

Sistema de Recomendación de Itinerarios Turísticos Personalizados: Selección y Adaptación de Algoritmos

Ethan Yáñez Felber

ethan.yanez@usm.cl

Heilym Ramirez
Profesora Guía

Franco Jorquera
Profesor Correferente

Resumen: Viña del Mar presenta un ecosistema turístico fragmentado, lo que dificulta la planificación para los visitantes y limita la visibilidad de la oferta local. Este trabajo implementa el sistema de recomendación de la plataforma web Amukan, diseñado para construir itinerarios personalizados según intereses, tiempo y presupuesto. El sistema integra modelos de *embeddings* (*Sentence Transformers*) para el perfilado semántico y adapta el *Team Orienteering Problem with Time Windows* (TOPTW) para la optimización logística de rutas. La validación, que combinó pruebas *offline* y cualitativas con usuarios, demostró consistencia semántica en la alineación de intereses y tiempos de respuesta promedio de ~866 ms (manteniéndose bajo el umbral crítico de 1.0 s en todos los escenarios simulados), pese a las limitaciones de datos de origen. El proyecto entrega un Producto Mínimo Viable (MVP) de bajo costo computacional, validando la aplicación de IA en entornos con restricción de recursos.

Palabras Clave: Recomendación, Itinerarios, Turismo, Optimización.

1 Introducción

La planificación de viajes turísticos en destinos como Viña del Mar suele exigir la consulta de múltiples fuentes de información dispersas tales como alojamiento, transporte, actividades y eventos, lo que implica una mayor inversión de tiempo y esfuerzo para los visitantes [1], [2]. Paralelamente, los propios residentes de la comuna presentan una baja integración con la oferta turística local, caracterizada por un desconocimiento de actividades, panoramas y oportunidades de disfrute, lo que limita su vínculo con el entorno y genera una percepción subvalorada del potencial turístico circundante [1], [3].

A su vez, los emprendimientos locales enfrentan fuertes barreras para competir en entornos digitales dominados por plataformas internacionales. Investigaciones del BID [4] y diagnósticos nacionales [1], [2], [3] revelan que la concentración de mercado, la falta de conocimientos digitales y la limitada conectividad regional restringen la



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Sistema de Recomendación de Itinerarios Turísticos Personalizados: Selección y Adaptación de Algoritmos

Nombre del candidato(a): Ethan Javier Yáñez Felber

Carrera / Grado: Ingeniería en Informática

Campus: José Miguel Carrera **Departamento:** ELINF

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Heilym Camila Ramirez Rico, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 10/07/2026

Firma:

Estudiante o Candidato(a):

Fecha: 10/07/2026

Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

visibilidad de pequeñas y medianas empresas, impidiendo que capturen la demanda turística a través de canales propios.

Finalmente, las instituciones públicas se encuentran en niveles bajos de transformación digital [5]. La fragmentación de datos y la centralización de información a nivel nacional dificultan la toma de decisiones locales, retrasando políticas que podrían responder a patrones dinámicos de turistas y residentes [1]. La Estrategia Nacional de Turismo 2030 [6] subraya la urgencia de implementar sistemas de Big Data y plataformas de información actualizada para sustentar la planificación del destino.

En este marco se plantea Amukan, una plataforma web que tiene como objetivo simplificar la organización de viajes y potenciar la oferta turística de Viña del Mar. La propuesta general consiste en recopilar datos de alojamiento, transporte, actividades y negocios locales a través de procesos de recolección automatizada y colaboración con socios estratégicos, poniendo a disposición información actualizada y confiable en un único espacio digital.

Sin embargo, la centralización de datos por sí sola no resuelve el principal problema a abordar: la falta de personalización en la planificación turística. Actualmente, los itinerarios disponibles en portales y guías digitales suelen ser genéricos, poco adaptados a los intereses específicos de cada visitante y ciegos ante variables contextuales como la disponibilidad local. Esta carencia limita la experiencia del usuario y desaprovecha el potencial de los datos para conectar la demanda con la oferta real del destino.

Bajo este contexto, el presente trabajo aborda el diseño del motor de Amukan como una instancia del *Tourist Trip Design Problem* (TTDP), modelado específicamente a través del *Team Orienteering Problem with Time Windows* (TOPTW) para la gestión multi-día y el respeto de restricciones horarias. Formalmente, se plantea la optimización de una función de utilidad S_i basada en la afinidad semántica de los Puntos de Interés (POIs), sujeta a restricciones de conectividad geoespacial, ventanas temporales $[a_i, b_i]$ operativas y un presupuesto máximo. El desarrollo se enfoca en integrar el filtrado de recomendaciones y la optimización de rutas en una herramienta única que mitigue la fragmentación de la oferta turística de Viña del Mar, operando bajo tres restricciones críticas: la escasez de datos estructurados, la exigencia de un Producto Mínimo Viable (MVP) en un plazo acotado y un presupuesto nulo. Esta arquitectura emplea exclusivamente APIs gratuitas, modelos de bajo consumo y herramientas de código abierto para asegurar la sostenibilidad técnica del sistema.

1.1 Objetivos

Objetivo General

Implementar el motor de recomendación de un sistema para la generación de itinerarios turísticos personalizados, mediante la selección, adaptación e integración de algoritmos existentes de recomendación y optimización, considerando restricciones de tiempo, presupuesto y preferencias de los usuarios.

Objetivos Específicos

- Adaptar algoritmos de recomendación y optimización adecuados para itinerarios turísticos.
- Modelar perfiles dinámicos de usuario a partir de datos de interacción.
- Integrar la arquitectura que conecte la base de datos, el motor de recomendación y el módulo de optimización.

- Calibrar los parámetros operativos del sistema para optimizar sus niveles de eficiencia y consistencia semántica.
- Evaluar el desempeño mediante pruebas técnicas, simulaciones controladas y métricas automáticas de calidad, diversidad y eficiencia.

1.2 Metodología

La metodología abordada se basa en el enfoque ágil *Scrum*, estructurado en tres iteraciones principales que permiten avanzar de forma incremental hacia un producto mínimo viable (MVP). Este enfoque facilita la adaptación continua, la validación temprana y la mejora progresiva del sistema de recomendación turística propuesto para la plataforma.

1.2.1 Levantamiento y Análisis

En esta etapa se realiza una revisión del estado del arte, abarcando sistemas de recomendación aplicados al turismo, algoritmos de optimización de rutas como el *Orienteering Problem* (OP) y el *Traveling Salesman Problem* (TSP), y arquitecturas escalables en entornos geoespaciales que integran tecnologías como PostGIS y Django. A partir de este análisis, se definen los requerimientos funcionales y no funcionales del sistema. Los requerimientos funcionales incluyen:

- **Gestión de Itinerarios:** El sistema debe permitir la generación automática de itinerarios turísticos de uno o varios días, ajustados al presupuesto total y las preferencias temáticas del usuario.
- **Visualización Geoespacial:** El sistema debe desplegar las rutas generadas en un mapa interactivo, mostrando la secuencia lógica de visitas y permitiendo acceder al detalle de cada Punto de Interés (POI).
- **Perfilado de Usuario:** El sistema debe permitir el registro de usuarios y la actualización dinámica de su perfil de preferencias basado en sus interacciones (guardado de rutas y marcadores de favoritos).
- **Filtrado Contextual:** El sistema debe ser capaz de reordenar los listados de lugares y actividades en función de la similitud semántica con el perfil actual del usuario.

Por su parte, los requerimientos no funcionales priorizan la viabilidad operativa y la mantenibilidad del software, estableciendo los siguientes criterios:

- **Eficiencia de Desempeño:** El sistema debe garantizar un tiempo de respuesta inferior a 1.0 segundo para la generación de listas de recomendación y planificación de itinerarios simples, asegurando una experiencia de usuario fluida que evite el abandono de la plataforma.
- **Eficiencia de Recursos:** La arquitectura debe ser capaz de operar en servidores de recursos limitados (VPS de gama baja o capa gratuita), optimizando el uso de memoria RAM y el uso de almacenamiento eficiente en base de datos.
- **Modularidad y Bajo Acoplamiento:** El sistema debe estructurarse en componentes independientes que interactúen mediante interfaces definidas, permitiendo la sustitución o actualización tecnológica de cualquier módulo específico sin afectar la estabilidad del sistema global.

1.2.2 Diseño

Durante el diseño se modelan los datos relevantes, incluyendo puntos de interés (POIs), eventos, perfiles de usuario y restricciones como tiempo disponible y presupuesto. Se normalizan fuentes externas, como APIs y *datasets* públicos, para integrarlas en una base de datos geoespacial unificada. En paralelo, se seleccionan y adaptan algoritmos de recomendación, considerando enfoques como recomendación basada en contenido y *embeddings*. Para la optimización de itinerarios se evalúan heurísticas específicas del *Team Orienteering Problem with Time Windows* (TOPTW). Finalmente, se diseña una arquitectura escalable basada en Django y PostGIS, que conecta el *backend* con el motor de recomendación y el módulo de optimización, dejando preparada la infraestructura para futuras extensiones con aprendizaje automático.

1.2.3 Implementación

La implementación comienza con el desarrollo del motor de recomendación, que genera candidatos Top-k en función del perfil dinámico del usuario y calcula similitudes entre POIs y preferencias mediante *embeddings* y etiquetas normalizadas. Posteriormente, se implementan heurísticas para resolver la variante TOPTW, considerando restricciones de tiempo, presupuesto y diversidad de actividades. El sistema se integra con la plataforma web y la base de datos, permitiendo la comunicación fluida entre los distintos módulos. Se incorpora un sistema de perfilado dinámico que captura interacciones del usuario, ajustando las recomendaciones de forma continua a partir de retroalimentación explícita e implícita.

1.2.4 Evaluación

La estrategia de evaluación se estructuró bajo un enfoque mixto que combina una simulación automatizada *offline* para medir el rendimiento técnico, y una auditoría cualitativa piloto para inspeccionar la lógica operativa del sistema.

Evaluación Offline (Simulación): Dado que no se dispone de un *ground truth* o datos históricos de usuarios (necesarios para métricas como *Precision@k*), la validación principal se basa en simulaciones. Para ello, se utilizarán perfiles de usuario sintéticos ("Personas") creados a partir de los datos estadísticos del informe PLADETUR de Viña del Mar 2025 [1].

Para el motor de recomendación, se medirán métricas operativas sobre las listas generadas para estos perfiles, siguiendo el enfoque de evaluación "*Beyond Accuracy*" [7] :

- **Cobertura:** Porcentaje de POIs distintos que aparecen en las recomendaciones, respecto al catálogo real de 1,999 ítems de OSM.
- **Diversidad:** Grado de variación temática entre los POIs recomendados.
- **Tiempo de respuesta:** Latencia promedio en la generación de recomendaciones.

Para el generador de itinerarios, se medirán:

- **Simulaciones de itinerarios:** Generación de rutas para cada "Persona".
- **Evaluación de Factibilidad:** Verificación automática de que los itinerarios cumplen las restricciones de presupuesto, tiempo y coherencia lógica (ej. no incluir alojamiento para visitas por el día).

Calidad de Software y Mocks: Para asegurar la estabilidad sin dependencia de red durante el desarrollo, se implementaron *Tests Unitarios* utilizando `unittest.mock` para simular las respuestas de la API de Hugging Face y el caché de vectores.

Auditoría Técnica Piloto (Verificación Cualitativa): Como complemento analítico a la simulación anterior, se estructuró una auditoría técnica de tipo piloto con un grupo focal reducido ($n=3$), cuyo objetivo no es obtener validez estadística ni generalización de resultados sobre la experiencia de usuario de manera masiva, sino ejecutar una verificación exploratoria de caja negra. Esta instancia actúa como un mecanismo de control temprano para identificar fallas en la lógica de negocio, inconsistencias geográficas y quiebres en la coherencia de las rutas antes de un eventual escalado del software.

El diseño metodológico y el levantamiento de información de esta auditoría se fundamentan en el marco ResQue [8] utilizado aquí exclusivamente como una guía de rúbrica cualitativa para estructurar las preguntas técnicas de control. Este estándar permite recolectar retroalimentación cualitativa estandarizada respecto a la relevancia percibida de los lugares sugeridos, la confianza del usuario en los criterios de ordenamiento espacial y la intención de uso real de los itinerarios generados.

2 Marco Conceptual

El presente trabajo se sitúa en la intersección entre la inteligencia artificial aplicada al turismo y el diseño de sistemas adaptativos de recomendación.

2.1 Sistemas de recomendación

Los sistemas de recomendación son herramientas computacionales diseñadas para sugerir contenidos, productos o servicios relevantes a los usuarios, en función de sus preferencias, comportamientos pasados o similitudes con otros perfiles. En el contexto turístico, permiten personalizar la experiencia del viajero, facilitando la selección de actividades, lugares y servicios que se ajusten a sus intereses, restricciones y contexto. Estudios recientes destacan su importancia para aumentar la satisfacción del usuario y la eficiencia operativa en plataformas turísticas [4], [9], [10].

Existen tres enfoques principales:

- **Filtrado colaborativo:** Basado en el comportamiento de usuarios similares. Requiere grandes volúmenes de datos históricos y presenta el problema de arranque en frío "*cold start*" cuando no hay suficiente información sobre nuevos usuarios o ítems. Investigaciones recientes confirman que este enfoque enfrenta dificultades en escenarios con datos escasos, un desafío común en aplicaciones turísticas donde surgen con frecuencia nuevos destinos o usuarios [11].
- **Recomendación basada en contenido:** Compara atributos de los ítems (por ejemplo, tipo de actividad, ubicación, categoría temática) con el perfil del usuario. No depende de otros usuarios, lo que lo hace ideal en contextos con baja densidad de datos o alta personalización. En turismo, los sistemas basados en contenido aprovechan atributos de los puntos de interés (POIs), como descripciones textuales, etiquetas semánticas o categorías, para coincidir con las preferencias del usuario. Estudios demuestran que este método es particularmente efectivo

en turismo debido a su capacidad para manejar datos escasos y proporcionar recomendaciones precisas [9], [12], [13].

- **Modelos híbridos:** Combinan ambos enfoques, integrando datos colaborativos y atributos de contenido. Suelen ofrecer mayor precisión, pero requieren mayor complejidad computacional y volumen de datos.

A partir del análisis anterior, se opta por un enfoque basado en contenido, ya que logra responder a la naturaleza de los datos disponibles, caracterizados por la ausencia de un historial previo de interacciones o "*ground truth*", imposibilitando la aplicación efectiva de métodos colaborativos o híbridos que dependen de la densidad de datos históricos. De este modo, el sistema aprovecha la capacidad del modelo para operar bajo el escenario de arranque frío "*cold start*", utilizando representaciones en espacios vectoriales para identificar candidatos Top-k que maximicen la afinidad semántica con el perfil del usuario.

2.2 Embeddings y Representación Semántica

La representación vectorial es fundamental en los sistemas de recomendación modernos, ya que permite codificar tanto los puntos de interés como los perfiles de usuario en espacios numéricos continuos de baja dimensión. A diferencia de las técnicas tradicionales (como *one-hot encoding*), los *embeddings* capturan relaciones semánticas complejas, permitiendo que elementos con significados similares se ubiquen geométricamente cerca en el espacio vectorial para ser comparados mediante métricas como la similitud coseno.

La selección del modelo lingüístico adecuado está condicionada por la naturaleza de los datos de entrada, el idioma del corpus y la capacidad de procesamiento de la infraestructura. En la Tabla 1 se comparan las técnicas más relevantes del estado del arte aplicadas al dominio turístico:

Tabla 1. Comparativa de Técnicas de Representación Semántica y Modelos de Lenguaje.

