios evaluables y comparabilidad entre motores y ciudades. Además, se mantienen fijos tanto el protocolo como la semilla para evitar comparaciones inconsistentes entre ejecuciones y facilitar la trazabilidad de los resultados.

Los hiperparámetros de los modelos no se fijaron manualmente, sino mediante barrido discreto por ciudad sobre espacios de búsqueda acotados, utilizando *nDCG@20* como criterio principal con *MRR* como desempate (excepto en el planificador de rutas, donde se emplea una función objetivo de calidad de ruta). La Tabla 3.4 resume los parámetros barridos, sus espacios de búsqueda y los valores seleccionados por ciudad; el detalle completo se recoge en el Apéndice A.2. El valor $k = 20$ se escogió por ser habitual en la literatura de recuperación de información sobre POIs y por ofrecer una ventana de recomendación realista para una ruta de un día. Los umbrales `min_train=2` y `min_test_pois=4` garantizan que cada usuario evaluado dispone de al menos un *trail* de entrenamiento con señal mínima y que la verdad de referencia del test contiene suficientes ítems para que las métricas sean informativas.

Tabla 3.4: Resumen del barrido de hiperparámetros por componente y ciudad.

Script	Parámetro	Espacio de búsqueda	Criterio	Osaka	Istanbul	PJ
`tune_markov.py`	`backoff`	{0.10, 0.15, …, 0.45}	nDCG@20 + MRR	0.30	0.25	0.25
`tune_embeddings_scoring.py`	`context_n`	{1, 2, 3}	nDCG@20 + MRR	2	1	3
`tune_als.py`	`factors`	{64, 128, 256}	nDCG@20 + MRR	64	128	64
`tune_als.py`	`iterations`	{15, 30, 60}	nDCG@20 + MRR	15	15	15
`tune_route_planner.py`	`candidate_pool`	{300, 600, 1000}	calidad ruta	300	600	600
`tune_hybrid.py`	pesos escenario `user_current`	por ciudad	nDCG@20 + MRR	véase Tabla 3.2		

3.4.3 Modelos y métricas consideradas

En la evaluación de ranking se comparan nueve modos de recomendación: `content`, `item`, `markov`, `embed`, `als`, `hybrid`, `rrf`, `popular` y `random`, lo que permite comparar enfoques semánticos, colaborativos, secuenciales, híbridos y baselines simples dentro del mismo protocolo experimental. Para la evaluación de calidad de ruta se consideran motores que generan salidas comparables en forma de itinerario: `content`, `item`, `markov`, `embed`, `als` y `hybrid`. Las métricas reportadas en ranking son *Hit@20*, *Precision@20*, *Recall@20* y *nDCG@20*, junto con sus equivalentes a nivel de categoría: *cat_hit@20*, *cat_precision@20*, *cat_recall@20* y *cat_nDCG@20* (en las tablas de resultados se presentan únicamente *cat_hit@20* y *cat_nDCG@20* por brevedad; las cuatro métricas están disponibles en los ficheros de resultados).

Estas métricas se justifican porque en turismo el objetivo del usuario no es necesariamente

visitar un lugar concreto, sino un tipo de experiencia: gastronomía, cultura, naturaleza u ocio. Recuperar el POI exacto es relevante, pero acertar el tipo de experiencia esperada es también un indicativo de acierto semántico.

Las métricas de categoría complementan a las exactas y permiten distinguir motores que aciertan el POI de aquellos que aciertan la experiencia buscada, lo que resulta especialmente informativo cuando los datasets presentan alta variabilidad en la cobertura de POIs concretos. La inclusión de `popular` y `random` como baselines permite anclar la escala de resultados: `popular` representa el límite de lo que puede lograrse sin ninguna personalización, mientras que `random` establece la cota de ruido puro esperada bajo el protocolo empleado. Además, se incluyen *novelty* y *diversity* como métricas complementarias: *novelty* mide la impopularidad media de los ítems recomendados, cuanto más alta, más inesperados son los POIs sugeridos, y *diversity* mide la variedad de categorías dentro de la lista recomendada; ambas capturan dimensiones de la calidad de recomendación que las métricas de recuperación exacta no contemplan. En calidad de ruta se reportan indicadores agregados como `n_routes`, `total_km`, `avg_leg_km`, `pct_legs_too_close`, `pct_legs_too_far` y `cat_match_ratio`.

Detrás de la arquitectura descrita en este capítulo hay un proceso de diseño que no fue lineal. Cada componente resolvió un problema concreto que fue apareciendo a lo largo del desarrollo: cómo representar las preferencias cuando no hay historial previo, cómo normalizar puntuaciones de motores con escalas completamente distintas, cómo construir una ruta que no lleve al usuario de un extremo de la ciudad al otro, o cuántos factores latentes necesita ALS para ser útil en una ciudad con datos escasos. Ninguna de esas preguntas tiene una respuesta única; todas requirieron iteración sobre datos reales y un criterio claro sobre qué se quería medir. Lo que queda es un sistema que entiende el problema que tiene delante: sabe cuándo apoyarse en patrones secuenciales, cuándo recurrir a señales colaborativas y cuándo simplemente usar lo que el contexto de la petición trae consigo. Esa capacidad de adaptación es, en última instancia, la decisión de diseño más importante de todo el trabajo.

4. Resultados

En este capítulo se presentan los resultados de la validación experimental del sistema y se realiza una interpretación crítica de los mismos. El protocolo de evaluación, la configuración experimental y las métricas empleadas se han descrito en la Sección 3.4. A partir de esa base, aquí se analizan, en primer lugar, los resultados de ranking de los motores de recomendación y, en segundo lugar, la calidad de las rutas generadas. Finalmente, se incorporan dos análisis complementarios sobre escenarios *cold-start* y posicionamiento orientativo frente a la literatura.

4.1 Resultados de ranking

En esta sección se comparan los motores de recomendación en la tarea de ranking de POIs bajo un mismo protocolo experimental. El análisis se realiza sobre las tres ciudades operativas del sistema y combina métricas exactas de recuperación con una lectura comparativa por ciudad y por tipo de señal explotada.

4.1.1 Comparativa global entre motores

La visión global de los resultados muestra que no existe un único motor dominante en todas las ciudades y métricas, aunque sí se observan patrones consistentes. En conjunto, los enfoques `hybrid`, `rrf`, `item` y `markov` concentran el mejor comportamiento global, mientras que `content` y, sobre todo, `random`, quedan claramente por detrás en capacidad de recuperación exacta de POIs. La comparación entre motores permite distinguir varios perfiles de comportamiento. Los modelos `item` y `markov` ofrecen resultados sólidos cuando la estructura histórica de visitas contiene patrones claros de co-visita o continuidad secuencial. El motor `als` presenta un comportamiento intermedio. Resulta competitivo cuando existe suficiente señal colaborativa, pero pierde fuerza en ciudades o escenarios con menor densidad de interacción. Por su parte, `embed` muestra un rendimiento desigual, con una caída especialmente acusada en Istanbul, lo que sugiere una mayor sensibilidad a escasez de datos y problemas de cobertura del vocabulario secuencial.

El motor `content` aunque obtiene valores bajos en métricas de recuperación exacta, destaca por su alta *novelty*, lo que indica que tiende a recomendar lugares menos obvios y menos repetitivos. Sin embargo, esta ventaja no compensa su debilidad en relevancia cuando se evalúa contra rutas reales. En el extremo opuesto, `popular` consigue resultados competitivos en varias ciudades, lo que confirma que la popularidad global sigue siendo una señal fuerte en turismo, aunque insuficiente como solución completa; actúa como referencia de comparación. El baseline `random`, por su parte, actúa como una cota inferior útil. Sus resultados en recuperación exacta son nulos o residuales, lo que refuerza que la tarea no puede resolverse de forma competitiva sin explotar estructura real de los datos. Los resultados también muestran que `rrf` funciona como una fusión muy eficaz

de rankings base, especialmente en Osaka y Petaling Jaya, mientras que `hybrid` destaca por ofrecer un equilibrio más consistente entre recuperación, ordenación y adaptabilidad al contexto. En consecuencia, los motores más útiles para el sistema final no son los más simples ni los más especializados aisladamente, sino aquellos capaces de integrar varias señales o aprovechar de forma robusta la estructura real del histórico de visitas. La Tabla 4.1 resume estas métricas por motor y ciudad, y la Figura 4.1 facilita su lectura comparativa en *nDCG@20*.

Tabla 4.1: Resumen comparativo de métricas offline por motor y ciudad.

Protocolo: `last_trail_user` · `--fair` · $k = 20$ · `seed=42`

Osaka (`Q35765`)

Motor	Hit@K	Prec@K	Recall@K	nDCG@K	Novelty	Diversity
rrf	0.443	0.032	0.172	0.107	0.477	0.342
item	0.378	0.029	0.153	0.099	0.459	0.426
markov†	0.380	0.027	0.149	0.100	0.600	0.674
popular	0.356	0.026	0.133	0.090	0.421	0.433
hybrid	0.450	0.031	0.165	0.107	0.497	0.370
als	0.325	0.020	0.109	0.068	0.566	0.570
embed	0.253	0.017	0.090	0.055	0.576	0.501
content	0.149	0.009	0.047	0.024	0.711	0.090
random	0.000	0.000	0.000	0.000	0.914	0.860

Istanbul (`Q406`)

Motor	Hit@K	Prec@K	Recall@K	nDCG@K	Novelty	Diversity
rrf	0.247	0.013	0.198	0.092	0.557	0.585
item	0.264	0.015	0.210	0.108	0.467	0.678
markov†	0.302	0.017	0.259	0.145	0.550	0.739
popular	0.255	0.014	0.198	0.087	0.388	0.712
hybrid	0.224	0.012	0.183	0.095	0.540	0.633
als	0.075	0.004	0.060	0.026	0.655	0.737
embed	0.004	0.000	0.002	0.001	0.852	0.754
content	0.024	0.001	0.019	0.010	0.766	0.141
random	0.012	0.001	0.010	0.003	0.837	0.865

Petaling Jaya (`Q864965`)

Motor	Hit@K	Prec@K	Recall@K	nDCG@K	Novelty	Diversity
rrf	0.424	0.028	0.169	0.105	0.526	0.462
item	0.396	0.026	0.164	0.102	0.531	0.607
markov†	0.333	0.021	0.133	0.097	0.660	0.796
popular	0.420	0.027	0.169	0.103	0.430	0.354
hybrid	0.424	0.028	0.177	0.120	0.535	0.487
als	0.248	0.016	0.103	0.063	0.582	0.704
embed	0.148	0.008	0.050	0.027	0.707	0.744
content	0.100	0.006	0.032	0.014	0.736	0.119
random	0.000	0.000	0.000	0.000	0.906	0.882

Verde = mejor motor por columna (excl. random) · Rojo = peor motor por columna · † N reducido (véase Secc. B.8)

4.1.2 Diferencias por ciudad y lectura de las métricas

El análisis por ciudad muestra que el comportamiento de los motores no es uniforme y depende de la estructura de datos y de los patrones de movilidad presentes en cada entorno.

En Osaka, los mejores resultados globales se reparten entre `hybrid` y `rrf`. El motor `hybrid` obtiene el mejor valor de *Hit@20*, mientras que `rrf` lidera *Precision@20* y *Recall@20*; ambos motores empatan prácticamente en *nDCG@20* (0,107). Esto sugiere que en esta ciudad la combinación de señales funciona especialmente bien y que la fusión de rankings proporciona una ordenación final muy competitiva. También destacan `item` y `markov`, ambos con resultados sólidos, mientras que `content` queda claramente por detrás en relevancia exacta.