Fuente: Elaboración Propia

Técnica Modelo	/	Enfoque Teórico	Ventajas Clave	Desventajas / Limitaciones	Aplicación en Literatura
Word2Vec Doc2Vec	/	Vectores estáticos basados en coocurrencia local de palabras.	Eficiencia en corpus pequeños; ideal para descripciones directas.	No captura la semántica contextual global; requiere corpus masivos para generalizar.	Empleado en el sistema CHIP [12] y evaluado como <i>baseline</i> en SBTRec [14].
BERT TourBERT	/	Modelos de lenguaje profundo basados en Transformers	Alta precisión; <i>embeddings</i> dinámicos según el contexto de la frase.	Elevada dimensionalidad (768+), alto consumo de RAM/GPU y dependencia de	Utilizado en TripCraft [15] (Persona Score) y modificado

	contextuales bidireccionales.		corpus especializados de trayectorias.	en el módulo POI-BERT de SBTRec [14].
Sentence Transformers (MiniLM)	Modelos ligeros optimizados mediante redes siamesas para similitud de oraciones completas.	Soporte multilingüe nativo, representaciones compactas (384 dimensiones) y bajo consumo de recursos en servidores sin GPU.	Menor capacidad de personalización profunda en comparación con arquitecturas BERT masivas <i>fine-tuned</i> .	Aplicado en SBTRec [14] (all-MiniLM-L6-v2) para análisis de sentimiento en reseñas.

A partir del análisis comparativo anterior, se descartó el uso de modelos puramente anglocéntricos o computacionalmente densos (como BERT o Word2Vec puro), y se seleccionó la variante `paraphrase-multilingual-MiniLM-L12-v2` de la familia de Sentence Transformers [16]. Respondiendo a tres factores críticos del destino:

- **Soporte Multilingüe Nativo:** Permite procesar e inferir relaciones semánticas directas entre una preferencia ingresada en español y establecimientos locales cuyos tags de OpenStreetMap utilizan terminología anglosajona (ej: *"Food Truck"*, *"Hostel"*, *"Skatepark"*), eliminando capas intermedias de traducción que añadirían latencia y pérdida de contexto.
- **Robustez ante la Dispersión de Datos (*Sparsity*):** A diferencia de arquitecturas que exigen reseñas densas de usuarios (como SBTRec), este modelo demuestra eficacia al generar embeddings consistentes utilizando exclusivamente *inputs* estructuralmente limitados (Nombre y Categoría de OSM).
- **Eficiencia Dimensional y Operativa:** Al producir vectores compactos de 384 dimensiones (en lugar de las 768 de BERT), se reduce a la mitad el almacenamiento en la base de datos y se acelera la complejidad computacional $O(D)$ de las operaciones algebraicas de distancia coseno. Esto garantiza tiempos de respuesta mínimos y viables para interacciones en tiempo real sobre una infraestructura estándar, sin requerir aceleración de GPUs.

2.3 Perfilado Dinámico de Usuarios

En los sistemas de recomendación turística, el perfilado dinámico se define como la construcción y actualización continua de representaciones individuales que reflejan intereses y comportamientos cambiantes. A diferencia de los modelos estáticos basados en cuestionarios iniciales, este enfoque reconoce que las preferencias del turista no son fijas, sino que evolucionan según el contexto y la interacción con la oferta disponible.

La literatura reciente identifica la dispersión de datos (*Data Sparsity*) como una de las mayores barreras para la personalización efectiva. Otaki y Baba [17] proponen un marco interactivo que aborda esta limitación mediante el uso de datos aumentados basados en la interacción; su sistema captura las modificaciones que el usuario realiza sobre los itinerarios para inferir preferencias implícitas y ajustar los modelos de *ranking*. En

Amukan, este mecanismo actúa como una estrategia para mitigar el sesgo de la información estática, permitiendo que el sistema "aprenda" de las selecciones del usuario para compensar la falta de metadatos detallados en la fuente de datos original.

Por otro lado, la investigación de Esper et al. [13] en el sistema +Tour destaca que la personalización no debe limitarse únicamente a los gustos temáticos, sino que debe integrarse con las restricciones técnicas del entorno. Amukan adopta este principio de optimización conjunta, donde el perfil dinámico no es un componente aislado, sino una entrada crítica para el motor de optimización logística. Al modelar al usuario como un vector de preferencias que evoluciona mediante aprendizaje incremental, el sistema ajusta la relevancia de los candidatos antes de que el algoritmo de rutas los procese.

Mientras que los enfoques de perfilado estáticos basados en cuestionarios iniciales presentan limitaciones al no capturar la evolución de los intereses del usuario durante el transcurso de un viaje, la integración de una política de actualización constante permite que el sistema mantenga un equilibrio adaptativo entre la memoria histórica y la novedad. Esta capacidad adaptativa es lo que permite que el motor de recomendación se mantenga ágil y sensible, transformando una navegación simple por la plataforma en un proceso de refinamiento continuo del itinerario final.

2.4 Modelos de Optimización en Turismo

La planificación de rutas personalizadas se ha modelado tradicionalmente bajo el marco del *Tourist Trip Design Problem* (TTDP). A diferencia del *Traveling Salesman Problem* (TSP), que busca la ruta más corta visitando todos los nodos, el TTDP es un problema de selección, en el cual, el sistema debe elegir qué lugares visitar para maximizar la utilidad del turista dentro de límites de tiempo y presupuesto [18]. El modelo base para resolver esto es el *Orienteering Problem* (OP), el cual ha evolucionado hacia el *Team Orienteering Problem with Time Windows* (TOPTW) para permitir la gestión de itinerarios multi-día y el respeto a los horarios de apertura de los lugares [18], [19].

Sin embargo, al contrastar la literatura con la realidad operativa de un destino como Viña del Mar, surge una brecha crítica. Investigaciones recientes proponen metaheurísticas híbridas de alta complejidad, como la integración de algoritmos genéticos y búsqueda de vecindad variable (GA-VND) [11] o búsquedas en árboles de Monte Carlo (MCTS) [9]. Si bien estos modelos reportan una precisión matemática superior en entornos controlados, operan bajo el supuesto de "información perfecta" (datos ricos, reseñas masivas y tiempos exactos). En la práctica, el uso de fuentes abiertas como OpenStreetMap (OSM) introduce el problema de la dispersión semántica *Data Sparsity*. Como señalan Saifudin y Widiyaningtyas [20], la falta de descripciones densas vuelve ineficiente el uso de algoritmos iterativos pesados, ya que estos terminan consumiendo recursos computacionales optimizando sobre atributos inexistentes o poco informativos.

Frente a esta falla de los modelos tradicionales, surge la necesidad de un enfoque que priorice la eficiencia. Autores como Granda y Vitoriano (2025) [21] han validado recientemente que las heurísticas constructivas de tipo *Greedy* pueden ser superiores a los métodos exactos en términos de rendimiento y factibilidad para variantes complejas del TOPTW. Mientras que un algoritmo exacto o una metaheurística pesada pueden requerir tiempos de cómputo inviables para una aplicación web, un esquema *Greedy* permite construir rutas de forma instantánea. Esta idea de "poda inteligente" es respaldada por Halder et al. (2022) [22], quienes proponen que, dada la magnitud del catálogo de puntos de interés, el éxito de la recomendación no reside en la búsqueda

exhaustiva del itinerario perfecto, sino en una pre-selección personalizada y una posterior construcción eficiente.

Por estas razones, Amukan se aleja de la tendencia actual de utilizar únicamente IA Generativa para la creación de rutas, técnica que, según Gayathri et al. [10], suele presentar fallas de coherencia logística y altos costos de latencia. En su lugar, adopta una estrategia de selección semántica seguida de una construcción *Greedy*. Donde primero, se delega a los *embeddings* la tarea de resolver la dispersión de datos de OSM, permitiendo "entender" qué lugares son afines al usuario incluso con descripciones mínimas [20]. Para luego utilizar una heurística constructiva secuencial que selecciona, paso a paso, el lugar que aporta mayor utilidad marginal. Como demuestran Hapsari et al. [23], este enfoque permite integrar restricciones horarias y categorías de turistas sin la rigidez de los modelos matemáticos puros, asegurando que el itinerario sea siempre físicamente realizable y computacionalmente sostenible.

2.5 Arquitectura de Software

La elección de la arquitectura es un factor determinante para la mantenibilidad y la agilidad del desarrollo, especialmente en sistemas que integran procesos de distinta naturaleza como el procesamiento vectorial y la optimización geoespacial. Para Amukan, se realizó una distinción entre la arquitectura de despliegue macro y la estructuración interna del núcleo del sistema. Mientras que a nivel externo se separa físicamente la interfaz, el servidor central y la persistencia, el desafío principal reside en organizar la lógica interna del backend de forma eficiente.

Al contrastar los paradigmas analizados por Su y Li [24], se identifican tres enfoques principales con distintos impactos operativos:

- **Arquitectura de Microservicios:** Se basa en el despliegue de múltiples procesos independientes que requieren configuraciones de red y formatos de serialización individuales. Esto suele ralentizar el ciclo de desarrollo y dificulta realizar cambios transversales entre servicios debido a la complejidad de la comunicación distribuida.
- **Monolito Tradicional:** Ofrece una alta velocidad de desarrollo inicial y simplicidad operativa al funcionar como una unidad única. Sin embargo, la falta de fronteras claras entre los componentes del sistema suele derivar en una estructura difícil de comprender y mantener conforme el código crece.
- **Monolito Modular:** Se presenta como un punto medio que combina la simplicidad operativa de una unidad única con el desacoplamiento propio de los sistemas distribuidos. Este modelo organiza el sistema en módulos con fronteras bien definidas y dependencias explícitas.

Por lo expuesto anteriormente, se optó por implementar un Monolito Modular para evitar una sobre-ingeniería que comprometería la viabilidad del MVP. Esta decisión permite simplificar el despliegue al no requerir orquestadores complejos y asegura una operación eficiente en infraestructuras con recursos limitados. Una ventaja técnica clave de este diseño es la eficiencia en la comunicación interna, ya que el intercambio de información entre los módulos de recomendación y optimización se realiza mediante interfaces nativas del lenguaje de programación.

Esto resulta significativamente más rápido que el uso de llamadas remotas típicas de arquitecturas separadas, facilitando además una organización del código superior a la de una aplicación convencional. De este modo, se establece una estructura modular que

mitiga los riesgos de definir incorrectamente los límites de los dominios en etapas tempranas. Esta base permite una futura migración hacia microservicios solo cuando la demanda de carga real lo justifique, garantizando una arquitectura sostenible y clara.

3 Diseño e Implementación del Sistema

El objetivo central de esta etapa fue traducir los modelos teóricos en una arquitectura de software viable y capaz de operar bajo las restricciones definidas para el proyecto.

3.1 Arquitectura de la Solución

Conforme a la justificación teórica expuesta en el marco conceptual (ver Sección 2.5), se implementó una arquitectura de monolito modular para estructurar el *backend* del sistema.

3.1.1 Arquitectura de Contenedores

La solución se organiza en contenedores que agrupan diferentes servicios y componentes esenciales del sistema. Estos contenedores, inspirados en los principios del modelo C4 de arquitectura de software, fueron definidos para simplificar la visualización de la estructura a dos niveles de detalle:

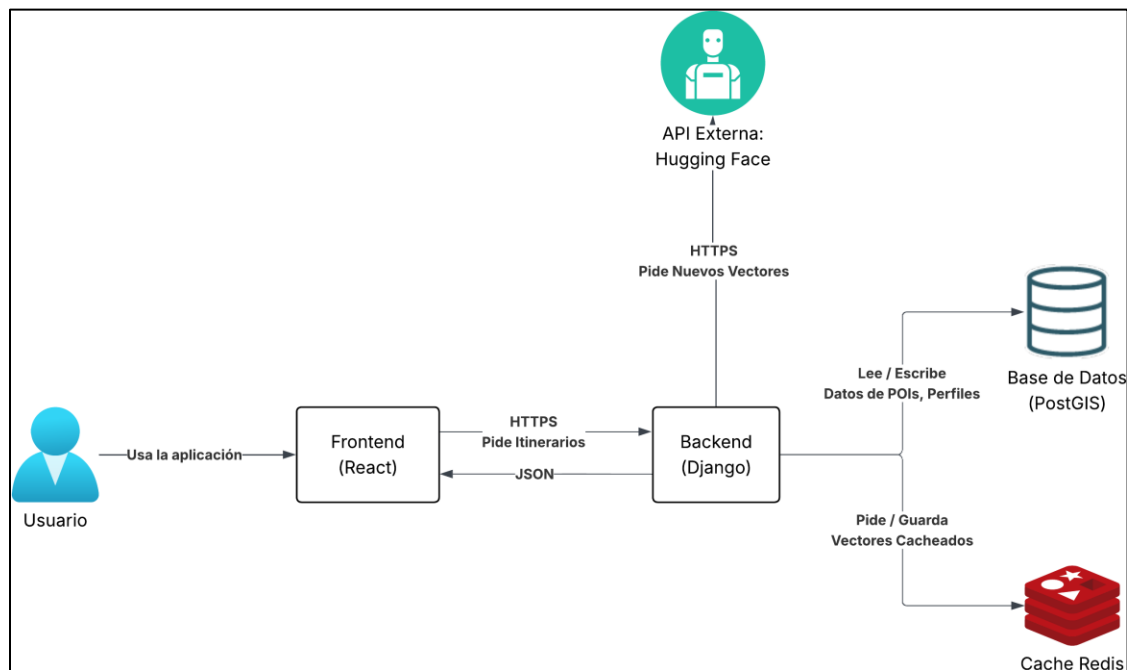


Figura 1. Diagrama de Arquitectura de Contenedores

Fuente: Elaboración Propia

- **Contenedor 1 - Frontend (React):** El *frontend* de la aplicación, desarrollado en React, es responsable de la interfaz y la interacción con el usuario final. Este contenedor se comunica con el *backend* Django a través de una API RESTful, permitiendo al usuario realizar acciones como la creación de itinerarios y la visualización de recomendaciones personalizadas.

- **Contenedor 2 - Backend (Django):** La aplicación Django sirve como el *backend* y proporciona una API RESTful al *frontend*. Dentro de este contenedor se encuentra toda la lógica de negocio del sistema, que abarca la orquestación de itinerarios, el perfilado dinámico de usuarios, y el motor de recomendaciones. Además, gestiona las interacciones entre el usuario y el sistema, y se encarga de la persistencia de los datos a través de la base de datos.
- **Contenedor 3 - Base de Datos (PostGIS):** Se utiliza una base de datos PostgreSQL con la extensión PostGIS para almacenar los datos persistentes. Esta base de datos almacena información clave, como los perfiles de usuario, los Puntos de Interés (POIs), y las interacciones de los usuarios, además de los datos geospaciales asociados, como las coordenadas de los POIs.
- **Contenedor 4 - Caché (Redis):** El contenedor de Redis se utiliza para almacenar en memoria los resultados de cálculos costosos, como los vectores de perfil de usuario o las recomendaciones previamente generadas. El uso de Redis mejora significativamente el rendimiento al evitar que los cálculos se repitan innecesariamente. Al ser una base de datos en memoria, permite obtener resultados de manera rápida y eficiente.
- **Contenedor 5 - Servicio Externo de Hugging Face:** Dado que el procesamiento de *embeddings* para las recomendaciones es un proceso computacionalmente costoso, se externalizó esta parte del sistema a Hugging Face, utilizando su API para acceder al modelo preentrenado `paraphrase-multilingual-MiniLM-L12-v2`. Externalizar este componente permite escalar fácilmente el modelo sin sobrecargar los recursos internos del servidor.

3.1.2 Estructura Interna del Backend: Componentes Lógicos

Dentro del *backend*, la lógica está dividida en módulos lógicos (denominados "*apps*" en el marco de trabajo con Django), cada uno con una responsabilidad bien definida. Esta modularidad interna permite un desarrollo más organizado y facilita el mantenimiento (Ver Figura 2).

Generación de Itinerarios (`apps.travel`): Este bloque agrupa la lógica central del negocio para la creación de rutas turísticas y se sustenta principalmente en los archivos `itinerary_generator.py` y `data_provider.py`.

- **Controlador de Itinerarios (`views.py`):** Es el punto de entrada del sistema. Su función principal es recibir las solicitudes HTTP desde la interfaz de usuario y validar la integridad de los datos de entrada, asegurando que las fechas sean coherentes (manejo de zonas horarias), que el presupuesto sea numérico y que el destino exista. Actúa como una barrera de validación antes de delegar el trabajo pesado al núcleo del sistema.
- **Servicio de Planificación (`services.py`):** Actúa como el director de orquesta del proceso. No realiza los cálculos matemáticos por sí mismo, sino que coordina el flujo de ejecución definido en las reglas de negocio. Sus responsabilidades incluyen:
 - Detectar el tipo de viaje (si es de un día o multi-día).
 - Solicitar los candidatos al Proveedor de Datos.

- Ejecutar las estrategias de reserva para servicios esenciales (Alojamiento y Comida) antes de la optimización recreativa.
 - Invocar al Motor de Optimización con los parámetros y restricciones ya procesados.
- **Proveedor de Datos (`data_provider.py`):** Es el encargado de abastecer al sistema con los Puntos de Interés. Su labor no es solo leer de la base de datos, sino enriquecer y normalizar la información para el algoritmo.
 - **Normalización:** Convierte diferentes tipos de entidades (hoteles, parques, eventos) en objetos estándar con atributos comunes (coordenadas, precio estimado, duración promedio).
 - **Puntuación (*Scoring*):** Aplica una capa de personalización inicial, aumentando el puntaje base de aquellos lugares que coinciden con los intereses explícitos del usuario (ej. "Aventura") o que han sido marcados como favoritos.
- **Motor de Optimización (`optimizer.py` y `optimization_core.py`):** Es el "cerebro" matemático del sistema. El archivo `optimizer.py` gestiona la configuración del problema (presupuesto, tiempos y viajeros), mientras que en `optimization_core.py` reside la implementación de la heurística para resolver el problema logístico. Su función es iterar sobre los candidatos puntuados y decidir cuáles agregar al itinerario evaluando:
 - La utilidad marginal (relación costo-beneficio).
 - La viabilidad logística (distancia desde el punto anterior).
 - La disponibilidad de ventanas de tiempo. Este componente resuelve el problema TOPTW devolviendo la secuencia óptima de actividades.

Recomendaciones (`apps.recommendation`): Estos módulos se encargan de filtrar el universo de lugares disponibles para entregar solo los más relevantes.

- **Sistema de Recomendación (`services.py`):** Implementa la lógica de filtrado basado en contenido. Utiliza la base de datos vectorial para realizar búsquedas de similitud entre el perfil del usuario y los POIs disponibles. Su salida es una lista reducida (Top-K) de candidatos que luego será consumida por el módulo de generación de itinerarios.
- **Cliente de Embeddings (`ml_model.py`):** Es el componente de integración encargado de transformar texto (descripciones de lugares, preferencias de usuario) en vectores numéricos (*embeddings*). Actúa como interfaz de comunicación con modelos de lenguaje externos (vía API a Hugging Face), permitiendo que el sistema "entienda" semánticamente las características de cada lugar turístico.

Perfilado de Usuario: Encargado de mantener actualizada la representación semántica de las preferencias del usuario.

- **Procesador de Perfil (Job Asíncrono):** Este componente se activa tras las interacciones del usuario (ej. al guardar un itinerario). Analiza la composición del viaje guardado (¿Qué porcentaje es aventura? ¿Cuánto es gastronomía?) y actualiza los pesos en el vector de intereses del usuario. Se ejecuta de manera asíncrona para no bloquear la experiencia de usuario, asegurando que las futuras recomendaciones sean cada vez más precisas.

Capa de Datos y Sistemas Externos

- **Base de Datos (PostGIS):** Almacena la información estructurada y geoespacial. Gracias a la extensión "pgvector", permite realizar consultas híbridas: filtros espaciales (lugares cercanos) y semánticos (lugares similares) en una misma sentencia.
- **Caché Redis:** Almacena resultados temporales de cálculos costosos (como matrices de distancias entre puntos frecuentes) para acelerar la respuesta del algoritmo de optimización.
- **API Externa (Hugging Face):** Provee la capacidad computacional para la vectorización de texto sin sobrecargar la infraestructura local.

Dentro del monolito, la comunicación entre los diferentes módulos lógicos se realiza mediante llamadas internas de funciones en Python, lo que garantiza una latencia mínima. Esto es esencial para mantener un rendimiento óptimo, especialmente cuando se manejan solicitudes concurrentes o grandes volúmenes de datos. Además, los contenedores (como la base de datos y Redis) se comunican con el *backend* Django utilizando protocolos estándar (SQL para la base de datos y API para la caché de Redis), lo que simplifica la integración y mantiene la solución flexible para que el sistema pueda adaptarse a nuevas necesidades y crecer de manera controlada.

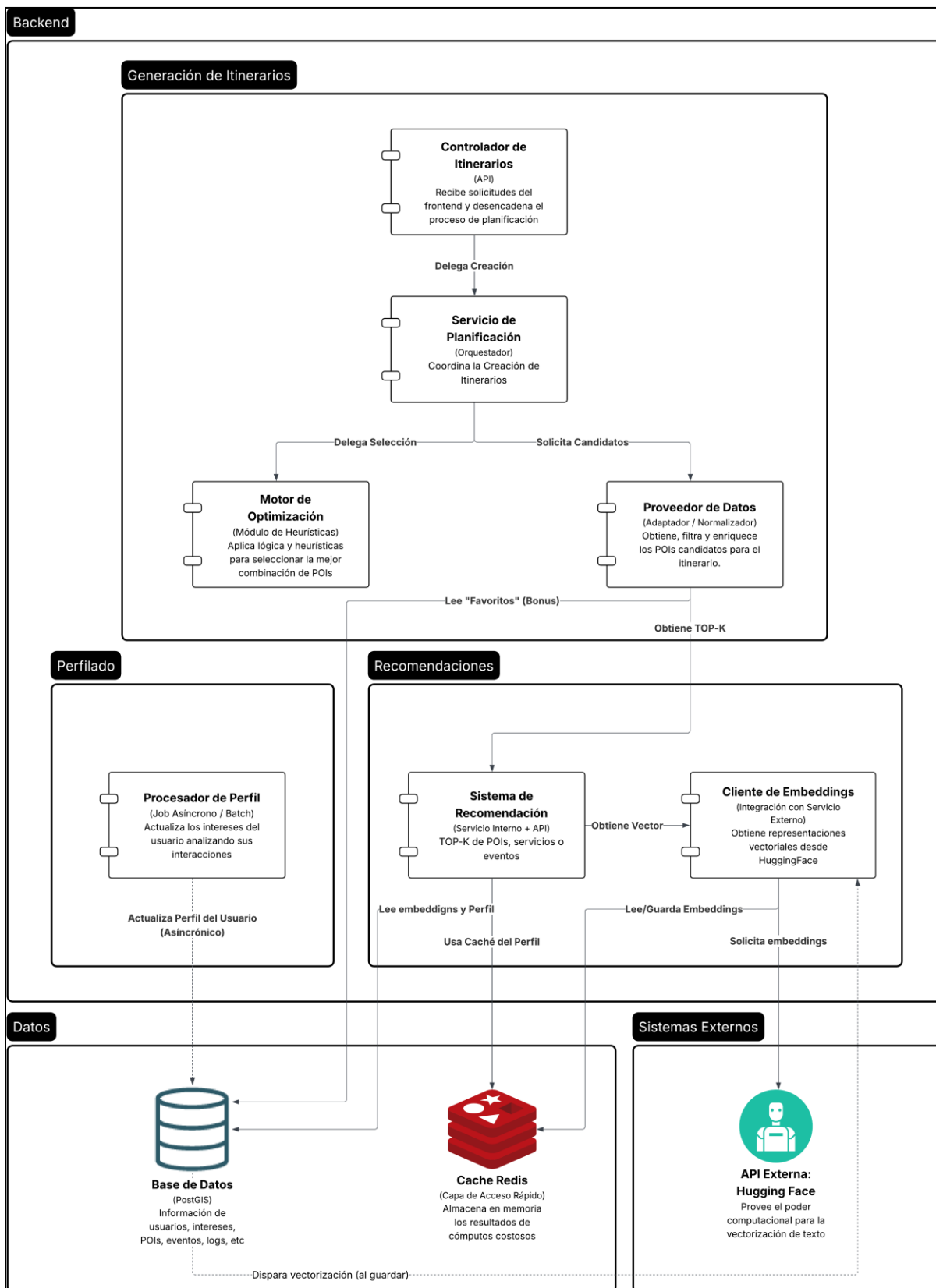


Figura 2. Diagrama de Arquitectura de Componentes

Fuente: Elaboración Propia

3.2 Perfilado Dinámico

El módulo de perfilado dinámico es el encargado de asegurar que las preferencias del usuario evolucionen a lo largo del tiempo, basándose en su interacción real con la plataforma. Este proceso se ejecuta de manera asíncrona y en un proceso de fondo (mediante un *cron job*) para evitar cualquier impacto en la latencia de las peticiones de generación de itinerarios.

El Procesador de Perfil implementa una lógica de feedback continuo, analizando el estado actual de la cuenta del usuario para inferir sus intenciones de viaje.

El siguiente diagrama representado en la Figura 3 detalla la recolección de interacciones con sus respectivas ponderaciones, el ciclo de actualización atómica que incorpora el decaimiento temporal y los límites de normalización, culminando en la invalidación del caché del vector para forzar su recálculo.

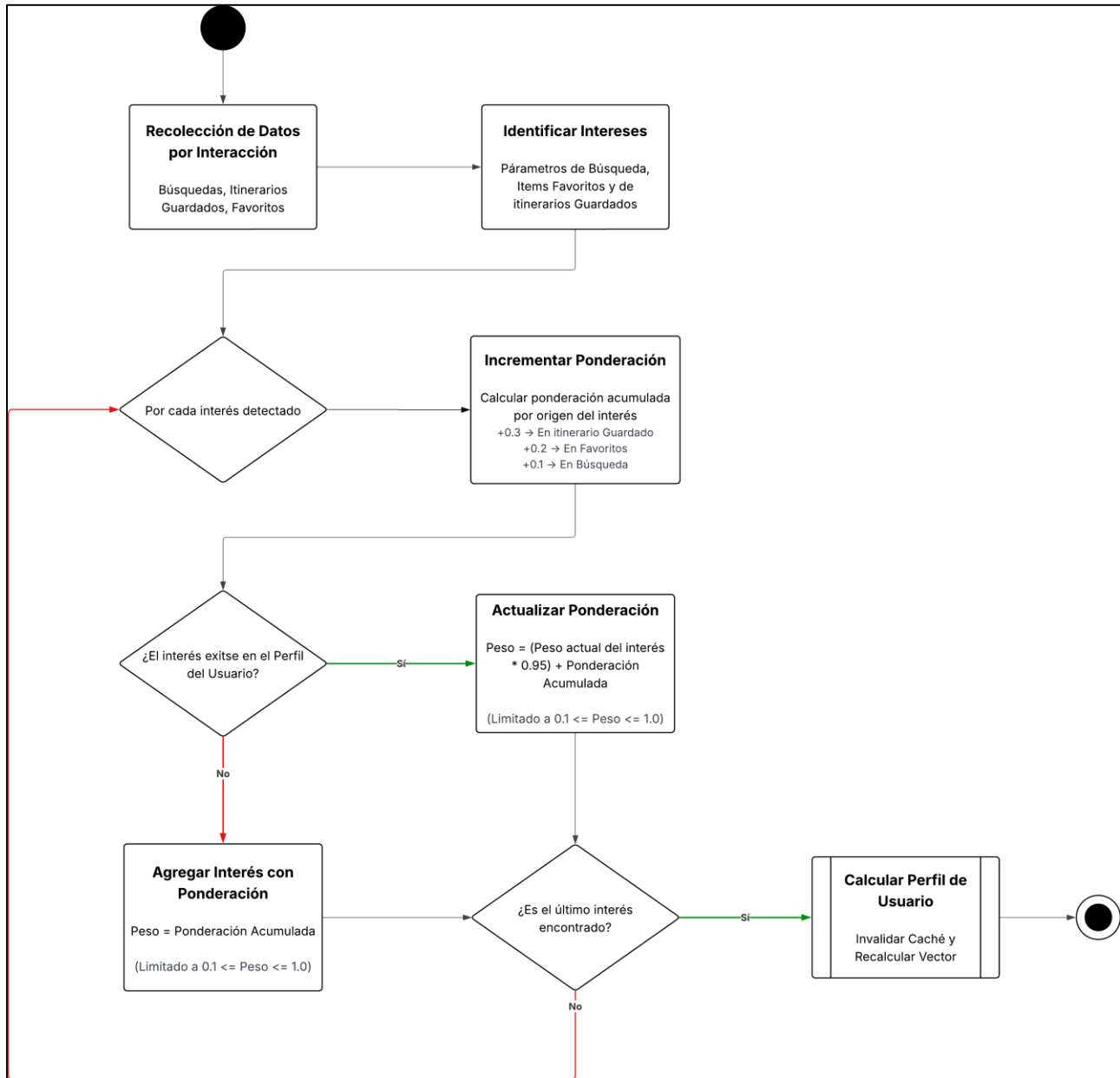


Figura 3. Flujo de Perfilado Dinámico de Usuario

Fuente: Elaboración Propia

3.2.1 Ponderación de Interacciones y Acumulación

El proceso de análisis comienza con la lectura de las interacciones almacenadas en la base de datos, incluyendo búsquedas, itinerarios guardados y marcadores de "Favoritos". El sistema prioriza las señales mediante un peso diferencial que refleja la certeza de la intención:

Tabla 2. Ponderación por Interacción

Fuente: Elaboración Propia

Fuente de Interacción	Ponderación Acumulada	Nivel de Intención
Itinerario Guardado	+0.3	Máximo (Confirmación de selección)
Item Favorito	+0.2	Alto (Interés estable a largo plazo)
Búsqueda (Search)	+0.1	Bajo (Exploración o curiosidad)

Si un mismo interés es detectado en múltiples interacciones dentro del mismo ciclo (Ej: una búsqueda y un guardado), sus pesos se acumulan en un incremento total (Ej: +0.4), que se aplica en el paso de actualización.

La elección de estos valores responde a una heurística de intención progresiva. Se asigna un valor base pequeño (+0.1) a la búsqueda, ya que representa un interés exploratorio o curiosidad que no necesariamente implica compromiso. El valor aumenta (+0.2) para los "Favoritos", que denotan una intención de visita futura estable. Finalmente, se asigna el mayor peso (+0.3) al "Guardado de Itinerario", ya que esta acción confirma que el usuario ha validado el conjunto de recomendaciones como una solución viable a su problema de viaje, representando el nivel máximo de satisfacción con el contenido.