En Istanbul se observa un patrón diferente. En esta ciudad, `markov` es el mejor motor en las principales métricas de ranking y supera tanto a `hybrid` como a `rrf`. Este comportamiento se explica en buena medida por la menor densidad de datos disponibles en Istanbul respecto a Osaka (véase Figura 3.3): con un volumen de check-ins y una red de transiciones más reducida (Figura 3.10), el motor Markov resulta más robusto porque no depende de representaciones latentes aprendidas, sino de frecuencias de transición directas que funcionan incluso con datos escasos. Por el contrario, motores como `embed` y `als` necesitan mayor cobertura de interacciones para construir representaciones fiables. Con pocos datos, los vectores de usuario o POI son ruidosos y el modelo tiende a colapsar hacia soluciones degeneradas. Este contraste se refleja también en el análisis cold/warm (Figura 4.3). La brecha de rendimiento entre usuarios con y sin historial es mayor en Istanbul que en el resto de ciudades, lo que confirma que la escasez de señal histórica penaliza especialmente a los motores colaborativos y de representación latente.

En Petaling Jaya, la situación vuelve a ser distinta. Los modelos `hybrid` y `rrf` empatan en *Hit@20*, mientras que `hybrid` obtiene los mejores valores en *Recall@20* y *nDCG@20*. Además, `popular` se mantiene muy cerca de los mejores motores, lo que indica que en esta ciudad la señal de popularidad global tiene un peso importante. Aun así, el enfoque híbrido sigue siendo el más equilibrado cuando se consideran simultáneamente recuperación, ordenación y adaptación contextual.

Además de las métricas exactas de recuperación, las métricas a nivel de categoría aportan una lectura semántica complementaria. En Osaka destaca especialmente `als` en adecuación por tipo de experiencia, mientras que en Istanbul vuelve a sobresalir `markov` y en Petaling Jaya se observa un patrón más repartido entre señales colaborativas, secuenciales e híbridas. Esta lectura refuerza la idea de que el sistema no debe evaluarse únicamente por acierto exacto de POI, sino también por su capacidad de aproximarse al tipo de experiencia turística esperada.

En conjunto, el análisis por ciudad confirma que no existe un único motor dominante en todos los escenarios. Sin embargo, también muestra que los enfoques híbridos y las estrategias de fusión son los más robustos globalmente, mientras que motores específicos como `markov` pueden destacar con fuerza en entornos concretos.

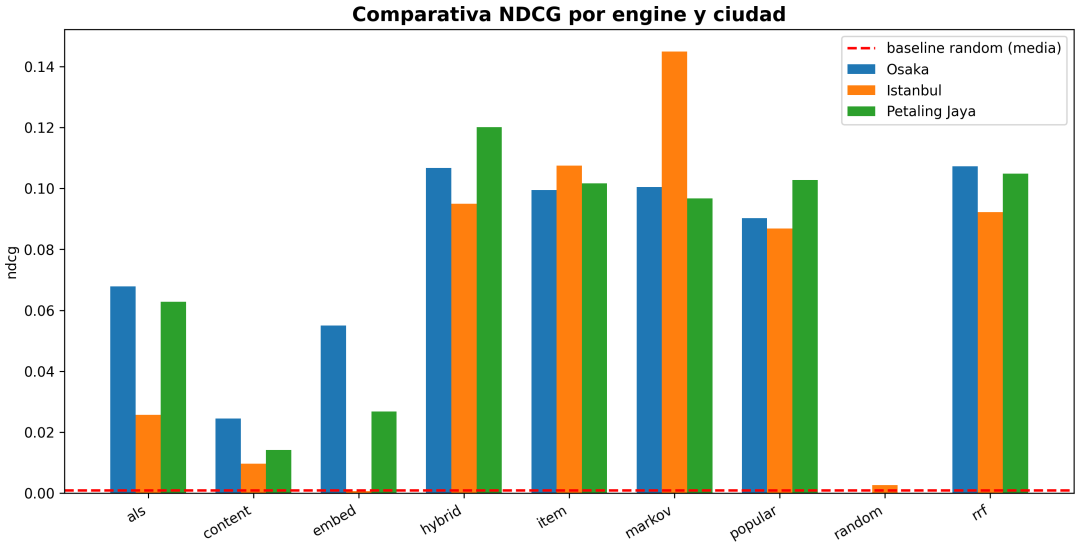


Figura 4.1: Comparación de *nDCG@20* por motor y ciudad.

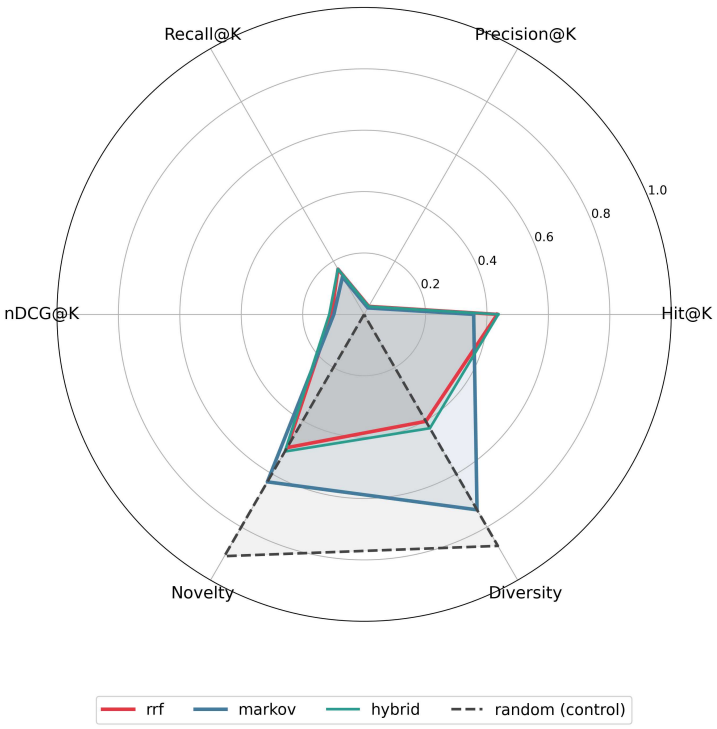


Figura 4.2: Perfil multi-métrica de los motores seleccionados sobre la media de Osaka y Petaling Jaya. Cada eje representa una métrica (Hit@20, nDCG@20, Precision@20, Recall@20, Novelty y Diversity); el área del polígono refleja el equilibrio global del motor. *Novelty* mide la impopularidad media de los ítems recomendados; *Diversity* mide la variedad de categorías dentro de la lista recomendada.

4.1.3 Compromiso entre relevancia, novedad y diversidad

Uno de los fenómenos más destacados que muestra la Figura 4.2 es la existencia de un compromiso inherente entre relevancia exacta y variedad (entendida aquí como novedad "*novelty*" y diversidad "*diversity*" conjuntamente). El área del polígono refleja el equilibrio general de cada motor y las irregularidades revelan los *trade-offs* entre calidad de ranking y variedad. Los motores con mayor *Hit@20* y *nDCG@20* tienden a concentrar sus recomendaciones en los POIs más populares y frecuentes, lo que eleva la recuperación exacta pero reduce la *novelty* y la *diversity*. Por el contrario, motores como `content` y `random` producen recomendaciones más variadas y novedosas, pero a costa de una caída clara en relevancia directa.

Este *trade-off* entre precisión y diversidad es estructural en los sistemas de recomendación [13]. Maximizar un único indicador de relevancia no produce automáticamente el sistema más útil. En el contexto turístico, un usuario puede valorar que la ruta incluya algún lugar menos obvio o que aporte una experiencia inesperada, sin sacrificar la coherencia global del itinerario. Por ello, el motor `hybrid` resulta especialmente adecuado. No maximiza una sola dimensión, sino que mantiene un compromiso razonable entre recuperación, adecuación semántica y variedad. Esta propiedad lo convierte en la opción más equilibrada para el sistema de producto, donde el objetivo no es optimizar una métrica aislada sino ofrecer rutas útiles y variadas en distintos contextos de usuario.

4.2 Resultados de calidad de ruta

En esta sección se analiza la calidad de las rutas generadas por el sistema. A diferencia de la evaluación de ranking, aquí el objetivo no es solo comprobar si los puntos de interés recomendados son relevantes, sino también si la secuencia final constituye un itinerario razonable desde el punto de vista espacial y temático.

La evaluación de rutas se ha realizado sobre los motores que generan salidas comparables en forma de itinerario: `content`, `item`, `markov`, `embed`, `als` y `hybrid`. Los indicadores considerados permiten medir tanto la eficiencia espacial del recorrido como su adecuación semántica con respecto a las rutas de referencia.

Para apoyar esta lectura cualitativa, la Tabla 4.2 resume los principales indicadores de calidad de ruta en Osaka, que actúa como ciudad de referencia por volumen de datos y estabilidad experimental. En ella puede observarse con claridad el compromiso entre compacidad espacial y adecuación temática, así como el perfil más equilibrado del motor `hybrid` frente a motores más especializados.

Tabla 4.2: Resumen de calidad de ruta en Osaka.

Motor	Total km	Tramo medio	% muy cortos	% muy largos	Cat. match
content	7.519	1.085	0.001	0.026	0.433
item	4.137	0.519	0.000	0.000	0.171
markov	4.179	0.681	0.030	0.006	0.139
embed	3.416	0.427	0.000	0.000	0.134
als	4.499	0.562	0.000	0.000	0.177
hybrid	3.956	0.495	0.000	0.000	0.212

4.2.1 Coherencia espacial

Los resultados muestran que la coherencia espacial varía de forma importante entre motores. En general, los modelos `hybrid`, `item`, `embed` y `als` tienden a producir rutas más compactas, mientras que `content` y, en algunos casos, `markov`, generan trayectorias más largas o con mayor dispersión geográfica.

Este patrón puede interpretarse a partir de la naturaleza de cada enfoque. Los motores más guiados por similitud estructural o por señales colaborativas tienden a concentrar mejor los candidatos en zonas compatibles entre sí, mientras que `content` prioriza afinidad temática y puede recomendar lugares semánticamente plausibles pero menos conectados espacialmente. En el caso de `markov`, la fortaleza secuencial no siempre se traduce en el recorrido más compacto, ya que la transición históricamente frecuente entre dos POIs no implica necesariamente cercanía geográfica directa.

Desde una perspectiva de producto, este resultado es importante porque una buena recomendación de POIs no garantiza por sí sola una buena ruta. La fase de construcción de itinerario añade valor precisamente porque transforma una lista relevante en un recorrido más utilizable, pero aun así la calidad final depende del tipo de candidatos que cada motor propone antes de construir la ruta.

4.2.2 Adecuación temática y equilibrio global

La adecuación temática permite valorar si la ruta generada conserva una lógica semántica razonable con respecto a la referencia. En esta dimensión, `content` muestra un comportamiento relativamente favorable, ya que su sesgo hacia similitud categórica facilita acercarse al tipo de experiencia esperada. Sin embargo, esta ventaja aparece acompañada de una menor eficiencia espacial, lo que confirma la existencia de un compromiso entre coherencia temática y compacidad geográfica.

Los motores `item`, `als` y `hybrid` ofrecen un perfil más equilibrado. Sin liderar necesariamente en la dimensión temática más pura, consiguen mantener valores razonables de adecuación sin degradar la estructura espacial de la ruta. Esta combinación resulta especialmente valiosa para un sistema aplicado, donde una ruta útil debe ser interpretable tanto por contenido como por recorrido.

El caso de `markov` es interesante porque sus buenos resultados en ranking no siempre se traducen en la mejor adecuación global de ruta. Esto indica que recuperar POIs plausibles en secuencia no equivale automáticamente a construir el mejor itinerario completo. Del mismo modo, `embed` puede producir rutas bastante compactas, pero no siempre iguala a otros motores en ajuste semántico o coincidencia con la referencia.

Considerando conjuntamente coherencia espacial y adecuación temática, el motor `hybrid` es el que ofrece el comportamiento más equilibrado entre los evaluados. No lidera todas las métricas individuales, pero evita los extremos observados en otros motores y mantiene un rendimiento estable en las tres ciudades. Esto refuerza la misma conclusión observada en ranking, los modelos más útiles para el sistema final son aquellos capaces de combinar varias señales y mantener un compromiso razonable entre relevancia, coherencia espacial y adecuación temática.

Nota sobre variantes de producto. Al margen del análisis por motor, conviene señalar que la variante `history` del esquema de respuesta multi-ruta tiende a producir rutas con una media de rating inferior a otras variantes como `inputs` o `full`. Esto se debe a que construye el itinerario a partir de lugares que el usuario ya visitó en el pasado, incluyendo sitios frecuentados por proximidad, hábito o conveniencia, sin filtrar por valoración global. Las variantes `inputs` y `full`, orientadas por preferencias explícitas o fusión de señales, pueden priorizar POIs con mejor rating medio, generando rutas potencialmente mejor valoradas en términos de calidad absoluta.

4.3 Análisis complementario

Esta sección recoge dos análisis adicionales que ayudan a interpretar los resultados principales desde una perspectiva más aplicada. El primero estudia el comportamiento del sistema en escenarios *cold-start* frente a usuarios con suficiente historial. El segundo sitúa de forma orientativa el rendimiento del TFG frente a referencias representativas de la literatura.

4.3.1 Desglose cold-start frente a warm-start

El análisis *cold/warm* (Figura 4.3) permite observar cómo cambia el rendimiento según la cantidad de información histórica disponible por usuario. En este trabajo se considera *cold* a un usuario con menos de cinco visitas en entrenamiento y *warm* a uno con cinco o más, umbral habitual en la literatura de cold-start en sistemas de recomendación [7].

Los valores numéricos exactos de *Hit@20* por segmento se recogen en la Tabla B.2 del apéndice. La figura muestra un patrón inicialmente contraintuitivo. En varias ciudades y motores, las barras *cold* aparecen por encima de *warm* en *Hit@20* y *nDCG@20*. Esto no implica que el sistema recomiende mejor a usuarios desconocidos, sino un efecto estadístico del protocolo. En usuarios *cold* suele haber menos señal y trayectorias de prueba más

simples, de modo que acertar con recomendaciones frecuentes resulta relativamente más probable. También se observan diferencias claras entre motores y ciudades. En Istanbul, `embed` colapsa a valores prácticamente nulos en ambos segmentos y `als` opera con poca efectividad independientemente del nivel de historial disponible, lo que confirma su alta dependencia de cobertura histórica para aprender representaciones útiles. En cambio, `markov` mantiene el comportamiento más estable y `hybrid` conserva mejor rendimiento relativo al combinar señales heterogéneas.

Esta lectura ayuda a interpretar por qué `hybrid` y `rrf` funcionan muy bien en Osaka y Petaling Jaya pero pierden ventaja en Istanbul, con menor volumen y mayor *sparsity*, la calidad de varios motores base cae y la fusión dispone de menos señal robusta que combinar. Desde el punto de vista aplicado, el resultado sugiere no depender en exceso de modelos colaborativos/latentes en entornos de baja cobertura y reforzar componentes secuenciales y de fusión robusta.

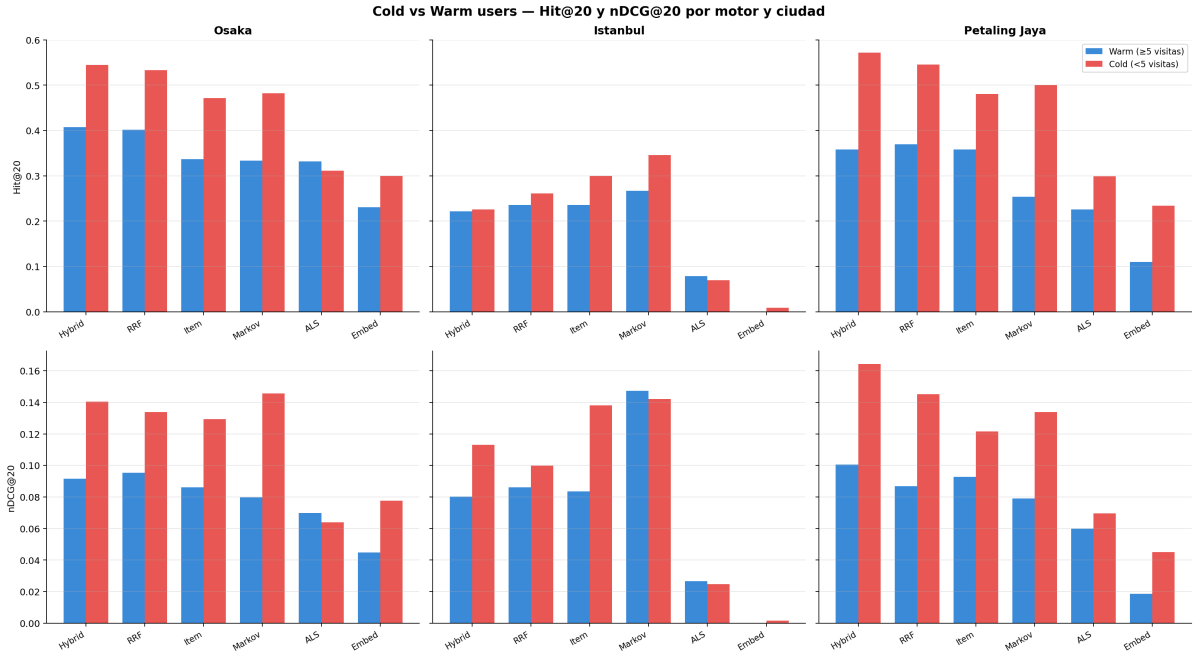


Figura 4.3: Comparación de rendimiento entre usuarios *cold* (< 5 visitas en train) y *warm* (≥ 5) por ciudad y motor.

4.3.2 Posicionamiento orientativo frente a la literatura

El análisis experimental de este trabajo se circunscribe a la tarea de *trail recommendation* bajo el protocolo `last_trail_user --fair`: dado el historial de un usuario, se reserva su último *trail* completo como conjunto de prueba y se evalúa si los POIs de ese trail aparecen entre las $K = 20$ posiciones más altas del ranking producido por cada motor. Las métricas reportadas son *Hit@20*, *Precision@20*, *Recall@20* y *nDCG@20*. La mayoría de los métodos de referencia en este dominio abordan, en cambio, la tarea de *next-POI prediction* (predecir el siguiente único punto de interés a partir del histórico de visitas

recientes de un usuario). Esta tarea se evalúa habitualmente con el protocolo *leave-one-out* y métricas a $K = 10$ (*Recall@10*, *nDCG@10*).

Existen, por tanto, tres diferencias estructurales que impiden trazar una comparación cuantitativa directa con los resultados de este TFG: (i) la formulación de la tarea es distinta (*trail recommendation* frente a *next-POI prediction*); (ii) el valor de K difiere, siendo las métricas a $K = 20$ sistemáticamente más altas que las correspondientes a $K = 10$ para un mismo sistema; y (iii) los conjuntos de datos de evaluación no son los mismos, ya que la mayor parte de la literatura utiliza las versiones de Foursquare correspondientes a Nueva York y Tokio (*NYC/TKY*), mientras que este trabajo emplea la versión *Semantic Trails* (2018) sobre Osaka, Istanbul y Petaling Jaya [16, 18].

La Tabla 4.3 recoge una selección representativa de métodos de la literatura, indicando la tarea abordada y la métrica y protocolo de evaluación empleados originalmente.

Tabla 4.3: Selección de métodos de referencia en la literatura de recomendación secuencial de POIs.

Método	**Ref.**	**Tarea original**	**Métrica / protocolo / datos**
PRME	[19]	*Next-new-POI* recommendation	Recall@K, leave-one-out, Foursquare NYC/LA
STGCN	[20]	*Next-POI* recommendation	Hit@K, nDCG@K, leave-one-out, Foursquare
STAN	[21]	*Next location* recommendation	Recall@K, leave-one-out, Foursquare NYC/TKY
GETNext	[22]	*Next-POI* recommendation	Recall@10, nDCG@10, leave-one-out, Foursquare NYC/TKY
STHGCN	[23]	*Next-POI* recommendation	Recall@K, nDCG@K, leave-one-out, Foursquare NYC/TKY
Este TFG	—	*Trail recommendation*	Hit@20, nDCG@20, `last_trail_user --fair`, Foursquare Osaka / Istanbul / PJ (2018)

NYC = Foursquare New York City; *TKY* = Foursquare Tokyo; *LA* = Los Angeles; *PJ* = Petaling Jaya. La fila *Este TFG* se incluye como referencia del protocolo propio y no es directamente comparable con ninguna de las filas anteriores.

La combinación de señales colaborativas, secuenciales y de contenido dentro de una arquitectura híbrida es la estrategia dominante en los sistemas de recomendación de POIs más competitivos [3, 9, 10]. Los métodos anteriores al aprendizaje profundo: modelos Markov, factorización matricial y métodos de métrica de ranking como PRME [19], sentaron las bases experimentales del área. Las arquitecturas más recientes, como los modelos basados en grafos espacio-temporales o en transformers aplicados a secuencias de POIs (STGCN [20], STAN [21], GETNext [22], STHGCN [23]), han ampliado la capacidad de capturar dependencias de largo alcance y contexto espacio-temporal, con ganancias sustanciales en sus experimentos originales respecto a los enfoques clásicos. Este TFG se sitúa, metodológicamente, en la capa de los sistemas híbridos modulares sin aprendizaje

profundo, más expresivo que los enfoques de un único modelo, pero con un margen de mejora identificado respecto a las arquitecturas de representación más avanzadas.

La ausencia de comparación numérica directa con los métodos anteriores responde a razones complementarias. Los modelos citados no disponen de código público reproducible compatible con el formato Foursquare 2018 y están diseñados para la tarea de *next-POI* (un único POI siguiente), no para *trail recommendation*; adaptar su arquitectura al protocolo propio requeriría un esfuerzo de la envergadura de un TFG adicional. A esto se suma que el protocolo de la literatura emplea *leave-one-out* sobre interacciones individuales con métricas Recall@K, mientras que este trabajo usa `last_trail_user --fair` con Hit@20; ejecutar los motores propios sobre NYC/TKY con ese protocolo mediría una tarea distinta para la que no fueron diseñados, produciendo una comparación metodológicamente engañosa. Por último, los datos de partida se eligieron antes de identificar NYC/TKY como los datasets más utilizados en la literatura; cuando se evaluó la posibilidad de cambiar, la plataforma Foursquare había introducido cambios en las variables del API que ya se estaban usando en el pipeline, por lo que se decidió mantener la coherencia con el dataset disponible y continuar con las tres ciudades ya procesadas. El siguiente paso natural sería alinear el protocolo de evaluación con el de la literatura sobre un subconjunto de datos comparable.

4.4 Síntesis crítica de resultados

Los tres bloques de resultados, tomados juntos, apuntan en la misma dirección. Que el baseline aleatorio obtenga cero en recuperación exacta en Osaka y Petaling Jaya confirma que la tarea tiene estructura explotable: bajo el protocolo empleado, el primer POI del trail de test se usa como semilla de entrada y queda excluido de la evaluación; el motor aleatorio sortea entre los miles de POIs no visitados por el usuario en entrenamiento, y con $k = 20$ la probabilidad de acertar alguno de los restantes ítems del trail es inferior al 1% por usuario, por lo que cero o casi cero aciertos sobre los ∼250 usuarios evaluados es el resultado esperado. Sin ese referente, cualquier diferencia entre motores sería difícil de interpretar. La necesidad de evaluar ranking y calidad de ruta por separado quedó clara al cruzar los resultados. El motor `markov` obtiene un *nDCG@20* de 0,097 en Petaling Jaya, pero sus rutas presentan tramos medios de cerca de 1,6 km, frente a 0,6 km del motor `hybrid` en la misma ciudad (Tabla B.3). Esa discrepancia solo es visible si se miran las dos dimensiones; reducir la evaluación al ranking habría ocultado ese contraste. El análisis por ciudad sugiere que la densidad y calidad de los datos condicionan de forma importante el rendimiento, en ocasiones tanto o más que la elección del modelo concreto. En Osaka, casi cualquier motor con señal colaborativa o secuencial funciona bien. En Istanbul, donde los trails son más cortos y los check-ins menos densos, solo `markov` mantiene resultados estables, porque opera sobre frecuencias de transición brutas y no necesita representaciones latentes densas para producir recomendaciones útiles.