3.2.2 Lógica de Decaimiento y Normalización

La lógica de actualización del perfil incorpora un mecanismo de decaimiento temporal. Este paso permite que el perfil se adapte dinámicamente y no quede anclado a preferencias obsoletas.

$$\text{Peso Base} = (\text{Peso Anterior} \times 0.95) + \text{Ponderación Acumulada}$$

- **Aplicación de Decaimiento:** Antes de sumar el incremento acumulado de las interacciones recientes, el sistema aplica un factor de atenuación del 5% al peso que el interés tenía previamente registrado en la base de datos.
- **Actualización Final:** El Peso Base se suma al incremento acumulado.
- **Tope y Piso (Clamping):** Para garantizar la estabilidad del perfil, el sistema se asegura de que el peso final de cualquier interés se mantenga en el rango de [0.1, 1.0], de esta forma, un interés con actividad nula decaerá gradualmente hasta el piso de 0.1 si no es reactivado.

3.2.3 Generación del Vector y Persistencia

El resultado del proceso es la actualización de los pesos en el registro de intereses del usuario. Una vez finalizado el ciclo de actualización de todos los intereses detectados, se ejecuta la "Invalidación del Caché", cuyo proceso llama una función que elimina el vector de *embedding* del usuario almacenado en la caché de Django, forzando al motor de recomendación a recalcular y almacenar un vector nuevo utilizando los pesos recién actualizados en la base de datos en la próxima solicitud de recomendación. Esto permite que el vector de usuario refleje instantáneamente el perfil más reciente.

3.3 Motor de Recomendación Basado en Contenido

El sistema de recomendación constituye el núcleo de la plataforma. Su función principal es calcular la afinidad semántica entre el perfil del usuario y el conjunto de Puntos de Interés, generando un listado Top-K de candidatos relevantes que posteriormente será procesado por el motor de optimización. Para reducir los tiempos de respuesta, este componente opera mediante estrategias de vectorización, delegación de cálculos a la base de datos y *caching*.

3.3.1 Vectorización

Siguiendo el marco conceptual del proyecto, la comparación entre POIs no se realiza a partir de etiquetas exactas, sino mediante su significado semántico. Para ello, las descripciones textuales de los POIs (por ejemplo: "Parque cultural con vistas al mar") se transforman en *embeddings* utilizando modelos de lenguaje. El perfil del usuario se representa mediante un vector calculado dinámicamente: el sistema recupera los intereses declarados, los pondera según su importancia (*weight*) y genera un promedio normalizado de sus *embeddings*.

Para la generación de estos vectores se utiliza el modelo `paraphrase-multilingual-MiniLM-L12-v2`. La justificación teórica y técnica de esta elección, basada en su capacidad multilingüe y eficiencia para entornos con escasez de datos, se detalló previamente en la sección 2.2. Sin embargo, su integración operativa presentó desafíos significativos en cuanto al despliegue de la infraestructura.

Durante la fase de desarrollo local, el modelo se instanció en memoria mediante un patrón *Singleton*, lo que permitía inferencias rápidas. No obstante, al preparar el entorno de producción, se evidenció una limitación crítica: la inclusión de las bibliotecas de `sentence-transformers` y los pesos del modelo elevaba el tamaño de la imagen Docker por encima de los 10 GB, excediendo las capacidades de almacenamiento y despliegue del servidor disponible.

Para resolver este conflicto, se optó por una estrategia de desacoplamiento del servicio de inferencia. El componente fue refactorizado para operar como un cliente HTTP, delegando el procesamiento vectorial a la API de inferencia de Hugging Face. Bajo este esquema, la aplicación evita la carga local del modelo y gestiona la vectorización, tanto de perfiles de usuario como de nuevos POIs, mediante solicitudes HTTPS.

Esta arquitectura permite trasladar parte de la carga computacional a un servicio externo a cambio de introducir una dependencia de red controlada.

3.3.2 Optimización del Cálculo de Similitud (pgvector)

Originalmente, la función central de recomendación cargaba todos los *embeddings* de los POIs en memoria y calculaba la afinidad iterando con NumPy. Este enfoque de fuerza bruta, con una complejidad algorítmica de $O(N)$, presentaba problemas de escalabilidad, generando alta latencia y un consumo de memoria lineal respecto al crecimiento del catálogo.

Para resolver este cuello de botella, se refactorizó el sistema delegando el cálculo intensivo directamente a la base de datos PostgreSQL mediante la extensión "pgvector". Esto permite realizar operaciones algebraicas nativas sobre los registros almacenados, aprovechando índices optimizados para búsqueda vectorial.

La métrica fundamental seleccionada para este sistema es la Similitud Coseno, la cual evalúa la afinidad semántica midiendo el coseno del ángulo entre el vector del usuario (u) y el vector del POI (p). A diferencia de métricas basadas en magnitud (como la distancia Euclidiana), la similitud coseno se centra en la orientación de los vectores, lo cual es el estándar para determinar la cercanía semántica en modelos de lenguaje (Sentence Transformers). Su definición matemática es:

$$\text{Similitud}(u, p) = \cos(\theta) = \frac{u \cdot p}{|u||p|} = \frac{\sum_{i=1}^n u_i p_i}{\sqrt{\sum_{i=1}^n u_i^2} \sqrt{\sum_{i=1}^n p_i^2}}$$

No obstante, para efectos de optimización en bases de datos, la extensión pgvector opera minimizando la distancia en lugar de maximizar la similitud. Por ello, la implementación técnica utiliza la Distancia Coseno, que se deriva inversamente de la similitud:

$$\text{Distancia}(u, p) = 1 - \text{Similitud}(u, p)$$

La implementación en Django utiliza consultas RawSQL para inyectar el vector del usuario directamente en la sentencia SQL. En este contexto, se emplea el operador dedicado de pgvector:

$$\text{vector_columna} \Leftrightarrow \text{vector_usuario}$$

Donde el operador $\Leftrightarrow$ calcula la distancia coseno definida anteriormente. Esto permite al motor de base de datos ordenar los resultados de menor a mayor distancia (es decir, de mayor a menor similitud) y limitar el conjunto de respuesta (Top-K) antes de devolver los datos a la aplicación, reduciendo drásticamente el tráfico de red y el uso de memoria.

3.3.3 Estrategia de Diversificación y Flujo Final

Para evitar la sobre-especialización de las sugerencias (el problema de la "burbuja de filtros"), el sistema no entrega directamente los resultados más similares. Como se ilustra en la Figura 4, se implementa un algoritmo de diversificación. Sin este paso, un usuario que demostró interés leve en "Museos" podría recibir un Top-K compuesto exclusivamente por museos, ignorando otras categorías complementarias (como parques o cafeterías) que enriquecen la experiencia turística. La inyección forzada de diversidad asegura un itinerario balanceado y aumenta la probabilidad de *serendipia* (descubrimiento de ítems inesperados pero relevantes):

- **Recuperación Ampliada:** Se solicitan a la base de datos el triple de los candidatos necesarios ($K \times 3$) ordenados por similitud.
- **Selección por Relevancia:** Se llena un porcentaje mayoritario de la lista final (ej. 70%) con los candidatos de mayor puntaje de similitud pura.
- **Inyección de Diversidad:** Para los espacios restantes, el algoritmo itera sobre los candidatos sobrantes seleccionando aquellos cuya categoría o tipo de servicio no esté ya presente en la lista actual.
- **Relleno (Fallback):** Si no es posible encontrar suficientes ítems diversos para completar el Top-K, el sistema rellena los espacios vacíos con los siguientes mejores candidatos disponibles, priorizando siempre la completitud de la respuesta.

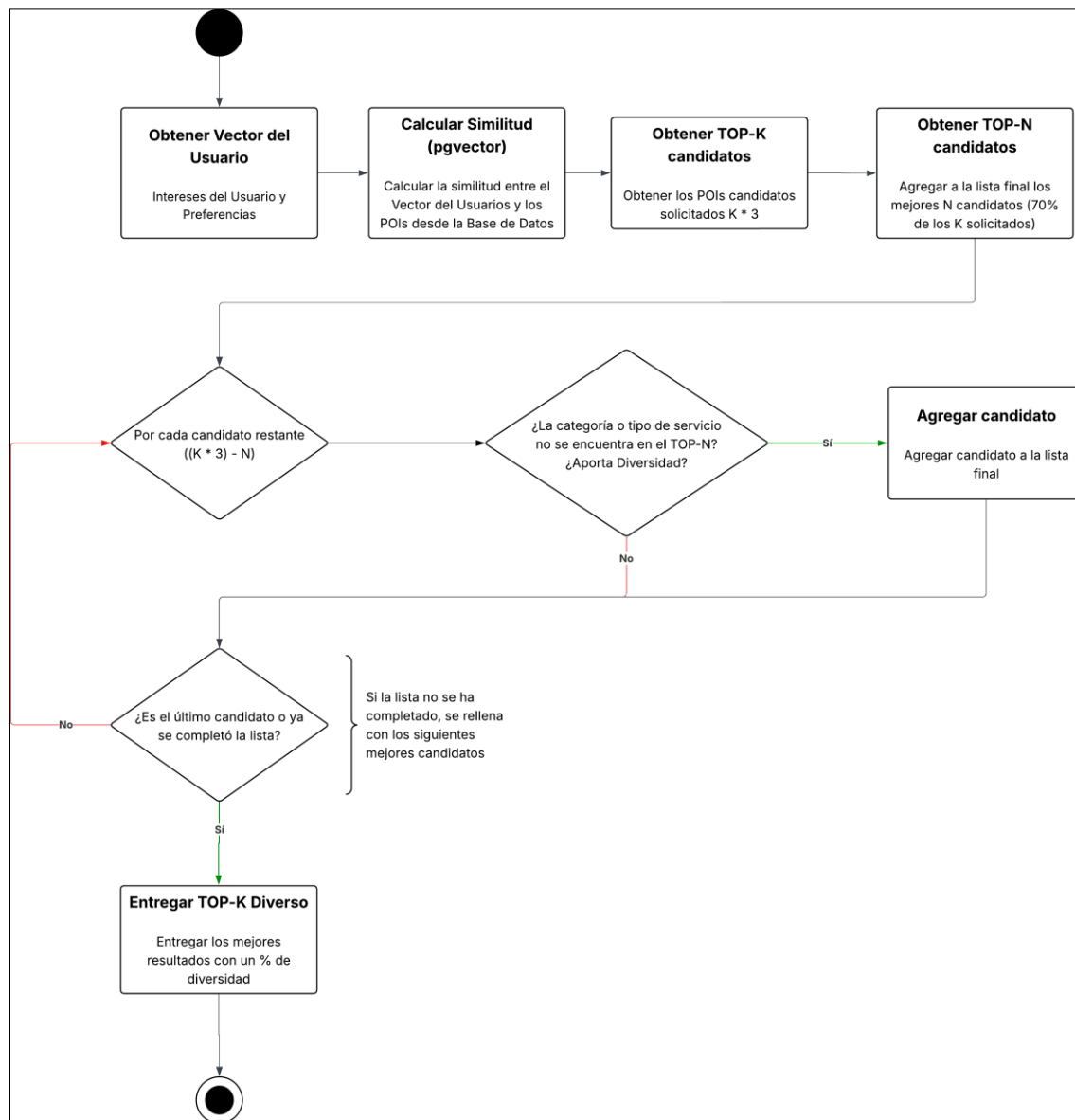


Figura 4. Flujo del sistema de recomendación

Fuente: Elaboración Propia

3.3.4 Integración Directa en la Plataforma

El motor de recomendación expone sus capacidades a través de *endpoints* REST que alimentan tres módulos críticos de la experiencia de usuario:

Home Personalizado: *Endpoint* `api/recommendations/`. Ofrece un mix variado de categorías para usuarios registrados, sirviendo como punto de entrada a la exploración (Ver Figura 5).

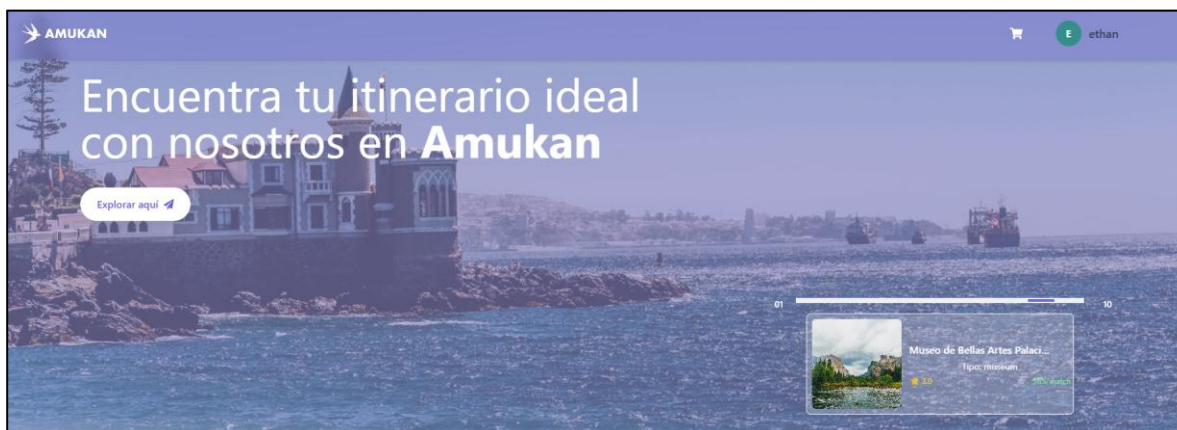


Figura 5. Integración con API de recomendaciones en Home

Fuente: Equipo Amukan

Reordenamiento Contextual: *Endpoint* `/recommendations/services/?type=...`. Utilizado al explorar categorías específicas o resultados de búsqueda; reorganiza los ítems disponibles basándose en el *score* de afinidad personal en lugar de un orden genérico (Ver Figura 6).

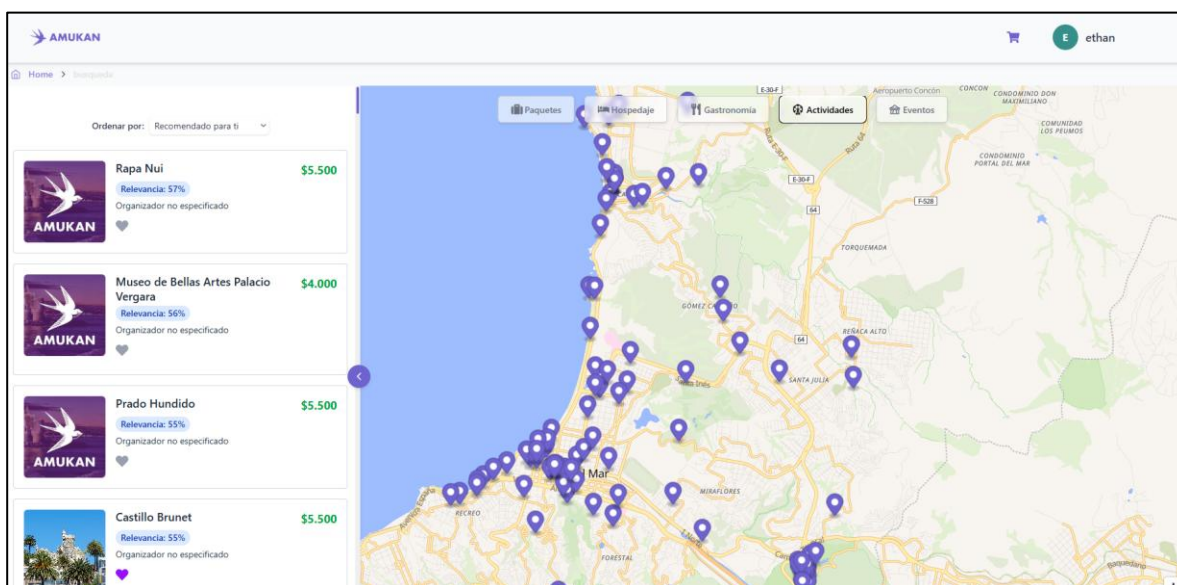


Figura 6. Integración con API de recomendaciones en orden de Resultado de Búsqueda

Fuente: Equipo Amukan

3.4 Generación de Itinerarios

Una vez que el motor de recomendación provee el pool de candidatos (un conjunto Top-K de actividades, eventos, restaurantes y alojamientos ajustados por las preferencias del usuario), el módulo de optimización debe seleccionar la secuencia óptima que conformará el itinerario final.