5. Conclusiones y trabajos futuros

En este capítulo se recogen las principales conclusiones del trabajo a partir del sistema desarrollado y de los resultados experimentales obtenidos. Además, se exponen las limitaciones más relevantes identificadas durante el desarrollo y se plantean varias líneas de trabajo futuro que permitirían ampliar el alcance del proyecto o mejorar su rendimiento.

5.1 Conclusiones del trabajo

La arquitectura modular ha funcionado bien en la práctica. Separar motores, *scoring*, construcción de rutas y evaluación en capas independientes facilitó la iteración. Cuando `embed` mostraba un deterioro claro en Istanbul, el problema quedaba aislado sin afectar al resto del pipeline, y comparar nueve motores bajo exactamente el mismo protocolo no requirió duplicar código de evaluación.

La decisión de evaluar también calidad de ruta, y no solo *ranking*, cambió algunas conclusiones del proyecto. Motores que obtienen buenos valores de *nDCG@20* no siempre producen los itinerarios más compactos o adecuados temáticamente, y al revés. Esa dimensión solo es visible si se miden las dos perspectivas de forma sistemática; la literatura de recomendación de POIs suele detenerse en la primera.

Los datos han resultado ser un factor determinante. La disparidad de volumen entre Osaka (675.807 visitas) e Istanbul (161.977) no es solo cuantitativa, afecta al vocabulario de transiciones disponible para Markov, a la densidad de la matriz usuario–POI que necesita ALS y a la cobertura del espacio de embeddings. Esa asimetría explica en buena parte por qué la configuración óptima varía entre ciudades y por qué los resultados de Istanbul requieren una lectura más cautelosa que los de Osaka o Petaling Jaya.

El prototipo web cierra el ciclo del trabajo. Una petición real al endpoint `/multi-recommend` devuelve cuatro variantes de ruta en tiempos compatibles con un uso interactivo, lo que convierte la evaluación offline en algo comprobable de forma funcional y no solo numérica.

5.2 Limitaciones del trabajo

La limitación más concreta es la dependencia de los datos. El dataset contiene ruido propio del dominio, categorías poco informativas y, sobre todo, una cobertura muy dispar entre ciudades. Istanbul tiene menos de una cuarta parte de los check-ins de Osaka, los trails son más cortos y hay usuarios con muy pocas visitas registradas, lo que hace que el protocolo `last_trail_user` tenga menos casos evaluables y que motores como ALS o `embed` operen con señal insuficiente. Los resultados en esa ciudad deben interpretarse con esa limitación en mente. En Petaling Jaya se observa además un patrón de uso más cotidiano que turístico: categorías como *Office* y *Structures* se encuentran entre los tipos

de lugar más frecuentes, lo que puede llevar al sistema a incluir lugares no turísticos en las rutas recomendadas. Esta característica del dataset limita la interpretabilidad de los resultados para ese entorno.

La evaluación es offline y eso tiene consecuencias. El protocolo empleado es sólido para comparar motores entre sí, pero no responde a si un usuario real encontraría útil la ruta generada. Que el sistema recupere los POIs del trail de referencia no implica necesariamente que el itinerario resultante sea atractivo para alguien que visita esa ciudad por primera vez. Validar ese aspecto requeriría experimentos con usuarios reales que estaban fuera del alcance del proyecto.

La planificación de rutas se apoya en *nearest neighbor* con mejora local *2-opt*, que es suficiente para las longitudes consideradas en la evaluación pero no garantiza optimalidad global. Para rutas más largas o con restricciones explícitas de horario habría que explorar solvers más avanzados. Del mismo modo, la geometría real de calles y el trazado sobre mapa dependen de OSRM, un servicio externo cuya latencia o indisponibilidad no afecta al núcleo de recomendación pero sí a la experiencia final del prototipo.

El ajuste por ciudad introduce riesgo de sobreajuste local. Los hiperparámetros de Markov, embeddings y ALS se eligieron maximizando *nDCG@20* en validación sobre la misma distribución de datos que se usa para reportar los resultados finales. Esta es la limitación metodológica más relevante del trabajo: implica que los valores de rendimiento reportados son optimistas respecto al comportamiento real del sistema sobre datos completamente nuevos. Una partición temporal independiente para la evaluación final sería metodológicamente más rigurosa. Por ahora, el versionado de configuraciones y la fijación de semilla mitigan ese riesgo, pero no lo eliminan.

El proyecto tampoco explora arquitecturas de aprendizaje profundo, como transformers secuenciales o modelos sobre grafos espacio-temporales. Los resultados son competitivos frente a los métodos clásicos implementados, pero están lejos del límite superior del estado del arte en recomendación secuencial de POIs.

5.3 Líneas de trabajo futuro

Las extensiones más directas del trabajo se derivan de sus propias limitaciones.

La más inmediata sería incorporar modelos secuenciales más avanzados y reproducir la evaluación de al menos uno de ellos sobre los mismos datos y protocolo empleados en este trabajo. Los transformers aplicados a secuencias de POIs, como GETNext o STHGCN (descritos en el Capítulo 2), han mostrado ganancias sustanciales sobre métodos clásicos en sus experimentos originales. Tener una comparación homogénea daría un referente real sobre el margen de mejora disponible.

El sistema actual ignora la dimensión temporal de los datos. Como ilustra la Figura 5.1, los check-ins en Osaka presentan picos claros en horario de comida y durante el fin de semana, una estructura que el modelo no explota en absoluto. Incorporar franja horaria, día de la semana o afluencia estimada en el *scoring* podría mejorar la personalización

contextual sin requerir un rediseño del pipeline.

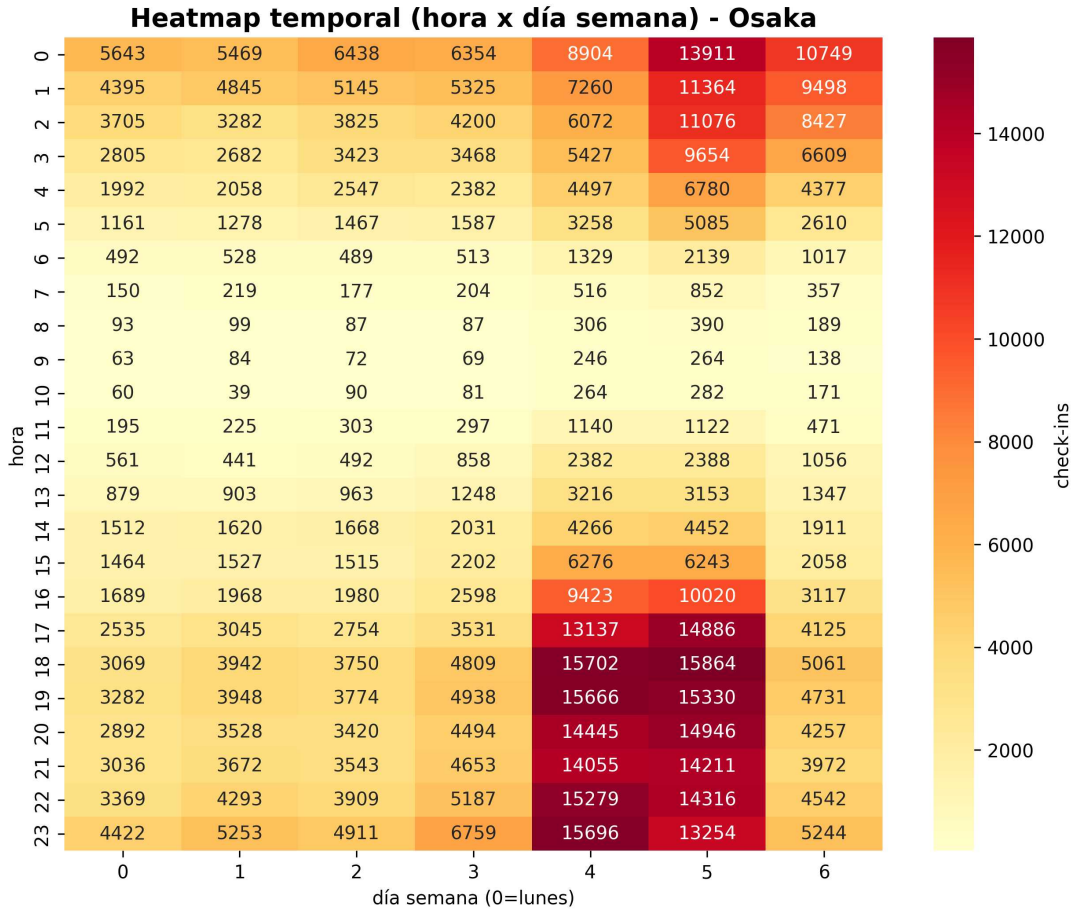


Figura 5.1: Patrón temporal hora × día de semana de los check-ins en Osaka.

Ampliar el sistema a más ciudades permitiría separar lo que es una propiedad del modelo de lo que es un artefacto del dataset. Con tres ciudades es difícil saber si el buen comportamiento de `hybrid` en Osaka se debe al diseño del sistema o a la riqueza de esos datos en particular. Ciudades con perfiles muy distintos (alta densidad turística, baja cobertura, distribución geográfica diferente) darían más soporte a cualquier conclusión sobre generalización.

La validación con usuarios reales es la deuda metodológica más importante. Las métricas offline permiten comparar motores entre sí, pero no responden a si alguien que llega a Osaka encontraría la ruta generada más útil que buscar en otra herramienta. Un estudio de preferencias o una evaluación A/B con rutas alternativas daría una señal directa sobre valor percibido que las métricas actuales no capturan.

Por último, la integración con un modelo de lenguaje conversacional es una extensión natural del prototipo. El *backend* ya expone toda la funcionalidad necesaria vía API; añadir un asistente que interprete consultas en lenguaje natural, llame a `/multi-recommend` y explique la ruta resultante no requeriría modificar la lógica de recomendación, sino añadir

una capa de interfaz sobre lo que ya existe.

5.4 Reflexión final

Lo más valioso del proyecto no ha sido ningún resultado concreto, sino haber completado la cadena entera: procesar datos con ruido, construir varios modelos que compiten entre sí bajo el mismo protocolo, identificar por qué unos funcionan mejor que otros según la ciudad, y hacer que todo eso sea accesible desde un navegador. Cada uno de esos pasos añade complejidad que no suele aparecer en los benchmarks de la literatura.

La decisión de medir calidad de ruta además de *ranking* no estaba planificada desde el inicio. Surgió al observar que los resultados numéricos no explicaban del todo las diferencias entre motores cuando se inspeccionaban las rutas visualmente. Esa tensión entre métricas de recuperación y utilidad práctica del itinerario refleja el problema de fondo en recomendación turística y, aunque este trabajo no lo resuelve definitivamente, sí lo aborda de forma sistemática.

El sistema tiene margen claro de mejora en modelado secuencial avanzado, evaluación con usuarios reales y cobertura de más ciudades. Eso facilita que una iteración futura pueda partir de algo concreto: código, datos, protocolos reproducibles y un prototipo funcional, sin necesidad de rediseñar el punto de partida.

6. Referencias Bibliográficas

[1] OECD, *OECD Tourism Trends and Policies 2024*. Paris: OECD Publishing, 2024, ISBN: 9789264698505. Accessed: Apr. 20, 2026. [Online]. Available: https://www.oecd.org/en/publications/oecd-tourism-trends-and-policies-2024_80885d8b-en/full-report.html

[2] European Commission. "Digital transition of tourism." Mobility and Transport, Accessed: Apr. 20, 2026. [Online]. Available: https://transport.ec.europa.eu/tourism/transition-eu-tourism/digital-transition-tourism_en

[3] Z. Wang, W. Höpken, and D. Jannach, "A survey on point-of-interest recommendations leveraging heterogeneous data," *Information Technology & Tourism*, vol. 27, pp. 29–73, 2025. DOI: 10.1007/s40558-024-00301-3 [Online]. Available: https://link.springer.com/article/10.1007/s40558-024-00301-3

[4] S. Zhang, Z. Luo, L. Yang, F. Teng, and T. Li, "A survey of route recommendations: Methods, applications, and opportunities," *Information Fusion*, vol. 108, p. 102 413, 2024. DOI: 10.1016/j.inffus.2024.102413 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S156625352400191X

[5] S. Halder, K. H. Lim, J. Chan, and X. Zhang, "A survey on personalized itinerary recommendation: From optimisation to deep learning," *Applied Soft Computing*, vol. 152, p. 111 200, 2024. DOI: 10.1016/j.asoc.2023.111200 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623012188

[6] O. A. S. Ibrahim, E. M. G. Younis, E. A. Mohamed, and W. N. Ismail, "Revisiting recommender systems: An investigative survey," *Neural Computing and Applications*, vol. 37, pp. 2145–2173, 2025. DOI: 10.1007/s00521-024-10828-5 [Online]. Available: https://link.springer.com/article/10.1007/s00521-024-10828-5

[7] P. Kar, M. Roy, and S. Datta, "Collaborative filtering and content-based systems," in *Recommender Systems: Algorithms and their Applications*, Springer, 2024. DOI: 10.1007/978-981-97-0538-2_3 [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-97-0538-2_3

[8] M. F. Aljunid, D. H. Manjaiah, M. K. Hooshmand, W. A. Ali, A. M. Shetty, and S. Q. Alzoubah, "A collaborative filtering recommender systems: Survey," *Neurocomputing*, vol. 617, p. 128 718, 2024. DOI: 10.1016/j.neucom.2024.128718 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231224014899

[9] T. F. Boka, Z. Niu, and R. B. Neupane, "A survey of sequential recommendation systems: Techniques, evaluation and future directions," *Information Systems*, vol. 125, p. 102 427, 2024. DOI: 10.1016/j.is.2024.102427 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437924000851

[10] L. Pan, W. Pan, M. Wei, H. Yin, and Z. Ming, "A survey on sequential recommendation," *Frontiers of Computer Science*, vol. 20, p. 2 003 606, 2025. DOI: 10.1007/s11704-025-41329-w [Online]. Available: https://link.springer.com/article/10.1007/s11704-025-41329-w

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013. DOI: 10.48550/arXiv.1301.3781 [Online]. Available: https://arxiv.org/abs/1301.3781

[12] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 263–272. DOI: 10.1109/ICDM.2008.22 [Online]. Available: https://yifanhu.net/PUB/cf.pdf

[13] C. Bauer, E. Zangerle, and A. Said, "Exploring the landscape of recommender systems evaluation: Practices and perspectives," *ACM Transactions on Recommender Systems*, vol. 2, no. 1, 11:1–11:31, 2024. DOI: 10.1145/3629170 [Online]. Available: https://christinebauer.eu/publications/bauer-2024-landscape/bauer-2024-landscape.pdf

[14] A. Jadon and A. Patil, "A comprehensive survey of evaluation techniques for recommendation systems," *Communications in Computer and Information Science*, 2024. DOI: 10.1007/978-3-031-71484-9_25 [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-71484-9_25

[15] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems*, vol. 20, no. 4, pp. 422–446, 2002. DOI: 10.1145/582415.582418 [Online]. Available: https://faculty.cc.gatech.edu/~zha/CS8803WST/dcg.pdf

[16] Foursquare. "Foursquare developer platform — places and mobility data." Fuente de los datos de check-ins y metadatos de puntos de interés utilizados en este proyecto (*Foursquare Semantic Trails 2018*), Accessed: Apr. 20, 2026. [Online]. Available: https://developer.foursquare.com/

[17] G. V. Cormack, C. L. A. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009, pp. 758–759. DOI: 10.1145/1571941.1572114 [Online]. Available: https://cormack.uwaterloo.ca/cormacksigir09-rrf.pdf

[18] W. Wongso, H. Xue, and F. D. Salim, "Massive-STEPS: Massive semantic trajectories for understanding POI check-ins — dataset and benchmarks," *arXiv preprint arXiv:2505.11239*, 2025. DOI: 10.48550/arXiv.2505.11239 [Online]. Available: https://arxiv.org/abs/2505.11239

[19] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan, "Personalized ranking metric embedding for next new POI recommendation," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, pp. 2069–2075. [Online]. Available: https://www.ijcai.org/Abstract/15/293

[20] H. Han et al., "STGCN: A spatial-temporal aware graph learning method for POI recommendation," in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2020. DOI: 10.1109/ICDM50108.2020.00124 [Online]. Available: https://ieeexplore.ieee.org/document/9338281

[21] Y. Luo, Q. Liu, and Z. Liu, "STAN: Spatio-temporal attention network for next location recommendation," in *Proceedings of The Web Conference 2021 (WWW)*, 2021. DOI: 10.1145/3442381.3449998 [Online]. Available: https://dl.acm.org/doi/10.1145/3442381.3449998

[22] S. Yang, J. Liu, and K. Zhao, "GETNext: Trajectory flow map enhanced transformer for next POI recommendation," in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2022. DOI: 10.1145/3477495.3531983 [Online]. Available: https://dl.acm.org/doi/10.1145/3477495.3531983

[23] X. Yan et al., "Spatio-temporal hypergraph learning for next POI recommendation," in *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2023. DOI: 10.1145/3539618.3591770 [Online]. Available: https://dl.acm.org/doi/10.1145/3539618.3591770

A. Reproducibilidad e implementación

A.1 Entorno de ejecución y dependencias

El proyecto requiere Python 3.11 o superior y Docker 24 o superior para la base de datos PostgreSQL 16. La Tabla A.1 recoge las dependencias de producción con sus versiones exactas, agrupadas por función.

Tabla A.1: Dependencias de producción (`requirements.txt`).

Grupo	**Paquete**	**Versión**
Core científico	`numpy`	1.26.4
	`pandas`	2.2.3
	`scipy`	1.13.1
	`scikit-learn`	1.6.1
	`joblib`	1.4.2
Base de datos	`psycopg[binary]`	3.2.11
API backend	`fastapi`	0.116.1
	`uvicorn`	0.35.0
	`pydantic`	2.11.1
HTTP y entorno	`python-dotenv`	1.1.1
	`requests`	2.32.5
	`httpx`	0.28.1
Modelos	`gensim`	4.3.3
	`implicit`	0.7.2
	`sentence-transformers`	3.3.1
Mapas y visualización	`folium`	0.20.0
	`matplotlib`	3.10.1
Utilidades	`tabulate`	0.9.0

Las dependencias de desarrollo (`requirements-dev.txt`) incluyen adicionalmente `pytest` 8.4.2, `pytest-cov` 7.0.0, `ruff` 0.13.0, `black` 25.1.0 e `ipykernel` 6.29.5.

Variables de entorno

Antes de ejecutar cualquier módulo del proyecto es necesario crear un fichero `.env` en la raíz del repositorio (plantilla disponible en `.env.example`). Las variables requeridas se recogen en la Tabla A.2.

Tabla A.2: Variables de entorno requeridas (`.env`).

Variable	Descripción
`FOURSQUARE_API_KEY`	Clave de la API Foursquare Places. Requerida únicamente en el paso ETL `03_fetch_pois.py`.
`POSTGRES_DSN`	Cadena de conexión completa utilizada por todos los módulos `src/`. Ejemplo: `postgresql://tfg:tfgpass@localhost:55432/tfg_routes`.
`POSTGRES_USER`	Usuario PostgreSQL (leído por Docker Compose).
`POSTGRES_PASSWORD`	Contraseña PostgreSQL (leída por Docker Compose).
`POSTGRES_DB`	Nombre de la base de datos (leído por Docker Compose).
`PGADMIN_DEFAULT_EMAIL`	Correo de acceso a pgAdmin (interfaz web en el puerto 8080).
`PGADMIN_DEFAULT_PASSWORD`	Contraseña de pgAdmin.

A.2 Configuración del sistema y artefactos entrenados

La configuración del sistema se centraliza en `configs/recommender.toml`, con *overrides* por ciudad en `configs/recommender_<city_qid>.toml`. La Tabla A.3 recoge los hiperparámetros más relevantes de la configuración global por defecto. Los parámetros de ALS (`factors`, `iterations`) se sobreescriben en cada ciudad: Osaka y Petaling Jaya usan `factors=64` e `iterations=15`; Istanbul usa `factors=128` e `iterations=15`. Los valores operativos por ciudad se recogen en la Tabla A.4.

Tabla A.3: Hiperparámetros principales del sistema (`recommender.toml`).

Sección	Parámetro	Valor
`[embeddings]`	`vector_size`	300
	`window`	12
	`epochs`	40
	`negative`	15
	`seed`	42
`[als]`	`factors`	256
	`regularization`	0.005
	`iterations`	60
	`alpha`	60.0
`[route_planner]`	`candidate_pool`	1000
	`max_per_category`	2
	`distance_weight`	0.35
	`diversity_bonus`	0.06

Los pesos del motor híbrido para el escenario `user_current` (historial e input de usuario disponibles) son `[0.08, 0.30, 0.27, 0.15, 0.20]` en el orden content/item/markov/embed/als, coincidentes con los valores globales de la Tabla 3.2. En el escenario `cold_start` (sin historial), el peso recae sobre el motor de contenido: `[0.60, 0.20, 0.20, 0.00, 0.00]`.

Diferencias de configuración por ciudad

Cada ciudad dispone de un fichero de *override* (`configs/recommender_<city_qid>.toml`) que ajusta los parámetros globales según las características del entorno. La Tabla A.4 recoge las diferencias más relevantes.

Tabla A.4: Principales diferencias de configuración entre ciudades respecto al valor global.

Parámetro	Global	Osaka	Istanbul	PJ
`als.factors`	256	64	128	64
`als.iterations`	60	15	15	15
`embeddings.context_n`	3	2	1	3
`route_planner.candidate_pool`	1000	300	600	600
`hybrid.user_current[markov]`	0.15	0.27	0.40	0.27
`markov.backoff`	—	0.30	0.25	0.25

Nota: La columna **Global** refleja los valores del fichero base `recommender.toml` antes de aplicar overrides por ciudad. Los valores operativos del escenario `user_current` tras optimización se recogen en la Tabla 3.2 (Petaling Jaya) y en la Sección B.8 (Osaka e Istanbul).

La reducción de `als.factors` en Osaka y Petaling Jaya se justifica por el mejor rendimiento empírico de modelos más pequeños en entornos con alta dispersión de usuarios. Istanbul aumenta el peso de Markov en el escenario `user_current` (0.40 frente al 0.15 global) porque es el motor más estable en esa ciudad. El `context_n=1` de Istanbul refleja que sus secuencias de visita son más cortas. El parámetro `markov.backoff` refleja otra decisión de ajuste por ciudad. Osaka usa 0.30, frente a 0.25 en Istanbul y Petaling Jaya. Con mayor volumen absoluto de visitas, el corpus de Osaka contiene más pares de transición distintos y menos repeticiones por par; en consecuencia, las frecuencias de orden-2 son más dispersas y el modelo necesita suavizar más hacia orden-1 para generalizar correctamente. En Istanbul y Petaling Jaya el catálogo de transiciones es más concentrado, de modo que las frecuencias de orden-2 resultan más representativas y el backoff puede ser más bajo.