3.4.1 Abstracción de Datos y Restricciones del Entorno

Antes de abordar el algoritmo, fue necesario definir la unidad mínima de planificación. Inicialmente, se contempló un modelo basado en "Servicios Específicos" (ej. "Habitación doble estándar" o "Tour guiado de las 10:00 AM"). Sin embargo, durante la fase de análisis de viabilidad técnica y legal, se identificaron barreras críticas: la alta dispersión de datos en la oferta local, los costos prohibitivos de APIs comerciales y las implicancias ético-legales del web *scraping* masivo.

En consecuencia, se optó por trabajar con una abstracción basada en Puntos de Interés provenientes principalmente de OpenStreetMap (OSM) y bases de datos propias para Eventos.

- **Alojamientos y Restaurantes:** Se modelan como entidades geográficas (POIs). El sistema recomienda el lugar (ej. "Hotel O'Higgins") basándose en precios promedio referenciales, en lugar de gestionar el inventario en tiempo real de habitaciones.
- **Actividades:** Se sugieren lugares visitables (ej. "Palacio Vergara", "Muelle Vergara") en lugar de servicios turísticos paquetizados.
- **Eventos:** Se tratan como entidades temporales con fecha de inicio y fin fija, siendo los únicos elementos con restricciones horarias rígidas en el modelo actual.

Esta decisión simplifica el problema computacional al eliminar la gestión de disponibilidad en tiempo real, permitiendo enfocar los recursos en la personalización y la logística de la ruta.

3.4.2 Modelo Algorítmico

Tal como se fundamentó en el Marco Conceptual, el problema de diseñar una ruta turística bajo restricciones se basa en el *Orienteering Problem* (OP). Sin embargo, para abordar la realidad operativa de un itinerario turístico completo, fue necesario adoptar una variante específica de este modelo: el *Team Orienteering Problem with Time Windows* (TOPTW).

Esta variante es considerada en la literatura reciente como el estándar para la resolución del *Tourist Trip Design Problem* (TTDP) en sistemas de turismo inteligente [18].

El modelo se define sobre un grafo completo $G = (V, A)$, donde el objetivo es determinar un conjunto de rutas K (equipos) que maximicen la recolección de utilidad total sin violar las restricciones temporales. Matemáticamente, se busca maximizar la función objetivo:

$$\text{Max} \sum_{k=1}^K \sum_{i=1}^N S_i \cdot y_{ik}$$

Sujeto a:

1. Conectividad y Flujo:

$$\sum_{j=1}^N x_{ijk} = y_{ik} \quad \forall i \in V, \forall k \in K$$

Si se visita el nodo i en la ruta k , debe haber un flujo de entrada/salida.

2. Restricción de Tiempo:

$$\sum_{i=0}^N \sum_{j=0}^N (c_{ij} + t_{visita_i}) \cdot x_{ijk} \leq T_{max} \quad \forall k \in K$$

Donde la suma de los tiempos de viaje c_{ij} más el tiempo de visita en cada nodo no debe exceder el tiempo máximo disponible por día.

3. Ventanas de Tiempo:

$$a_i \leq Inicio_{ik} \leq b_i \quad \forall i \in V, \forall k \in K$$

En el contexto de Amukan, la correspondencia entre las variables matemáticas y la realidad turística se define de la siguiente manera:

- **Equipos (k):** Representan el número de días disponibles en el itinerario de viaje. En el marco del problema TOPTW aplicado al turismo, cada miembro del "equipo" se mapea formalmente como una ruta diaria independiente asignada al mismo usuario [18]. Esta analogía permite optimizar conjuntamente múltiples jornadas interconectadas, asegurando tanto la continuidad geográfica como la diversidad de la experiencia entre los distintos días para evitar repeticiones innecesarias de atractivos.
- **Nodos (i):** Son los Puntos de Interés (POIs) candidatos, incluyendo el alojamiento como punto de partida y llegada (nodo depósito).
- **Beneficio (S_i):** Es el *Score* de Similitud derivado del motor de recomendación vectorial. En enfoques modernos y variantes avanzadas del problema, como el modelo TOPTWVP propuesto por Marzal y Sebastia [18], se establece que este puntaje no debe ser tratado como un atributo estático o rígido. Por el contrario, representa una función de "satisfacción esperada" u optimización de utilidad que varía dinámicamente según las preferencias semánticas del visitante y el contexto de la interacción, permitiendo una personalización profunda de la ruta.
- **Costo (c_{ij}):** Representado por el tiempo de traslado entre nodos. Aunque no penaliza directamente el puntaje en la función objetivo, actúa como una restricción limitante (*Budget Constraint*). El algoritmo solo puede agregar nodos mientras la suma acumulada de los tiempos de traslado (c_{ij}) y tiempos de visita no exceda la duración total del día o las ventanas disponibles.
- **Ventanas ($[a_i, b_i]$):** Corresponden a los horarios de apertura de los lugares o a las preferencias temporales del usuario, asegurando que la visita ocurra en momentos factibles.

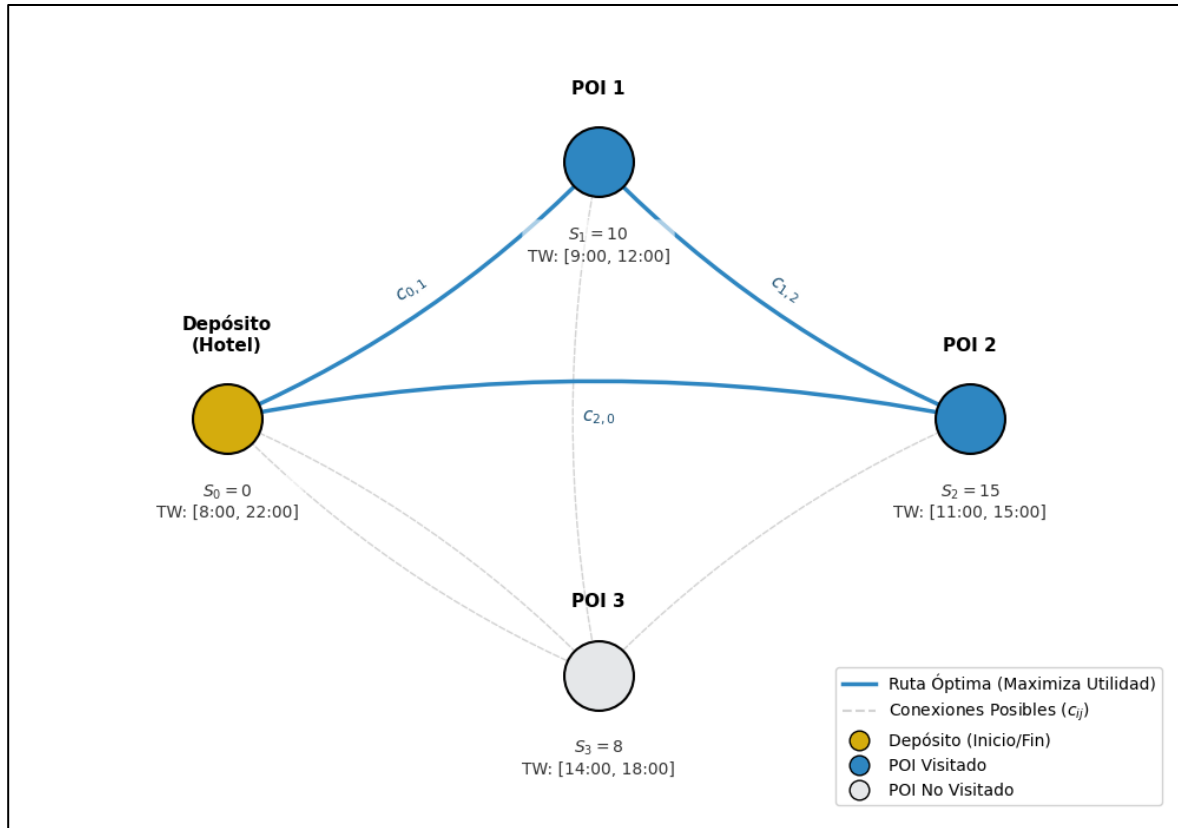


Figura 7. Modelo Conceptual de TOPTW

Fuente: Elaboración Propia

La Figura 7 ilustra la lógica de selección del algoritmo sobre un grafo simplificado. Cada nodo representa un Punto de Interés (POI) y los arcos representan los traslados posibles. La selección de la ruta óptima (líneas azules) depende de la interacción entre tres variables críticas definidas en el modelo matemático.

La viabilidad de este enfoque multi-día depende estrictamente de la incorporación de Restricciones Temporales, dado que la satisfacción real del visitante está condicionada a arribar a los puntos de interés dentro de sus respectivos horarios operativos y de accesibilidad. Por consiguiente, el modelo integra ventanas de tiempo rígidas por cada nodo, consolidándose como la restricción más crítica para garantizar que la secuencia logística diseñada sea realista y ejecutable en el destino [23].

3.4.3 Flujo de Ejecución Híbrido

La implementación final no es un *Greedy* puro, sino un sistema híbrido que combina Reglas de Negocio para necesidades básicas y Optimización Algorítmica para experiencias recreativas donde el proceso se dividió en cuatro fases secuenciales:

Fase 1 - Definición del Espacio de Búsqueda: El sistema procesa las restricciones "duras" ingresadas por el usuario (rango de fechas, presupuesto total B , tamaño del grupo). Se generan las "Ventanas de Tiempo" disponibles, segmentando los días en

bloques lógicos (Mañana, Tarde, Noche) y detectando si el viaje es *Same-Day* (mismo día) o multi-día.

Fase 2 - Reserva de Servicios Esenciales: La priorización de alojamiento y alimentación no responde a una preferencia algorítmica, sino a una jerarquía de necesidades fisiológicas y logísticas. Un itinerario multi-día es inviable si no se asegura primero la pernoctación y los tiempos de alimentación. Por tanto, el sistema trata estos servicios como "bloques estructurales" que deben resolverse antes de optimizar la utilidad recreativa (actividades), garantizando así que la ruta final sea humanamente realizable. Para ello, el sistema los selecciona primero utilizando estrategias predefinidas (*Budget, Balanced, Premium*) sobre los POIs de OSM.

El costo estimado de estos servicios se descuenta del presupuesto total (B), resultando en un presupuesto remanente (B') que será maximizado exclusivamente en actividades y eventos.

Fase 3 - Cálculo de Utilidad Marginal: Para las actividades y eventos restantes en el *pool*, el algoritmo debe decidir cuál agregar. Para ello, calcula un ratio de eficiencia (R_i) para cada candidato i en cada iteración. Este ratio no solo busca el mejor puntaje, sino el "costo de oportunidad" más bajo:

$$R_i = \frac{score_i \times Bonus\ Diversidad(c)}{Costo\ Monetario_i + Costo\ Temporal_i}$$

Donde:

- **$score_i$:** Puntuación base proveniente de la similitud coseno y favoritos.
- **$Bonus\ Diversidad(c)$:** Factor de penalización que disminuye si ya se han seleccionado muchos ítems del mismo clúster geográfico c , fomentando la exploración de nuevas zonas.
- **$Costo\ Temporal_i$:** Tiempo de actividad + tiempo de viaje desde el último punto seleccionado.

Fase 4 - Construcción y Validación: El algoritmo itera seleccionando el candidato con mayor R_i . Antes de insertarlo en el itinerario, realiza una verificación estricta de viabilidad:

- **Presupuesto:** ¿ $Costo_i \leq B'_{sobrante}$?
- **Ventana Temporal:** ¿Existe un bloque horario disponible donde quepa la actividad + el traslado?
- **Logística:** ¿El tiempo de viaje desde el punto anterior es razonable (ej. < 45 min)?

Este ciclo se repite hasta agotar el presupuesto o las ventanas de tiempo. Finalmente, el itinerario resultante pasa por un validador de calidad que revisa métricas globales (diversidad de categorías, densidad de actividades) antes de ser entregado al *frontend*.

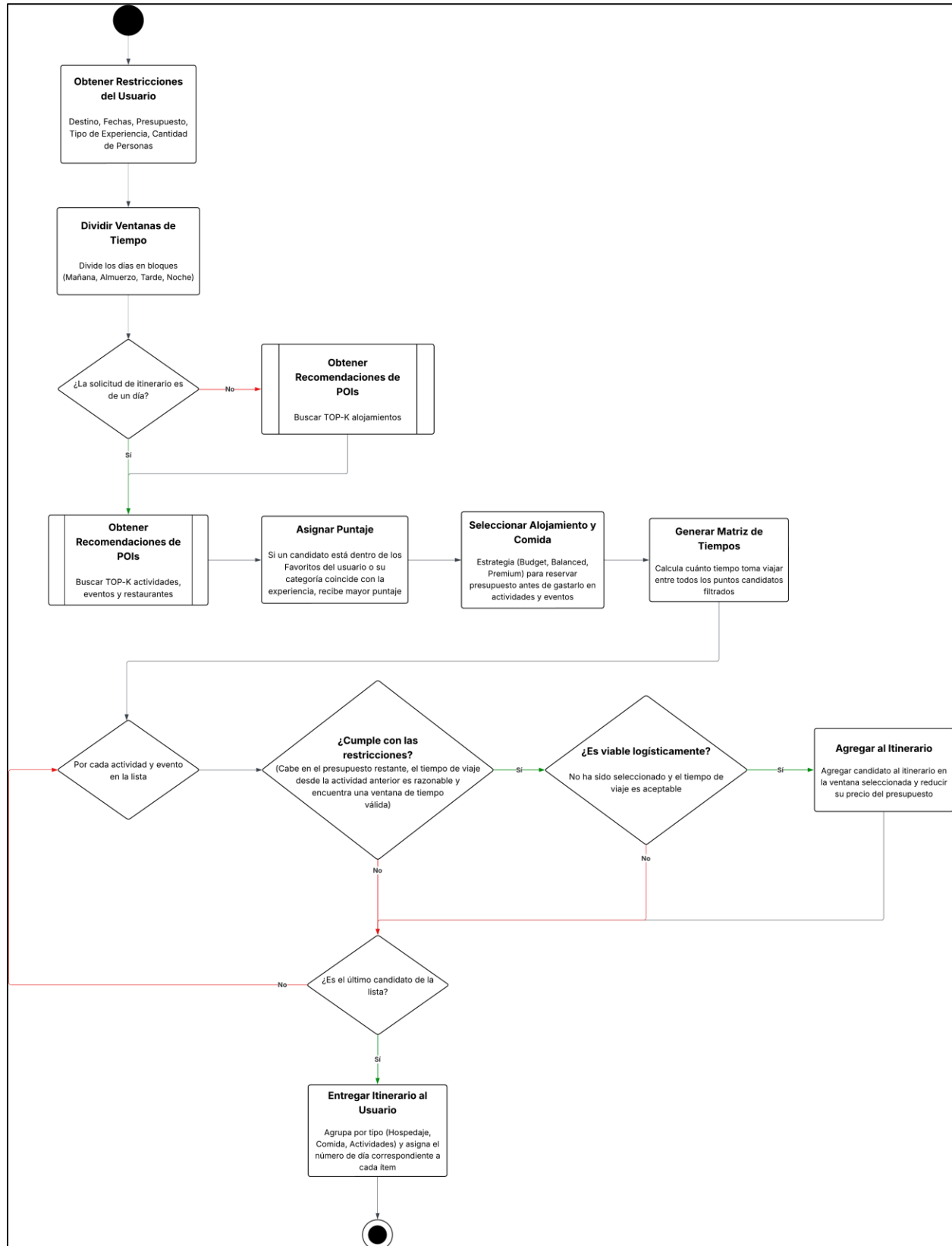


Figura 8. Flujo de la Generación de Itinerarios

Fuente: Elaboración Propia

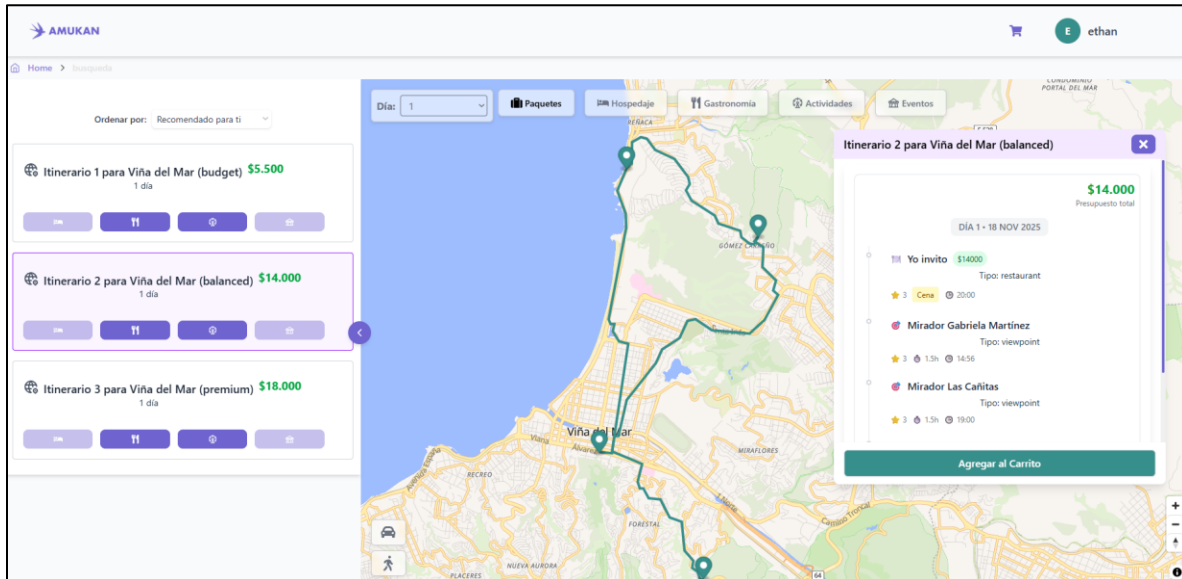


Figura 9. Ejemplo de itinerarios generados para el mismo día

Fuente: Equipo Amukan

3.5 Entorno de Desarrollo y Protocolo de Reproducibilidad

Para garantizar la reproducibilidad de los resultados, se definen las siguientes especificaciones técnicas:

- **Configuración del Entorno:** El sistema se orquestó mediante Docker Compose, utilizando una imagen base de `python:3.12-slim` para el servidor de aplicaciones y `pgvector/pgvector:pg15` para el motor de base de datos. El entorno experimental de desarrollo y ejecución de pruebas locales se configuró sobre una arquitectura de CPU comercial estándar (AMD Ryzen 5 5600, 6 núcleos, con 32 GB de memoria RAM), operando localmente sin requerir aceleración por hardware gráfico (GPU).
- **Versiones de Software Críticas:**
 1. **Backend (Core):** Django 5.2.4 y Django REST Framework 3.15.2.
 2. **Base de Datos:** PostgreSQL 15 con extensiones PostGIS 3 (geoespacial) y pgvector (vectorial).
 3. **Procesamiento Vectorial:** Numpy 1.26.4 para normalización matemática.
 4. **Geoprocésamiento:** geopandas 0.14.3 y shapely 2.0.4
 5. **Procesamiento de IA:** Inferencia semántica mediante el modelo `paraphrase-multilingual-MiniLM-L12-v2` vía Hugging Face API.
- **Dataset Experimental e ingesta automatizada:** La base de datos se pobló con un catálogo real de 1,999 Puntos de Interés (POIs) de Viña del Mar, extraídos y normalizados desde OpenStreetMap (OSM). Este set de datos se encuentra automatizado mediante scripts de gestión nativos de Django ejecutados sobre el contenedor de la aplicación (web). El pipeline de poblamiento e indexación sigue la secuencia operativa:


```
docker-compose exec web python manage.py import_comunas
```

```
docker-compose exec web python manage.py import_pois_vina
docker-compose exec web python manage.py populate_services
docker-compose exec web python manage.py generate_embeddings
```

Específicamente, el script `import_pois_vina.py` realiza de forma determinista la consulta a la API Overpass bajo las llaves estructurales *tourism*, *amenity*, *leisure* e *historic*, gestionando de manera asíncrona la resolución de direcciones vía Nominatim, la persistencia de recursos multimedia en *buckets S3* de MinIO y la asignación de un *rating* de infraestructura base acotado matemáticamente en [2.5, 4.5] basado en el equipamiento reportado en los tags de OSM.

3.6 Estándares de Calidad y Auditoría de Código

Para garantizar la mantenibilidad y el rigor técnico del software, se implementó un flujo de auditoría basado en herramientas de análisis estático y pruebas dinámicas. Este proceso permite certificar que la solución cumple con los estándares de la industria y responde a las restricciones definidas para el proyecto.

3.6.1 Análisis Estático y Complejidad

El control de calidad comienza con la normalización léxica de las aplicaciones `apps.travel` y `apps.recommendation`. Para ello, se utilizó el formateador Black, asegurando una base de código uniforme que sigue estrictamente la convención PEP 8. Complementariamente, se integró la herramienta Flake8 para auditar la lógica interna e identificar puntos de alta complejidad ciclomática.

En este análisis, el motor de optimización (`optimizer.py` y `optimization_core.py`) registró valores de complejidad de hasta 31 en la gestión de ventanas de tiempo. Dicha cifra se considera "inherente al dominio" del problema TOPTW, donde la integración de restricciones espaciales, temporales y presupuestarias en tiempo real exige una lógica de control densa.

3.6.2 Validación de Cobertura y Desacoplamiento (Unit Testing)

Como evidencia verificable de la modularidad del sistema, se ejecutó una suite de pruebas automatizadas utilizando "pytest" y "pytest-cov". La auditoría se centró en las capas lógicas de las dos aplicaciones núcleo del sistema, cuyos resultados se detallan a continuación:

Tabla 3. Reporte de Cobertura de Código (`apps.recommendation`)

Fuente: Elaboración Propia

Módulo	Cobertura	Interpretación Técnica
<code>metrics.py</code>	75%	Validación robusta de la lógica matemática y métricas semánticas.
<code>ml_model.py</code>	22%	Cobertura de la lógica local de procesamiento de vectores.

services.py	16%	Orquestación del filtrado basado en contenido y Top-K.
Total App	33%	Cobertura concentrada en la lógica crítica de negocio.

Tabla 4. Reporte de Cobertura de Código (apps.travel)

Fuente: Elaboración Propia

Módulo	Cobertura	Interpretación Técnica
services.py	15%	Orquestación del flujo de planificación y tipos de viaje.
optimizer.py	12%	Validación de la configuración del problema TOPTW.
data_provider.py	11%	Normalización de POIs y servicios esenciales.
views.py serializers.py	0%	Evidencia de separación total entre lógica de negocio y capa web.

3.6.3 Análisis Transversal de Modularidad

El cruce de datos entre la auditoría estática y la cobertura de pruebas dinámicas revela tanto las limitaciones estructurales como la suficiencia técnica del sistema desarrollado. Por un lado, la alta complejidad ciclomática de 31 registrada en la gestión de ventanas de tiempo (`optimizer.py`) evidencia que el algoritmo constructivo demanda un flujo de control denso y difícil de mantener, un costo técnico alto que es inherente a la resolución del problema TOPTW. No obstante, la suite de pruebas dinámicas demuestra que la arquitectura logró mitigar este impacto aislando dicha complejidad dentro del módulo logístico, lo que queda reflejado en la concentración de la cobertura en las clases puras de Python y en un 0% de alcance en la capa web (`views.py`). Este comportamiento métrico confirma un desacoplamiento efectivo del *framework*, asegurando que la lógica central de negocio no dependa de la infraestructura de presentación.

Por otro lado, los bajos porcentajes de cobertura global en las aplicaciones de recomendaciones (33%) y de viajes exponen las restricciones de alcance propias de este MVP. La suite automatizada se limitó a asegurar los flujos básicos *core* y a aislar las dependencias críticas de red mediante el uso de *mocks*, postergando la verificación exhaustiva de escenarios alternativos y el manejo complejo de excepciones secundarias, lo que constituye una deuda técnica explícita para fases futuras. A pesar de estas deficiencias en la profundidad del testeo, el diseño modular implementado se muestra suficiente para los objetivos trazados, ya que proporciona una base de código lo suficientemente limpia y compartimentada para permitir auditorías locales o una eventual transición hacia servicios autónomos sin comprometer la estabilidad global de la plataforma.

4 Validación y Resultados

La estrategia de validación se diseñó para mitigar la ausencia de un historial de interacciones reales (*ground truth*), utilizando un enfoque híbrido: una evaluación *Offline* técnica basada en la simulación de escenarios críticos con datos reales, y una auditoría técnica piloto de carácter cualitativo y exploratorio con un grupo focal reducido ($n=3$) para auditar la lógica operativa del sistema ante escenarios reales.

4.1 Evaluación Técnica (Offline)

El objetivo de esta fase fue medir la eficiencia técnica, la calidad algorítmica y la robustez del sistema frente a variaciones en sus parámetros fundamentales utilizando el catálogo de datos real de la plataforma.

4.1.1. Entorno Experimental y Dataset

Para asegurar la validez de las pruebas, se descartó el uso de datos sintéticos generados aleatoriamente porque carecen de la densidad semántica necesaria para evaluar un recomendador basado en contenido. Un *dataset* sintético podría simular coordenadas y precios, pero no puede replicar las relaciones semánticas sutiles entre una descripción textual (ej: "vistas al mar") y los intereses del usuario, que es precisamente el núcleo de la evaluación de los *embeddings*. Esta decisión es respaldada por Saifudin y Widiyaningtyas [20], quienes establecen que la validación efectiva de algoritmos basados en contenido requiere *datasets* con atributos descriptivos reales y complejos, ya que la simulación numérica no logra capturar la densidad semántica necesaria para entrenar modelos de similitud textual. La validación se ejecutó sobre la base de datos de prueba de la plataforma, la cual contiene 1.999 Puntos de Interés (POIs) únicamente de Viña del Mar, extraídos y normalizados desde OpenStreetMap (OSM). Con el fin de certificar la reproducibilidad de estas métricas *offline*, la suite automatizada se consolidó en un comando de gestión de Django ejecutado mediante la herramienta *memory-profiler* (`python -m memory_profiler manage.py validate_algorithms`). Este entorno inyecta de forma determinista las configuraciones de las "Personas" evaluadas, interactúa directamente con el catálogo indexado en la base de datos geoespacial, gatilla las llamadas vectoriales externas y audita el consumo neto de memoria en tiempo real.

Las métricas seleccionadas para evaluar el motor fueron el Tiempo de Respuesta (latencia de la API), la Diversidad (*Intra-List Diversity* mediante distancia coseno) y la Cobertura de Catálogo. Respecto a esta última, es imperativo destacar una restricción estructural inherente a la fuente de datos: los registros de OSM utilizados se caracterizan por una alta dispersión semántica, donde la mayoría de los POIs carecen de descripciones detalladas o reseñas, limitando la información textual disponible únicamente al Nombre del lugar y su Etiqueta de Categoría (ej: "Amenity: Restaurant"). Esto impone un desafío significativo para el motor de *embeddings*, que debe inferir similitud basándose en textos extremadamente cortos, lo que justifica la tendencia del sistema a concentrar sus recomendaciones en un subconjunto específico de ítems que sí poseen categorías explícitamente alineadas con los intereses declarados.

Ante la falta de historiales de popularidad externa, el atributo "*Rating*" utilizado en este estudio es un indicador sintético de calidad de infraestructura, calculado algorítmicamente durante la fase de importación según el nivel de equipamiento (WiFi, accesibilidad, servicios) reportado en los tags de OSM. Esto permite establecer un *baseline* "ingenuo" que prioriza el equipamiento técnico por sobre la relevancia semántica.

Finalmente, como decisión de diseño, se excluyeron los eventos temporales en esta validación automatizada. Dada la volatilidad de los eventos (fechas específicas) y la falta de estandarización en fuentes públicas, su inclusión introduciría ruido que impediría aislar el desempeño del algoritmo de optimización espacial y presupuestaria.

4.1.2. Diseño de Escenarios (Personas)

Para la simulación, se configuraron tres perfiles de usuario ("Personas") en el script de validación. Estos perfiles no son arbitrarios, sino que se derivan estrictamente de los clústeres identificados en el informe PLADETUR de Viña del Mar 2025 [1]:

Escenario A - "Escapada Romántica Nacional": El 77% de los visitantes son residentes de Chile, y el grupo de viaje predominante son parejas (60%). Sus principales motivaciones son "Sol y playa" (82%) y la Gastronomía. Por lo tanto, para la configuración del test se creó un usuario con intereses en ['Playas', 'Gastronomía Local', 'Romántico'], estadía de 2 días y presupuesto medio (\$150.000 total).

Escenario B - "Visitante Familiar por el Día": Un 36% de los visitantes viaja en grupo familiar. Críticamente, un 52% de los turistas nacionales declara que no pernocta en el destino (visita por el día), y un 26% tiene un gasto diario inferior a \$50.000. Para el test se configuró una restricción temporal estricta (0 noches, Full Day), presupuesto acotado (\$45.000) e intereses en ['Actividades Familiares', 'Parques de Aventura', 'Comida Callejera'].

Escenario C - "Turista Extranjero de Compras": El 86% de los turistas extranjeros provienen de Argentina. Este segmento se diferencia por un interés excepcionalmente alto en "Centros Comerciales" (65%) frente al 30% del turista nacional. En este caso se configuró un perfil con alta afinidad a ['Malls', 'Tiendas de Lujo', 'Alta Cocina'] y presupuesto alto (\$600.000).

4.1.3. Resultados de la Simulación (Recomendación)

En primera instancia, se contrastó el motor de Amukan contra dos enfoques comunes para cuantificar la ganancia en personalización y descubrimiento. Como se observa en la Tabla 5, el uso de *embeddings* multilingües permite superar las limitaciones de los sistemas basados puramente en calidad sintética de infraestructura (*Rating* de equipamiento) o coincidencia exacta.

Tabla 5. Comparativa de Métricas de Descubrimiento.

Fuente: Elaboración Propia

Algoritmo	Cobertura	Diversidad	Latencia	Justificación Técnica
Popularidad (<i>Rating</i>)	0.50%	0.652	~7 ms	Sesgo hacia ítems " <i>Blockbuster</i> ".
Léxico (TF-IDF)	0.12%	0.100	~15 ms	Falla ante sinónimos y barreras de idioma.
Amukan	1.40%	0.441	~866 ms	Balance entre relevancia y exploración.

El incremento del 180% en la cobertura respecto al *baseline* de popularidad valida la capacidad de Amukan para mitigar el sesgo de popularidad (*Popularity Bias*), comportamiento fundamental para otorgar visibilidad al "*Long Tail*" del catálogo y permitiendo que el usuario descubra lugares menos conocidos, pero altamente afines a su perfil [25].

Más allá de la comparativa, la ejecución de los algoritmos reales sobre la base de datos arrojó los indicadores promedio detallados en la Tabla 6. Estos resultados confirman que el sistema es capaz de mantener un equilibrio entre calidad algorítmica y eficiencia computacional.