Ajuste de hiperparámetros por barrido discreto

Los valores finales de pesos híbridos, factores ALS, `context_n`, `backoff` y restricciones del planificador de rutas se obtuvieron mediante barrido discreto por ciudad sobre espacios de búsqueda acotados. El proyecto incluye seis scripts especializados: `tune_hybrid.py`, `tune_markov.py`, `tune_als.py`, `tune_embeddings_scoring.py`, `tune_route_planner.py`

y `tune_all.py`. En términos generales, las configuraciones se ordenan por *nDCG@20* y se desempatan con *MRR*; en el caso del planificador de rutas, la selección se realiza con una función objetivo de calidad de ruta. Los resultados se almacenan en `data/reports/tune_*.json` y los mejores valores se vuelcan manualmente a los ficheros de configuración por ciudad correspondientes. Los artefactos entrenados se almacenan en `src/recommender/cache/` con los nombres `word2vec_<city_qid>.joblib` (embeddings secuenciales) y `als_<city_qid>.joblib` (filtrado colaborativo), donde `<city_qid>` es el identificador Wikidata de la ciudad (`Q35765` Osaka, `Q406` Istanbul, `Q864965` Petaling Jaya).

A.3 Comandos principales de ejecución

Base de datos. PostgreSQL 16 y pgAdmin se levantan mediante Docker Compose:

```
docker compose up -d
```

El fichero `docker-compose.yml` define dos servicios: `db` (PostgreSQL 16, imagen `postgres:16`, container `tfg_postgres`, puerto externo **55432** → 5432, volumen persistente `pgdata`) y `pgadmin` (imagen `dpage/pgadmin4:8`, container `tfg_pgadmin`, acceso web en el **puerto 8080**). Las credenciales se leen del fichero `.env`. El puerto no estándar 55432 evita conflictos con instalaciones locales de PostgreSQL.

Pipeline ETL. Los ocho pasos de limpieza, enriquecimiento y carga se ejecutan en orden desde la raíz del repositorio:

```
python -m src.etl.01_clean_std
python -m src.etl.02_extract_ids
python -m src.etl.03_fetch_pois       # requiere FOURSQUARE_API_KEY en .env
python -m src.etl.04_normalize_pois
python -m src.etl.05_label_categories
python -m src.etl.05_run_bert_classifier
python -m src.etl.06_impute_pois
python -m src.etl.07_diagnostics
python -m src.etl.08_load_postgres    # carga final en PostgreSQL
```

La Tabla A.5 describe brevemente la responsabilidad de cada paso.

El criterio de descartar POIs sin coordenadas en lugar de imputar valores aproximados responde a un requisito de la fase de ruta. El planificador calcula distancias reales mediante Haversine, por lo que una coordenada sintetizada corrompería los cálculos de distancia por tramo y haría inútil la heurística de selección espacial. Los atributos de precio y valoración, en cambio, se usan como señales de filtrado y ranking, no como posiciones en el espacio, por lo que admiten imputación por mediana de categoría sin afectar la coherencia geográfica de las rutas.

Tabla A.5: Descripción de los pasos del pipeline ETL.

#	Script	Descripción
01	`01_clean_std.py`	Normaliza prefijos (`wd:`, `foursquare:`), parsea marcas de tiempo y filtra los registros a las tres ciudades objetivo.
02	`02_extract_ids.py`	Extrae y enumera identificadores únicos de venues, usuarios y ciudades.
03	`03_fetch_pois.py`	Consulta la API Foursquare Places para obtener coordenadas, rating, nombre y jerarquía de categorías de cada POI. Requiere `FOURSQUARE_API_KEY`.
04	`04_normalize_pois.py`	Valida coordenadas geoespaciales y aplana la jerarquía de categorías Foursquare.
05a	`05_label_categories.py`	Aplica etiquetado manual de categorías principales.
05b	`05_run_bert_classifier.py`	Predice la franja de precio (`price_tier`) con un clasificador basado en `sentence-transformers` para los POIs sin dato explícito de Foursquare. Esta es la razón de la dependencia `sentence-transformers==3.3.1` en `requirements.txt`.
06	`06_impute_pois.py`	Enriquece cada POI con las etiquetas de precio del clasificador (paso 05b) como fuente primaria; imputa `price_tier`, `rating` y `total_ratings` por la mediana de su categoría cuando no hay dato explícito de Foursquare. Los POIs sin coordenadas válidas se descartan del conjunto.
07	`07_diagnostics.py`	Genera reportes de calidad de datos: cobertura de categorías, conteos por ciudad y distribución de POIs.
08	`08_load_postgres.py`	Carga final en PostgreSQL: aplica el esquema SQL y realiza el *upsert* de visitas, POIs y categorías.

Entrenamiento de artefactos. Los embeddings y el modelo ALS se entrenan por ciudad (ejemplo para Osaka, `Q35765`):

```
python -m src.recommender.train_embeddings --city-qid Q35765
python -m src.recommender.train_als --city-qid Q35765
```

Evaluación experimental. La evaluación de ranking y de calidad de ruta, y el benchmark completo:

```
python -m src.recommender.eval.evaluate --city-qid Q35765 --fair
python -m src.recommender.eval.evaluate_routes --city-qid Q35765
python -m src.recommender.benchmark_3cities
```

Arranque del sistema. El backend FastAPI y el frontend estático se arrancan desde la raíz:

```
uvicorn src.recommender.api:app --port 8000
python -m http.server 8001 --directory frontend/
```

A.4 Estructura del repositorio y módulos principales

La organización de directorios se describe en la Sección 3.3.1. La Tabla A.6 detalla los módulos clave dentro de `src/recommender/`, que constituye el núcleo del sistema.

Tabla A.6: Módulos principales de `src/recommender/`.

Módulo	**Responsabilidad**
`scorer.py`	Integración de motores, reranking y fusión de puntuaciones.
`route_planner.py`	Selección greedy de POIs con restricciones espaciales y de diversidad.
`route_builder.py`	Ordenación final del itinerario y exportación GeoJSON.
`multi_route_service.py`	Contrato multi-variante (History / Inputs / Location / Full).
`api.py`	Exposición del sistema vía FastAPI (endpoints REST).
`train_embeddings.py`	Entrenamiento del modelo Word2Vec sobre trayectorias de visita.
`train_als.py`	Entrenamiento del modelo ALS de filtrado colaborativo implícito.
`eval/evaluate.py`	Evaluación offline de ranking con protocolo `last_trail_user`.
`eval/evaluate_routes.py`	Evaluación de calidad de ruta (distancia, diversidad, adecuación).
`benchmark_3cities.py`	Ejecución sistemática del benchmark sobre las tres ciudades.

A.5 Resumen de endpoints de la API

El backend expone seis endpoints REST a través de FastAPI. La Tabla A.7 los recoge con sus métodos y funciones principales.

Tabla A.7: Endpoints de la API REST (`src/recommender/api.py`).

Método	**Ruta**	**Descripción**
GET	`/health`	Comprueba que el servicio está activo.
POST	`/recommend`	Recomendación con un único motor (`mode`: hybrid, content, item, markov, embed, als). Devuelve lista de POIs y, si `build_route=true`, una ruta ordenada.
POST	`/multi-recommend`	Recomendación multi-variante. Devuelve hasta cuatro rutas: History, Inputs, Location y Full.
POST	`/saved-routes`	Guarda una ruta generada en la base de datos.
GET	`/saved-routes`	Lista las rutas guardadas para una ciudad y usuario.
DELETE	`/saved-routes`	Elimina rutas guardadas según criterios de ciudad y usuario.

Los campos comunes del *request* incluyen `city_qid` (identificador de ciudad), `user_id` (opcional), `k` (número de POIs, 1–200), `lat`/`lon` (localización inicial), `prefs` (preferencias categóricas) y `build_route` (activa la construcción de ruta ordenada).

A.6 Sistema de categorías e intenciones turísticas

El módulo `category_intents.py` implementa un mapa entre las categorías de la taxonomía Foursquare y las intenciones turísticas semánticas que el sistema reconoce. Este componente es el responsable de que preferencias expresadas en lenguaje natural como `"museum"`, `"culture"` o `"free"` puedan traducirse a señales operativas de recomendación.

El sistema contempla ocho intenciones principales: *food*, *culture*, *nature*, *museum*, *park*, *transport*, *nightlife* y *entertainment*. La asignación de cada categoría Foursquare a una intención combina reglas manuales con una clasificación basada en similitud de texto.

El módulo opera en dos modos de forma independiente para cada preferencia declarada:

- **Modo *soft***: añade un *boost* de puntuación ($\delta_{\text{cat}} = 0.20$) a los candidatos cuya intención coincide con la preferencia. El candidato no queda excluido aunque no coincida.
- **Modo *strict***: filtra directamente los candidatos que no pertenecen a la intención solicitada, reduciendo el pool de candidatos antes de puntuar.

Las correcciones manuales para categorías ambiguas o mal clasificadas se recogen en `configs/category_intent_overrides.json`, que permite anular la asignación automática a nivel de categoría concreta sin modificar el código. La Tabla A.8 muestra ejemplos representativos del mapeo.

Tabla A.8: Ejemplos de mapeo categoría Foursquare → intención turística.

Categoría Foursquare	**Intención**
Japanese Restaurant, Café, Sake Bar	*food*
Art Museum, History Museum	*museum*
Historic and Protected Site, Temple	*culture*
Park, Garden, Nature Reserve	*nature* / *park*
Bar, Nightclub	*nightlife*
Train Station, Subway	*transport*

B. Resultados ampliados y material complementario

B.1 Estadísticas descriptivas del dataset

La Tabla B.1 amplía la información de la Tabla 3.1 del Capítulo 3 con las cinco categorías más frecuentes por ciudad, extraídas del diagnóstico ETL generado por `07_diagnostics.py`.

Tabla B.1: Estadísticas del *dataset* por ciudad con top-5 categorías (*Foursquare Semantic Trails Dataset*, 2018).

Ciudad (QID)	Detalle
Osaka (Q35765)	Visitas: 675.807 • Usuarios: 25.467 • Trails: 97.697
	Top-5: Sake Bar (1.183), Japanese Restaurant (1.133), Café (1.032), Ramen Restaurant (664), Convenience Store (607)
Istanbul (Q406)	Visitas: 161.977 • Usuarios: 15.853 • Trails: 35.648
	Top-5: Café (602), Turkish Restaurant (249), Restaurant (207), Hotel (160), Historic and Protected Site (152)
Petaling Jaya (Q864965)	Visitas: 460.584 • Usuarios: 18.346 • Trails: 82.951
	Top-5: Office (1.235), Malay Restaurant (1.162), Café (1.010), Structure (930), Asian Restaurant (892)

Istanbul presenta el menor volumen de visitas y trails, lo que explica estructuralmente su comportamiento diferenciado en la evaluación experimental. Petaling Jaya tiene más visitas que Istanbul pero una categoría dominante muy diferente (Office/Structure), lo que refleja un patrón de uso más cotidiano frente al turístico de Osaka.

B.2 Tablas completas de evaluación de ranking

Las tablas numéricas completas de métricas de ranking para los nueve motores y las tres ciudades se encuentran en la Tabla 4.1 del Capítulo 4. Como apoyo visual, la Figura B.1 ofrece una vista de calor global que facilita la comparación entre motores y métricas.