Tabla 6. Resultados Evaluación Offline al Sistema de Recomendaciones

Fuente: Elaboración Propia

Métrica	Resultado	Interpretación
Latencia Promedio	~866 ms	El tiempo de respuesta se estabilizó en menos de 1 segundo. Este resultado valida la elección de la heurística constructiva (<i>Greedy</i>) frente a métodos exactos [21].
Diversidad	0.44	Se obtuvo un índice de diversidad cercano a 0.5 (en escala 0-1). Esto indica un balance positivo consistente con la literatura [7]: el sistema ofrece variedad temática dentro de los intereses del usuario, evitando sugerir, por ejemplo, 5 restaurantes idénticos consecutivamente.
Cobertura	1.40%	El sistema activó 28 POIs únicos de los 1.999 disponibles. Este valor bajo es esperable en una prueba con solo 3 perfiles de usuario muy específicos. Además, valida que el sistema filtra eficazmente la gran cantidad de POIs no turísticos presentes en OSM (ej: bancos, farmacias), concentrándose en la oferta relevante.

El bajo indicador de cobertura (1.40%) debe interpretarse bajo la restricción de la fuente de datos. Al depender exclusivamente de pares Nombre-Categoría, el modelo de lenguaje tiende a agrupar fuertemente los ítems. Este fenómeno es consistente con el problema de Dispersión de Datos (*Sparsity*) identificado por Saifudin y Widiyaningtyas [20] en sistemas de filtrado basado en contenido, donde la falta de descripciones ricas limita la capacidad del algoritmo para diferenciar ítems del "*Long Tail*", sacrificando cobertura por precisión [7]. Por ejemplo, un POI etiquetado solo como "Plaza" sin atributos descriptivos adicionales generará un vector muy similar a cualquier otra "Plaza", reduciendo la capacidad del sistema para distinguir matices (ej. "Plaza con juegos" vs "Plaza histórica").

Sin embargo, esto valida la robustez del modelo `paraphrase-multilingual-MiniLM-L12-v2`: incluso con inputs mínimos, logró generar recomendaciones temáticamente coherentes (ej. asociar "Muelle" con el perfil "Playa"), demostrando que el conocimiento pre-entrenado del modelo compensa parcialmente la falta de descripciones ricas en la base de datos.

Es importante destacar que, durante la escritura de este documento, Hugging Face migró su infraestructura de inferencia gratuita hacia un sistema de enrutamiento *Serverless* (`router.huggingface.co`). Si bien este cambio técnico obligó a reestructurar las llamadas a la API, el incidente permitió validar de forma empírica la modularidad del

sistema: el alto nivel de desacoplamiento de la arquitectura permitió resolver la incompatibilidad ajustando exclusivamente el módulo encargado de la comunicación con la IA (`ml_model.py`), sin necesidad de alterar la lógica de negocio en el orquestador de recomendaciones o en el generador de itinerarios.

Este ajuste en la infraestructura explica la variación observada en la latencia promedio, la cual se estabilizó en ~866 ms frente a mediciones preliminares de ~685 ms. Este incremento se atribuye al tiempo de resolución del enrutador externo y a la implementación de capas adicionales de normalización de vectores en el cliente local para asegurar la integridad de la similitud coseno.

4.1.4. Resultados del Generador de Itinerarios (Factibilidad)

A continuación, se detalla el reporte consolidado extraído de la ejecución del *script* de validación automatizada para cada uno de los perfiles turísticos simulados, donde se integró la herramienta *memory-profiler* para auditar el impacto neto en la memoria RAM del servidor durante la ejecución de los algoritmos:

Tabla 7. Reporte de Desempeño y Consumo por Perfil de Usuario (Métricas Offline)

Fuente: Elaboración Propia

Variable / Perfil de Prueba	Escenario A: Escapada Romántica	Escenario B: Familiar por el Día	Escenario C: Turista Extranjero
Intereses Declarados	Playas, Gastronomía, Romántico	Act. Familiares, Aventura, Comida Callejera	Malls, Vida Nocturna, Cocina
Restricción Temporal	2 días (Con Hotel)	Full Day (Sin Hotel)	4 días (Con Hotel)
Presupuesto Máximo (\$)	\$150.000	\$45.000	\$600.000
Presupuesto Ejecutado (\$)	\$146.000 (OK)	\$14.000 (OK)	\$402.000 (OK)
Cumplimiento Logístico	Exitoso (Hotel: Sí)	Exitoso (Hotel: No)	Exitoso (Hotel: Sí)
Latencia Recomendador	959.16 ms	868.95 ms	769.21 ms
Diversidad (Intra-List)	0.4496	0.4132	0.4615
Memoria Base Django	486.1 MiB	489.5 MiB	490.0 MiB
Delta Memoria TOPTW	+2.7 MiB	+0.5 MiB	+0.5 MiB

La validación de factibilidad se centró en someter al algoritmo *Greedy* a pruebas estrictas de cumplimiento de restricciones duras (*Hard Constraints*), confirmando la robustez lógica del generador. Un hallazgo destacado se observó en la prueba de lógica "*Full Day*" (Escenario B), donde el sistema interpretó correctamente la restricción de cero pernотaciones. En consecuencia, se generó un itinerario compuesto exclusivamente por parques y oferta gastronómica, excluyendo exitosamente la categoría de alojamiento.

Por otro lado, la prueba de adherencia al presupuesto demostró que, en todos los escenarios evaluados, el costo total estimado del itinerario se mantuvo por debajo del límite declarado por el usuario. En el Caso B, caracterizado por un bajo presupuesto (\$45.000), el sistema generó un plan eficiente de \$14.000, mientras que en el Caso C (presupuesto alto), el algoritmo evidenció su capacidad de escalar la oferta al incluir servicios de mayor valor, resultando en un plan de \$402.000. Adicionalmente, se ejecutó una prueba de estrés (presupuesto \$0), donde el sistema generó correctamente un itinerario de costo cero compuesto por atractivos naturales y plazas, validando la resiliencia del orquestador ante la ausencia total de fondos.

Complementando la factibilidad del negocio, el perfilamiento de infraestructura arrojó conclusiones críticas sobre la eficiencia del software en condiciones operativas reales. La latencia total del recomendador se estabilizó de forma consistente bajo el segundo de respuesta en todos los perfiles, promediando ~ 866 ms, lo que valida la viabilidad del patrón de comunicación sincrónica. En términos de consumo de hardware, la memoria base del entorno Django se situó en una franja sumamente controlada de ~ 486 MiB a ~ 490 MiB. El hito técnico más relevante corresponde al delta de memoria neta requerido por el núcleo del motor logístico para resolver la variante matemática TOPTW: el algoritmo constructivo *Greedy* requirió un incremento máximo de apenas 2.7 MiB en el itinerario de múltiples jornadas (Escenario A) y un impacto marginal de 0.5 MiB en escenarios de un solo día, ratificando de manera empírica que el sistema es computacionalmente sostenible, liviano y apto para ser desplegado en arquitecturas de servidores básicos.

4.1.5. Análisis de Sensibilidad

Con el propósito de evaluar la robustez y estabilidad del sistema ante variaciones en su configuración interna, se ejecutó un análisis de sensibilidad sobre los hiperparámetros clave: el factor de decaimiento temporal (γ) en el perfilado dinámico y el impacto de K en la preselección de ítems sobre la diversidad y latencia frente al catálogo de 1.999 POIs.

Tabla 8. Análisis de Sensibilidad del Parámetro K (Top-k) sobre el Catálogo Real.

Fuente: Elaboración Propia

Valor de K	Diversidad Semántica	Latencia (ms)	Costo-Beneficio
K=5	0.266	~ 420	Baja variedad temática.
K=20	0.257	~ 850	Punto de equilibrio óptimo.
K=50	0.252	~ 1100	Diversidad estable; latencia al límite.

A partir de los resultados de la Tabla 8, se determinó que $K=20$ es el punto de equilibrio óptimo, logrando una latencia de 850 ms que asegura una experiencia fluida dentro del margen establecido. Incrementar el volumen a $K=50$ resulta ineficiente, ya que la diversidad semántica solo mejora un 0,5% mientras que el tiempo de respuesta penaliza la experiencia de usuario al superar el límite aceptable. Esta configuración garantiza suficiencia logística al proveer ítems suficientes para viajes de varios días, permitiendo que el algoritmo *Greedy* optimice eficazmente las rutas por cercanía geográfica.

Complementariamente a la definición del espacio de búsqueda, la efectividad de estas recomendaciones depende de la estabilidad y evolución del perfil del usuario en el tiempo. En este contexto, la elección de un factor $\gamma = 0.95$ para el perfilado dinámico busca mitigar la retroalimentación de sesgos y la excesiva concentración en ítems

populares. Mediante simulaciones unitarias en `test_mock.py`, se contrastó este valor frente a otros factores para validar la curva de aprendizaje del perfil ante una secuencia de 15 búsquedas constantes con un incremento de 0.1. Como se observa en la Tabla 9, factores iguales o inferiores a 0.80 presentan un estancamiento prematuro que impide al interés alcanzar su relevancia máxima, mientras que un factor de 0.99 resulta en una rigidez excesiva que dificulta la actualización de las preferencias. En cambio, el valor 0.95 permite un crecimiento asintótico controlado que alcanza la saturación (1.0) en la octava iteración, logrando un buen equilibrio entre la estabilidad de la memoria histórica y la adaptabilidad ante nuevos intereses del turista.

Tabla 9. Estudio de Sensibilidad: Dinámica de Convergencia del Perfil (γ).

Fuente: Elaboración Propia

Iteración	$\gamma=0.50$	$\gamma=0.80$	$\gamma=0.90$	$\gamma=0.95$ (Amukan)	$\gamma=0.99$
1	0.3500	0.5000	0.5500	0.5750	0.5950
3	0.2375	0.5000	0.6355	0.7139	0.7822
5	0.2094	0.5000	0.7048	0.8393	0.9656
7	0.2023	0.5000	0.7609	0.9525	1.0000 (Límite)
9	0.2006	0.5000	0.8063	1.0000 (Límite)	1.0000
11	0.2001	0.5000	0.8431	1.0000	1.0000
13	0.2000	0.5000	0.8729	1.0000	1.0000
15	0.2000	0.5000	0.8971	1.0000	1.0000

4.2 Inspección Operativa Basada en Usuarios (Control de Calidad Técnico)

Como complemento a la validación automatizada *offline*, se ejecutó una verificación exploratoria piloto mediante un grupo focal reducido ($n=3$). Siguiendo los lineamientos conceptuales del marco de evaluación centrado en el usuario (*User-Centric Framework*) [8], el objetivo de esta fase no consistió en medir métricas cuantitativas de usabilidad en la interfaz de usuario, sino en realizar una auditoría de caja negra enfocada en la calidad percibida de las recomendaciones, la precisión de las descripciones semánticas y la viabilidad logística de los itinerarios construidos de forma dinámica.

Por ende, esta evaluación opera estrictamente como un control de calidad de software tipo "*Sanity Check*" y sus hallazgos no persiguen la generalización estadística de la experiencia de usuario. Este enfoque cualitativo permite validar la competencia operativa del sistema bajo los principios de una Inteligencia Artificial confiable (*Trustworthy AI*), actuando como un mecanismo de control temprano frente a anomalías de ordenamiento espacial que resultan invisibles para las métricas e indicadores automáticos. Un ejemplo crítico inspeccionado durante esta auditoría fue el denominado "efecto ping-pong", un quiebre logístico donde el algoritmo satura la ruta obligando al usuario a trasladarse alternadamente entre puntos distantes debido a una alta afinidad semántica, ignorando la coherencia geográfica. La participación de usuarios reales permitió auditar la resiliencia del orquestador local frente a este tipo de comportamientos, asegurando la solidez de las reglas de negocio en escenarios reales antes de un despliegue masivo.

4.2.1 Metodología y Caracterización de la Muestra

Se utilizó un muestreo estratégico para cubrir distintos perfiles de comportamiento turístico, asegurando la representatividad de los segmentos clave identificados en el estudio de demanda.

Participante 1: La Experta Local (Validación Semántica)

- **Perfil:** Mujer, 47 años, residente de Quilpué.
- **Características:** Conocedora de la oferta turística tradicional, busca redescubrir el destino con experiencias específicas en "Naturaleza" y "Gastronomía".
- **Escenario de Prueba:** Planificación de fin de semana en pareja con presupuesto medio (\$50.000) y foco exclusivo en actividades (sin alojamiento).
- **Rol en la auditoría:** Evaluar si el sistema es capaz de sorprender a alguien que ya conoce el territorio y si la interpretación de gustos es precisa.

Participante 2: El Estudiante Pragmático (Estrés de Restricciones)

- **Perfil:** Hombre, 24 años, estudiante residente de Quilpué.
- **Características:** Usuario intensivo con restricciones fuertes de presupuesto y movilidad (dependencia de transporte público/caminata).
- **Escenario de Prueba:** Planificación de "último minuto" (mismo día, 20:00 hrs) y día siguiente, con presupuesto muy bajo (\$15.000).
- **Rol en la auditoría:** Poner a prueba los límites del algoritmo *Greedy* frente a escasez de tiempo y dinero, y evaluar la coherencia geográfica de los traslados.

Participante 3: La Turista Internacional (Blind Test)

- **Perfil:** Mujer, 23 años, residente en México (nunca ha visitado Chile).
- **Características:** Desconocimiento total del destino y del costo de vida local.
- **Escenario de Prueba:** Planificación de fin de semana completo con presupuesto holgado (\$500.000) y un día con presupuesto ajustado.
- **Rol en la auditoría:** Evaluar la barrera de entrada por falta de contexto en los datos y la coherencia del itinerario ante la abundancia de presupuesto.

4.2.2. Protocolo de Entrevistas

La sesión consistió en una tarea de planificación libre seguida de una entrevista semi-estructurada orientada a responder cuatro preguntas de control técnico:

1. **Relevancia:** "Según las 3 primeras recomendaciones, ¿sientes que el sistema comprendió tus gustos?"
2. **Veracidad:** "¿La información coincide con la realidad que conoces?" (Para locales).
3. **Logística:** "¿Es posible realizar la ruta en los tiempos dados?"
4. **Semántica de Datos:** "Sin conocer la ciudad, ¿entiendes qué es este lugar solo con su nombre?" (Para extranjera).

4.2.3 Hallazgos Preliminares

La triangulación de las tres entrevistas arrojó hallazgos que validan la inteligencia del recomendador, pero exponen limitaciones críticas en la lógica espacial del generador de itinerarios y la riqueza de los datos base.

Precisión Semántica y "Serendipia" (Validado): El motor de recomendación demostró una alta capacidad para interpretar intereses complejos. La Participante 1 destacó que el sistema recomendó correctamente cafeterías específicas, validando la precisión de los *embeddings* para capturar sub-intereses. A pesar de ser residentes, los Participantes 1 y 2 reportaron desconocer varios de los lugares recomendados. El Participante 2 validó la "inteligencia emocional" del sistema al recibir como recomendación un lugar con alto valor personal (locación de una foto significativa), sugiriendo una alineación exitosa entre el perfilado y la oferta oculta de la ciudad. Este hallazgo valida la capacidad del sistema para generar *Serendipia*, definida como la cualidad de una recomendación de ser simultáneamente "inesperada y útil", lo cual es superior a la simple novedad [7].

Comportamiento ante el Presupuesto (Validado con Observaciones): El algoritmo respondió correctamente a las restricciones económicas, ajustando la oferta según el poder adquisitivo simulado. En el caso del Participante 2 (\$15.000), el sistema priorizó actividades gratuitas y alimentación de bajo costo, cumpliendo la restricción. La Participante 1 valoró positivamente que el sistema excluyera el alojamiento en itinerarios donde el presupuesto se enfocaba en actividades, validando la regla de negocio implementada. La Participante 3 (presupuesto alto) evidenció un comportamiento "demasiado codicioso" (*too greedy*) del algoritmo, llenando cada bloque horario disponible. Al tener fondos demasiado altos, el sistema intentó llenar cada bloque horario disponible con actividades, generando un itinerario saturado y agotador. Este comportamiento se alinea con lo que Marzal y Sebastia [18] clasifican como "*Busy Trips*", donde el algoritmo prioriza maximizar la cantidad de visitas sobre el tiempo de ocio (*spare time*), al no contar con una configuración explícita de preferencia de ritmo de viaje.

Incoherencia Espacial y "Efecto Ping-Pong" (Hallazgo Crítico): Si bien el algoritmo agrupa actividades geográficamente, la auditoría reveló fallas en la secuencia lógica entre categorías distintas (Comida vs. Actividad). El Participante 2 señaló que los tiempos de traslado asumidos por el sistema parecen corresponder a viajes en automóvil, resultando inviables o incómodos para turistas que dependen del transporte público o caminata.

La Participante 3 detectó que el itinerario la hacía desplazarse largas distancias para una actividad y luego regresar cerca del punto de inicio para comer, generando un patrón de movimiento ineficiente ("Ping-Pong"). Esto indica que el algoritmo optimiza la utilidad global pero no penaliza suficientemente la distancia entre el *step* anterior y el siguiente cuando cambia la categoría del POI.

Según Granda y Vitoriano [21], este es un defecto intrínseco de las heurísticas constructivas, las cuales toman decisiones paso a paso basándose en el estado actual sin "ver" las consecuencias futuras (ceguera al futuro), lo que a menudo resulta en rutas con retrocesos logísticos.

Analizando esto bajo el marco de IA Confiable [26], este error logístico daña la percepción de Competencia del sistema. Aunque hubo "Calidez" (buena intención al recomendar lugares gustados), la falta de robustez lógica en la ruta disminuye la confianza del usuario [27], identificando la penalización de distancia como una mejora crítica necesaria.

La Brecha de Información en OSM (Confirmada): La limitación de los datos de OpenStreetMap (Título + Categoría) fue identificada como la barrera de usabilidad más

severa. La Participante 1 indicó que "no todos los lugares se entienden por su nombre", requiriendo conocimiento previo para decidir. Para la Participante 3, esta carencia fue crítica. Al leer nombres de plazas o hitos sin descripciones ni fotos, le fue imposible valorar la recomendación, confirmando que la riqueza de los metadatos es un requisito indispensable para la conversión de turistas internacionales.

5 Conclusiones y Trabajo Futuro

El desarrollo de la plataforma Amukan ha permitido validar la factibilidad técnica de construir un sistema de recomendación de itinerarios turísticos personalizados en un entorno de alta incertidumbre y escasez de recursos. A través de la implementación de un Producto Mínimo Viable (MVP), este trabajo logró abordar la fragmentación de la oferta turística de Viña del Mar, integrando tecnologías de inteligencia artificial y optimización geoespacial en una arquitectura de bajo costo operativo. El cumplimiento de los objetivos planteados se refleja en la capacidad del sistema para operar bajo restricciones estrictas de presupuesto y cómputo. La adopción de una arquitectura monolítica modular, complementada con servicios externos para la vectorización, demostró ser una estrategia eficiente, logrando un consumo de memoria RAM base de entre 486 MiB y 490 MiB, y un tiempo de respuesta promedio de 866 ms, lo cual asegura una experiencia de usuario fluida sin requerir infraestructura costosa.

La validación cualitativa confirmó que el uso de modelos de *embeddings*, específicamente Sentence Transformers, permite superar el enfoque léxico tradicional (TF-IDF). Esta tecnología ofrece recomendaciones con cierto grado de serendipia, ajustándose a los intereses latentes del usuario incluso cuando estos son complejos. Asimismo, el algoritmo *Greedy* adaptado para el problema TOPTW logró respetar las restricciones duras de presupuesto y tiempo en los escenarios simulados, requiriendo un incremento de memoria neta máximo de apenas 2.7 MiB para rutas multi-día y de 0.5 MiB para un solo día. Además, la modularidad se validó empíricamente ante una migración imprevista en la infraestructura de Hugging Face hacia su nuevo enrutador *serverless*, la cual se resolvió modificando exclusivamente el componente cliente local sin alterar la lógica de negocio del sistema.

Sin embargo, la fase de validación expuso limitaciones críticas, principalmente relacionadas con la pobreza semántica de los datos públicos disponibles en OpenStreetMap y la generación de rutas con ineficiencias logísticas. Al depender de pares de nombre y categoría, la cobertura global del catálogo se situó en un 1.40%, lo que refleja cómo la falta de descripciones ricas limita la capacidad del algoritmo para diferenciar matices entre los ítems, sacrificando cobertura por precisión. Por otro lado, la auditoría con usuarios evidenció desplazamientos innecesarios de ida y vuelta debidos al efecto ping-pong, además de una saturación excesiva de actividades en bloques horarios cuando los presupuestos eran muy altos, lo que se alinea con la definición de viajes sobreocupados al no contar con una preferencia explícita del ritmo de viaje.

Para consolidar la propuesta de valor de Amukan, el trabajo futuro deberá enfocarse prioritariamente en el despliegue operativo de las estrategias de enriquecimiento de datos ya contempladas en la plataforma global. A medida que la información base de OpenStreetMap sea complementada por descripciones detalladas y material visual aportado directamente por los actores locales mediante crowdsourcing, se espera resolver la barrera de confianza detectada en el MVP y mejorar la precisión de los *embeddings*. En paralelo, es indispensable llevar a cabo una evaluación longitudinal del módulo de perfilado dinámico para calibrar los hiperparámetros del mecanismo de

decaimiento temporal y normalización. Dado que las ponderaciones asignadas a las interacciones (+0.1, +0.2 y +0.3) responden a una aproximación heurística inicial, se requiere un estudio extendido con datos de uso masivo para evaluar la convergencia del perfil ante variaciones del factor de decaimiento, o bien, transicionar hacia modelos de aprendizaje supervisado y técnicas de optimización bayesiana.

Asimismo, es necesario refinar la lógica de tiempos de traslado, la cual fue percibida como rígida por algunos usuarios. Se propone incorporar funcionalidades que permitan al propio usuario ajustar los parámetros de desplazamiento, seleccionando velocidades personalizadas o modalidades de transporte, para que la estimación de los tiempos de viaje se alinee con sus capacidades y preferencias reales. Finalmente, la optimización espacial requiere un análisis profundo para mitigar el "efecto ping-pong" observado en las rutas. Para ello, se plantea realizar pruebas comparativas sobre tres posibles soluciones técnicas. En primera instancia, dado que la selección de alojamiento y comida responde a una regla de negocio ejecutada fuera del bucle de optimización, se debe verificar si el algoritmo está considerando estos puntos preseleccionados como nodos de origen para la siguiente actividad; de no ser así, la solución inmediata sería ajustar la memoria secuencial para que el cálculo de distancia contemple el último punto físico visitado, sea este recreativo o de servicio. Una segunda vía de solución implica integrar toda la lógica de selección de servicios esenciales dentro del modelo matemático TOPTW, eliminando la separación actual entre heurísticas de negocio y optimización recreativa. Si estas correcciones estructurales no fuesen suficientes, se deberá abordar la tercera causa probable: la limitación intrínseca de "ceguera al futuro" del algoritmo *Greedy*, lo cual obligaría a migrar hacia metaheurísticas con visión global para garantizar la coherencia geográfica del itinerario.

Agradecimientos. Quiero agradecer sinceramente al equipo de Amukan. Especialmente a Zada Riquelme y Ashley Iturriaga, por ser capaces de plasmar nuestra visión en una interfaz tanto atractiva como funcional; y a Benjamín Lillo y Natanael Roque, por quedarse trabajando hasta el último solucionando errores.

A mis amigos Yarko, Sofía y Max, quienes me acompañaron y me dieron su apoyo constante durante este último tramo de la escritura del documento.

Por último, dedico este trabajo a mi familia, por su comprensión, paciencia y apoyo incondicional durante las largas jornadas de desarrollo y encierro que exigió este proyecto. Sin ustedes, este logro no habría sido posible.

6 Referencias

- [1] Municipalidad de Viña del Mar and DTS Consultores, "INFORME FINAL PLADETUR SUSTENTABLE COMUNA DE VIÑA DEL MAR," Viña del Mar, Jun. 2025. Accessed: Oct. 03, 2025. [Online]. Available: https://visitavina.munivina.cl/wp-content/uploads/2025/10/INFORME-FINAL_-PLADETUR-VDM.pdf#page=363.09
- [2] CORFO, "Servicio de Consultoría Experta para Identificación de Oportunidades y Levantamiento de Brechas y la Confección de la Hoja de Ruta," Santiago, Sep. 2015. Accessed: Oct. 03, 2025. [Online]. Available: <https://transformaturismo.cl/wp-content/uploads/2020/08/Informe-Diagnostico-Brechas-y-Oportunidades-Transforma-Turismo.pdf>

- [3] Gobierno Regional de Valparaíso, "Plan Estratégico de desarrollo turístico 2021-2025 Region de Valparaíso," Valparaíso, 2021. Accessed: Sep. 27, 2025. [Online]. Available: <https://www.gobiernovalparaiso.cl/administracion/archivo/archivoDocumento/2025/documento20250226194356.pdf>
- [4] Mariano Tappatá and Estefanía Lotitto, "Maximizando los beneficios del uso de Plataformas Traveltech en América Latina," Washington, D.C., May 2021. Accessed: Sep. 27, 2025. [Online]. Available: <https://publications.iadb.org/publications/spanish/document/Maximizando-los-beneficios-del-uso-de-plataformas-Traveltech-en-America-Latina.pdf>
- [5] Departamento de Control de Gestión y Sistemas de Información, "Índice de Madurez de Transformación Digital Sector Público 2024," 2024. Accessed: Oct. 03, 2025. [Online]. Available: https://www.contabilidadysistemas.cl/contenido/file/reporte/11_1728066452.pdf
- [6] Subsecretaría de Turismo, "ESTRATEGIA NACIONAL DE TURISMO 2030," Santiago, Feb. 2022. Accessed: Sep. 27, 2025. [Online]. Available: <https://www.subturismo.gob.cl/wp-content/uploads/2022/03/Estrategia-Nacional-de-Turismo-2030.pdf>
- [7] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach, *Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity*. Association for Computing Machinery, 2010. Accessed: Nov. 30, 2025. [Online]. Available: <https://dl.acm.org/doi/10.1145/1864708.1864761>
- [8] P. Pu and L. Chen, "A User-Centric Evaluation Framework of Recommender Systems," 2010.
- [9] D. Liu, L. Wang, Y. Zhong, Y. Dong, and J. Kong, "Personalized Tour Itinerary Recommendation Algorithm Based on Tourist Comprehensive Satisfaction," *Applied Sciences (Switzerland)*, vol. 14, no. 12, Jun. 2024, doi: 10.3390/app14125195.
- [10] M. V. Gayathri, S. S. Ram, M. Vinodhan, and M. Ganesh Kumar, "Smart and Dynamic AI-Powered Travel Planning: A Machine Learning Approach for Personalized and Real-Time Itinerary Generation," *International Research Journal on Advanced Engineering and Management*, vol. 3, Mar. 2025, doi: 10.47392/IRJAEM.2025.087.
- [11] F. Ghobadi, A. Divsalar, H. Jandaghi, and R. Barzegar Nozari, "An integrated recommender system for multi-day tourist itinerary," *Appl. Soft Comput.*, vol. 149, Dec. 2023, doi: 10.1016/j.asoc.2023.110942.
- [12] C. Binucci, G. Biondi, E. Di Giacomo, W. Didimo, A. Fortuni, and G. Liotta, "CHIP: a Recommender System and a Travel Planner for Cultural Tourism," 2024. [Online]. Available: <https://mozart.diei.unipg.it/rasta/>.
- [13] J. P. Esper, L. de S. Fraga, A. C. Viana, K. V. Cardoso, and S. L. Correa, "+Tour: Recommending personalized itineraries for smart tourism," Feb. 2025, doi: 10.1016/j.comnet.2025.111118.
- [14] N. L. Ho, R. K.-W. Lee, and K. H. Lim, "SBTRec- A Transformer Framework for Personalized Tour Recommendation Problem with Sentiment Analysis," Nov. 2023, [Online]. Available: <http://arxiv.org/abs/2311.11071>
- [15] S. Chaudhuri *et al.*, "TripCraft: A Benchmark for Spatio-Temporally Fine Grained Travel Planning," 2025.
- [16] sentence-transformers, "paraphrase-multilingual-MiniLM-L12-v2." Accessed: Oct. 04, 2025. [Online]. Available: <https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2>

- [17] K. Otaki and Y. Baba, "Travel itinerary recommendation using interaction-based augmented data," *Expert Syst. Appl.*, vol. 269, Apr. 2025, doi: 10.1016/j.eswa.2024.126294.
- [18] E. Marzal and L. Sebastia, "Solving the tourist trip design problem with time windows and variable profit using incremental local search," *Appl. Soft Comput.*, vol. 155, Apr. 2024, doi: 10.1016/j.asoc.2024.111399.
- [19] B. Granda and B. Vitoriano, "The team orienteering problem with variable time windows," *International Transactions in Operational Research*, 2024, doi: 10.1111/itor.13570.
- [20] I. Saifudin and T. Widiyaningtyas, "Systematic Literature Review on Recommender System: Approach, Problem, Evaluation Techniques, Datasets," *IEEE Access*, vol. 12, pp. 19827–19847, 2024, doi: 10.1109/ACCESS.2024.3359274.
- [21] B. Granda and B. Vitoriano, "A Greedy Constructive Heuristic for Solving the Team Orienteering Problem with Variable Time Windows," *International Journal of Computational Intelligence Systems*, vol. 18, no. 1, Dec. 2025, doi: 10.1007/s44196-025-00797-5.
- [22] S. Halder, K. H. Lim, J. Chan, and X. Zhang, "Efficient itinerary recommendation via personalized POI selection and pruning," *Knowl. Inf. Syst.*, vol. 64, no. 4, pp. 963–993, Apr. 2022, doi: 10.1007/s10115-021-01648-3.
- [23] I. Hapsari, H. Iswadi, and Y. O. Cahyadi, "Tourism Route Planning Route Based on Tourist Category using Orienteering Problem with Time Windows," 2019. [Online]. Available: www.banyuwangitourism.com,
- [24] R. Su and X. Li, "Modular Monolith: Is This the Trend in Software Architecture?," in *Proceedings - 2024 IEEE/ACM International Workshop New Trends in Software Architecture, SATrends 2024*, Institute of Electrical and Electronics Engineers Inc., Aug. 2024, pp. 10–13. doi: 10.1145/3643657.3643911.
- [25] A. Klimashevskaya, D. Jannach, M. Elahi, and C. Trattner, "A survey on popularity bias in recommender systems," *User Model. User-adapt. Interact.*, vol. 34, no. 5, pp. 1777–1834, Nov. 2024, doi: 10.1007/s11257-024-09406-0.
- [26] Y. Li, B. Wu, Y. Huang, and S. Luan, "Developing trustworthy artificial intelligence: insights from research on interpersonal, human-automation, and human-AI trust," 2024, *Frontiers Media SA*. doi: 10.3389/fpsyg.2024.1382693.
- [27] S. Wang, X. Zhang, Y. Wang, and F. Ricci, "Trustworthy Recommender Systems," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 4, Jul. 2024, doi: 10.1145/3627826.