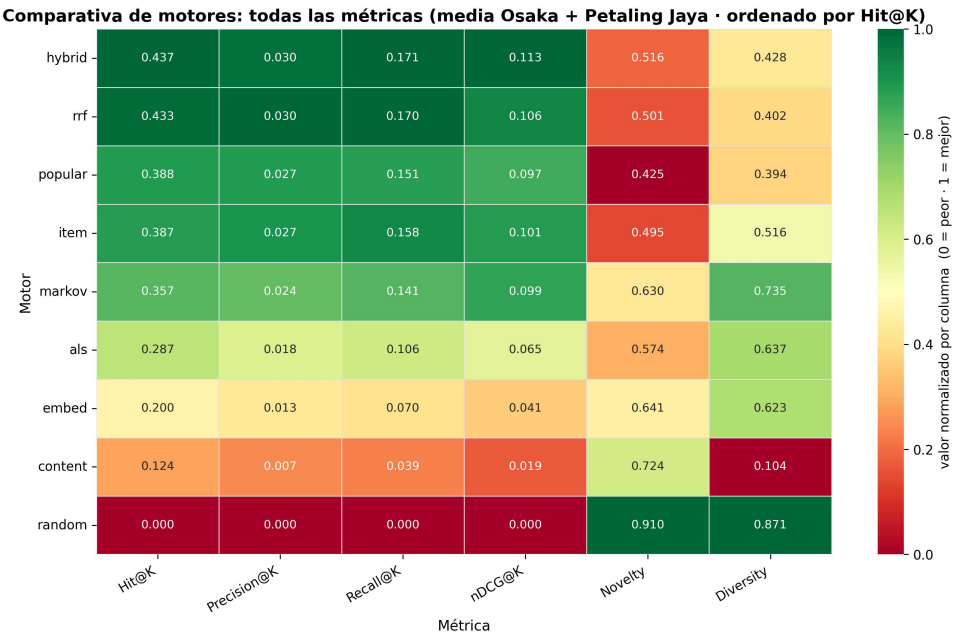


Figura B.1: Heatmap completo motor × métrica ordenado por *Hit@20* descendente. Media calculada sobre Osaka y Petaling Jaya (Istanbul excluida por volumen). Las celdas más oscuras corresponden a los valores más altos por métrica.

Tabla B.2: Desglose numérico *cold* vs. *warm*: *Hit@20* por motor y ciudad (N_c/N_w: Osaka 90/199 · Istanbul 115/140 · Petaling Jaya 77/173).

	Osaka		**Istanbul**		**Pet. Jaya**	
Motor	cold	warm	cold	warm	cold	warm
embed	0.300	0.231	0.009	0.000	0.234	0.110
item	0.472	0.337	0.300	0.236	0.481	0.358
markov	0.482	0.333	0.346	0.267	0.500	0.254
als	0.311	0.332	0.070	0.079	0.299	0.225
hybrid	0.544	0.407	0.226	0.221	0.571	0.358
content	0.211	0.121	0.009	0.036	0.143	0.081

B.3 Tablas completas de calidad de ruta

Los indicadores de calidad de ruta para Osaka se presentan en la Tabla 4.2 del Capítulo 4. Las tablas siguientes recogen los mismos indicadores para Petaling Jaya e Istanbul, con el mismo protocolo experimental (`last_trail_user`, $k = 8$, 200 casos).

Tabla B.3: Calidad de ruta en Petaling Jaya (`Q864965`).

Motor	Total km	Tramo medio	% muy cortos	% muy largos	Cat. match
content	8.533	1.255	0.000	0.038	0.383
item	5.423	0.703	0.000	0.006	0.123
markov	8.439	1.587	0.039	0.070	0.111
embed	4.814	0.602	0.000	0.000	0.074
als	4.934	0.617	0.000	0.000	0.128
hybrid	4.507	0.563	0.000	0.000	0.185

Tabla B.4: Calidad de ruta en Istanbul (`Q406`).

Motor	Total km	Tramo medio	% muy cortos	% muy largos	Cat. match
content	6.170	0.774	0.000	0.002	0.273
item	7.199	0.937	0.000	0.004	0.116
markov	4.149	0.765	0.036	0.015	0.108
embed	3.165	0.396	0.000	0.001	0.055
als	6.934	0.867	0.000	0.002	0.104
hybrid	5.824	0.728	0.000	0.001	0.143

B.4 Métricas de categoría completas

El Capítulo 4 justifica el uso de métricas de categoría (*cat_hit@20*, *cat_nDCG@20*) y describe cualitativamente sus resultados. La Tabla B.5 ofrece los valores numéricos completos para los nueve motores y las tres ciudades, calculados con el mismo protocolo `last_trail_user --fair`.

Tabla B.5: Métricas de categoría por motor y ciudad (*cat_hit@20* / *cat_nDCG@20*). Protocolo `last_trail_user`, `--fair`, $k = 20$.

	Osaka		Istanbul		Petaling Jaya	
Motor	**cat-Hit**	**cat-nDCG**	**cat-Hit**	**cat-nDCG**	**cat-Hit**	**cat-nDCG**
als	**0.758**	**0.293**	0.482	0.195	**0.692**	0.231
hybrid	0.751	0.287	0.549	0.248	0.676	**0.280**
markov	0.708	0.268	**0.579**	**0.328**	0.667	0.258
item	0.708	0.263	0.536	0.239	0.660	0.268
rrf	0.702	0.262	0.529	0.243	0.648	0.266
embed	0.595	0.223	0.208	0.061	0.532	0.169
popular	0.574	0.237	0.400	0.166	0.560	0.239
random	0.439	0.087	0.361	0.124	0.404	0.091
content	0.391	0.173	0.220	0.129	0.328	0.140

Destacan varias lecturas complementarias a las métricas exactas del Capítulo 4. En Osaka, `als` lidera en adecuación categórica (*cat-hit*=0.758) por encima de `hybrid` (0.751), aunque en recuperación exacta de POI ocurre lo contrario, `als` obtiene 0.325 frente a 0.450 de `hybrid`. Esto confirma que ALS aprende bien el tipo de experiencia que le gusta al usuario, pero no necesariamente el lugar concreto. En Istanbul, `markov` domina también las métricas de categoría (*cat-nDCG*=0.328), lo que refuerza que su ventaja en esa ciudad no es solo en ranking exacto, sino en comprensión del tipo de lugar esperado. El motor `content` muestra los peores valores de categoría a pesar de estar basado en similitud semántica, lo que refleja que su perfil de usuario TF-IDF no captura bien la distribución real de preferencias.

B.5 Resultados desglosados por ciudad

Los resultados de ranking por ciudad se presentan de forma completa en la Tabla 4.1. Como se analiza en el Capítulo 4, Istanbul presenta un comportamiento estructuralmente diferente al de Osaka y Petaling Jaya. El menor volumen de check-ins y la mayor dispersión geográfica de sus POIs provocan que motores como `embed` y `als` colapsen a métricas casi nulas, mientras que `markov` emerge como el motor más estable en esa ciudad. Esta asimetría justifica la exclusión de Istanbul del cálculo de medias en las comparaciones globales y en el posicionamiento orientativo frente a la literatura.

Los resultados de calidad de ruta muestran un patrón similar. Istanbul presenta distancias totales generalmente menores que Osaka y Petaling Jaya, lo que refleja que la menor cobertura de POIs concentra las rutas en zonas más reducidas. El motor `embed` es especialmente afectado, produciendo rutas muy compactas pero con baja adecuación temática (*cat. match* = 0.055), resultado coherente con su colapso en ranking.

B.6 Figuras y ejemplos adicionales

El esquema relacional completo del sistema se recoge en el Listado B.2. Las tres tablas principales son definidas por `sql/schema.sql`; la tabla `saved_routes`, utilizada para persistir rutas desde la capa de producto, se crea dinámicamente mediante el propio backend. La descripción funcional de cada tabla puede consultarse en la Sección 3.3.2.

Tabla `saved_routes`. Las tres tablas anteriores se definen en `sql/schema.sql`. La tabla `saved_routes`, utilizada para persistir rutas desde la capa de producto, es creada dinámicamente por el backend (`api.py`) al arrancar si no existe:

```
CREATE TABLE IF NOT EXISTS saved_routes (
    id          BIGSERIAL PRIMARY KEY,
    user_id     BIGINT,
    city_qid    TEXT,
    route_type  TEXT NOT NULL,          -- 'history' | 'inputs' | 'location' | 'full'
    source      TEXT NOT NULL DEFAULT 'frontend',
    payload     JSONB,                  -- ruta completa serializada en JSON
```

```
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE INDEX IF NOT EXISTS idx_saved_routes_user_created
    ON saved_routes(user_id, created_at DESC);
CREATE INDEX IF NOT EXISTS idx_saved_routes_city_created
    ON saved_routes(city_qid, created_at DESC);
```

Listado B.1: Esquema de la tabla `saved_routes` (gestionada por `api.py`).

El campo `payload` contiene la ruta completa en formato JSONB, incluyendo la lista de POIs ordenados, sus metadatos y la variante generada. Los índices sobre `(user_id, created_at)` y `(city_qid, created_at)` permiten recuperar eficientemente las rutas guardadas por usuario o por ciudad en orden cronológico inverso.

```
CREATE TABLE IF NOT EXISTS visits (
  trail_id       integer,
  user_id        integer,
  venue_id       text,
  venue_category text,
  venue_schema   text,
  venue_city     text,
  venue_country  text,
  "timestamp"    timestamptz,
  trail_id_orig  text,
  user_id_orig   text
);
CREATE INDEX IF NOT EXISTS idx_visits_venue ON visits(venue_id);
CREATE INDEX IF NOT EXISTS idx_visits_city  ON visits(venue_city);

CREATE TABLE IF NOT EXISTS pois (
  fsq_id           text PRIMARY KEY,
  name             text,
  lat              double precision,
  lon              double precision,
  city             text,
  city_qid         text,
  country          text,
  rating           double precision,
  price_tier       integer,
  total_ratings    integer,
  primary_category text,
  is_free          boolean DEFAULT false
);
CREATE INDEX IF NOT EXISTS idx_pois_city        ON pois(city);
CREATE INDEX IF NOT EXISTS idx_pois_city_qid    ON pois(city_qid);
CREATE INDEX IF NOT EXISTS idx_pois_primary_cat ON pois(primary_category);

CREATE TABLE IF NOT EXISTS poi_categories (
  fsq_id        text REFERENCES pois(fsq_id) ON DELETE CASCADE,
  category_id   text,
  category_name text,
  PRIMARY KEY (fsq_id, category_id)
);
```

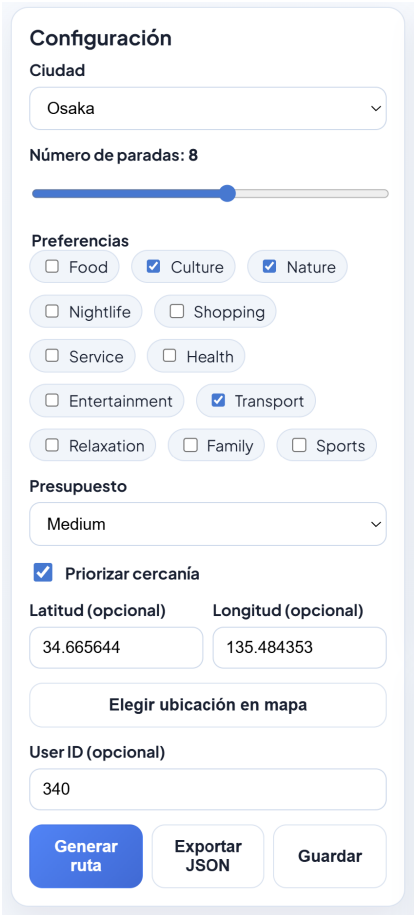
Listado B.2: Esquema relacional principal (`sql/schema.sql`).

B.7 Capturas adicionales del prototipo web y rutas generadas

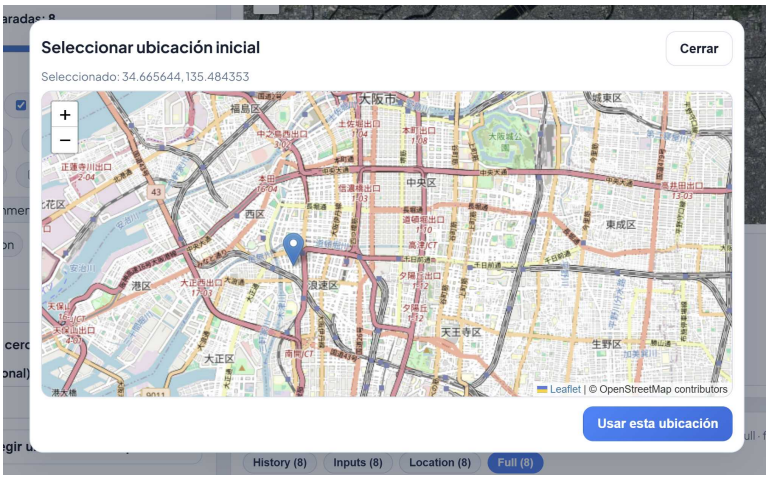
Además de la configuración básica y la visualización de rutas descrita en la Sección 3.3.4, el frontend incorpora las siguientes funcionalidades avanzadas:

- **Modo pantalla completa**: el mapa puede expandirse a pantalla completa para una mejor inspección visual de la ruta.
- **Panel de rutas guardadas**: muestra las rutas persistidas en el backend para el usuario y ciudad activos. Si el backend no está disponible, el panel hace *fallback* automático a `localStorage` del navegador.
- **Exportación a JSON**: permite descargar la ruta activa como fichero JSON estructurado, incluyendo la lista de POIs con coordenadas, categorías y puntuaciones.
- **Geometría por calles (OSRM)**: los segmentos rectos entre POIs pueden sustituirse por trazado real de vías urbanas mediante una superposición calculada a través de OSRM, mejorando la interpretabilidad del itinerario.

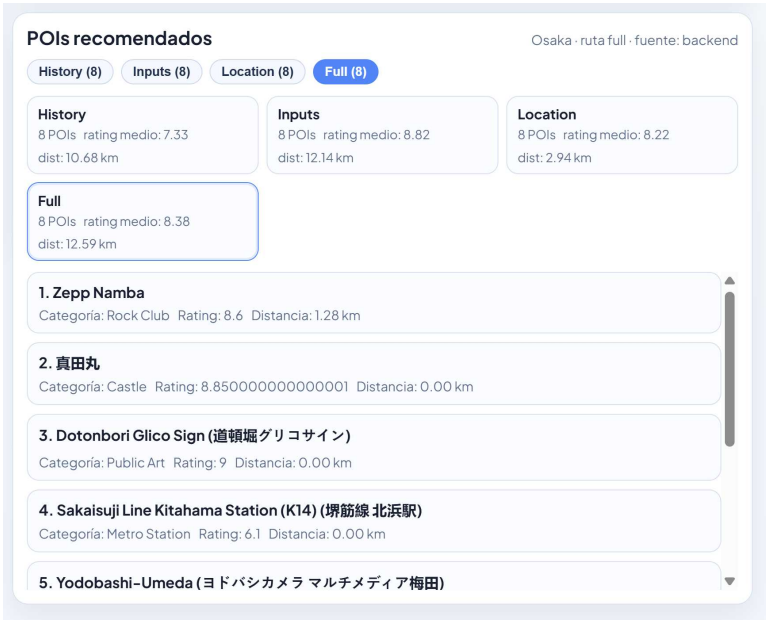
La Figura B.2 resume el flujo de generación de una ruta en la web. Primero se fija la configuración de la petición, después se selecciona el punto inicial en el mapa y, por último, se observa la lista de POIs recomendados que se usará para construir el itinerario final.



(a) Configuración inicial de la petición.



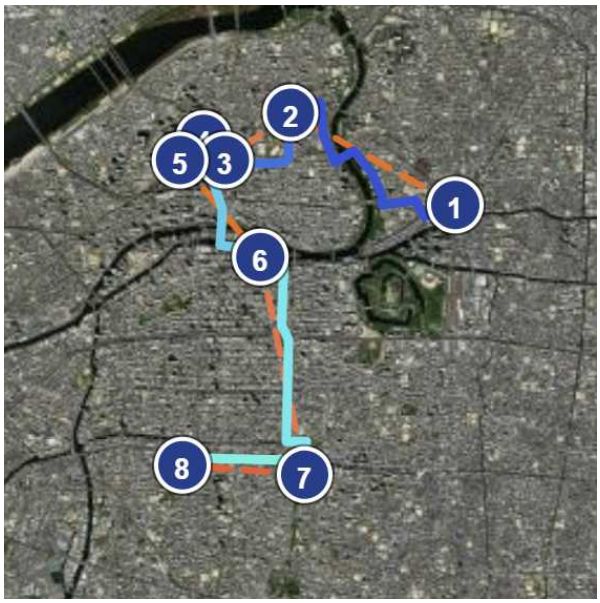
(b) Selección del punto inicial en el mapa.



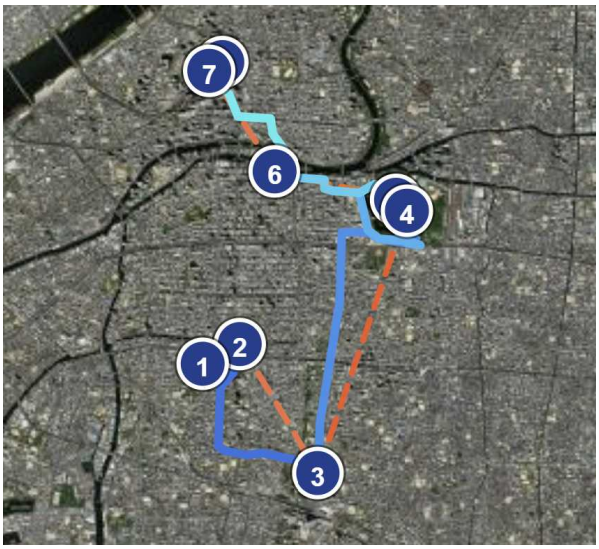
(c) POIs recomendados antes del trazado final.

Figura B.2: Proceso de preparación de una ruta en el prototipo web: configuración, selección del punto inicial y obtención de candidatos recomendados.

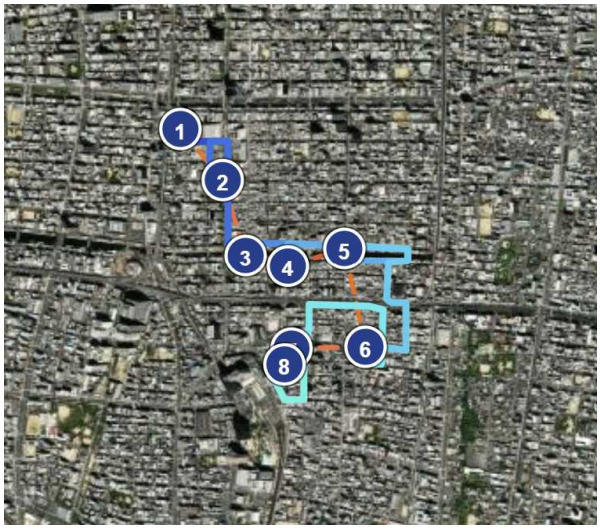
La Figura B.3 compara las cuatro variantes de salida del esquema de respuesta multi-ruta. `history` refleja una ruta basada en historial previo; `full` combina todas las señales disponibles; `inputs` prioriza las preferencias explícitas introducidas en la petición; y `location` genera un recorrido más local, con paradas más próximas al punto inicial y menor dispersión espacial.



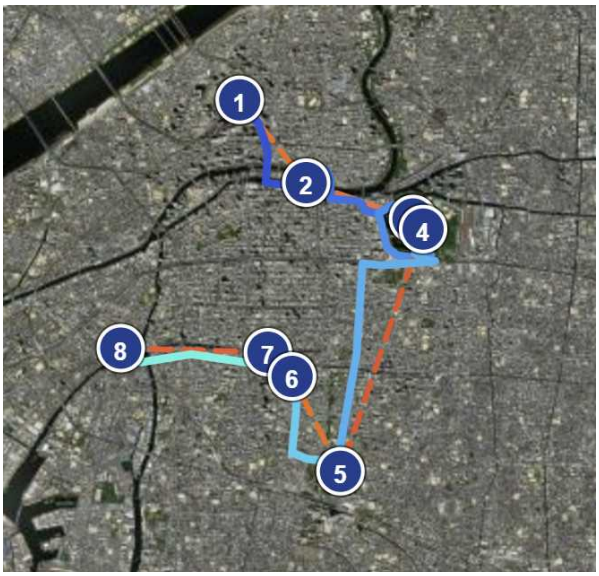
(a) Variante `history`.



(b) Variante `full`.



(c) Variante `location` *(recorrido más corto y concentrado alrededor del inicio).*



(d) Variante `inputs`.

Figura B.3: Comparativa visual de las variantes `history`, `full`, `location` e `inputs` generadas por el sistema.

B.8 Notas metodológicas sobre evaluación y configuración del híbrido

Pesos del motor híbrido por ciudad. La Tabla 3.2 del cuerpo muestra los pesos de la configuración global por defecto del sistema. Los escenarios *user_current* y

user_only se ajustan por ciudad mediante optimización offline, por lo que los valores operativos difieren de los mostrados. Osaka emplea [0.09, 0.25, 0.27, 0.18, 0.21] e Istanbul [0.05, 0.35, 0.40, 0.05, 0.15] (orden: *content, item, markov, embed, als*). Petaling Jaya coincide con los valores globales. Los escenarios *current_only*, *embed_or_als* y *cold_start* son idénticos en las tres ciudades. El resto de parámetros que varían por ciudad se recogen en la Tabla A.4.

Escenario `trail_current`. Este escenario se activa exclusivamente durante la construcción greedy del itinerario. En cada paso, el motor recibe el último POI añadido a la ruta como POI de referencia y pondera las señales secuenciales (Markov y embeddings) por encima de la personalización a largo plazo (ALS). Los pesos por ciudad, optimizados mediante `tune_hybrid.py`, son los siguientes (orden: *content, item, markov, embed, als*):

Ciudad	content	item	markov	embed	als
Osaka	0.00	0.29	0.39	0.20	0.13
Istanbul	0.06	0.29	0.48	0.09	0.08
Petaling Jaya	0.00	0.39	0.56	0.00	0.05

N de usuarios evaluados para el motor Markov (†). El motor `markov` requiere un POI de referencia como semilla para generar recomendaciones, por lo que solo puede evaluarse sobre usuarios cuyo historial incluye al menos una transición secuencial registrada en el dataset. Bajo el protocolo `last_trail_user`, esto reduce el conjunto de usuarios evaluados respecto al resto de motores:

Ciudad	Markov (N)	Resto de motores (N)	Diferencia
Osaka	271	289	−18
Istanbul	235	255	−20
Petaling Jaya	204	250	−46

Las métricas del motor `markov` en la Tabla 4.1 se calculan sobre este subconjunto reducido y no son estrictamente comparables con las del resto de motores en términos de población evaluada.

Reproducibilidad por usuario en la evaluación. Para garantizar reproducibilidad sin imponer el mismo estado aleatorio a todos los usuarios, el evaluador calcula una semilla individual por caso mediante `_stable_case_seed()`. El identificador de ciudad y de usuario se codifican con Blake2b (8 bytes) y el resultado se combina con la semilla global. De este modo, el motor `random` produce exactamente el mismo orden de candidatos en cada ejecución para cada usuario, pero órdenes distintos entre usuarios, evitando el sesgo que introduciría un estado aleatorio compartido entre todos los casos.