Universidad de la República

Facultad de Ingeniería

Recuperación de Información y Recomendaciones en la Web

Amazon Copilot

Autores:
Juan Pablo Conde
Xavier Iribarnegaray
Juan Pablo Sotelo

Profesores:
Libertad Tansini

27 de junio de 2025

Índice

1. Introducción

Amazon Copilot constituye un sistema diseñado para optimizar la búsqueda y selección de productos en plataformas de comercio electrónico. Basado en un conjunto de datos de productos de Amazon, el sistema implementará técnicas avanzadas de recuperación de información y procesamiento de lenguaje natural para proporcionar una experiencia de compra intuitiva y personalizada.

El proyecto integra tres funcionalidades principales: un sistema de búsqueda híbrido que combina técnicas semánticas y tradicionales para ofrecer resultados más relevantes; un asistente conversacional (Agente de Inteligencia Artificial) que proporciona una interfaz de búsqueda interactiva, refinando requerimientos mediante diálogo natural y utilizando como mecanismo subyacente el sistema de búsqueda híbrido implementado; y un sistema de recomendación que, basándose en los productos seleccionados en el carrito, emplea el mismo agente para identificar artículos similares o complementarios, mejorando así la experiencia de descubrimiento de productos.

2. Funcionalidades del Sistema

En esta sección se presentan las tres principales funcionalidades que se implementarán en Amazon Copilot.

2.1. Búsqueda Híbrida de Productos

Este componente combina técnicas de búsqueda semántica con métodos tradicionales de correspondencia textual para optimizar resultados. Incorpora representaciones vectoriales semánticas (embeddings) para capturar el significado contextual de las consultas, mantiene búsqueda por términos exactos para consultas específicas e implementa filtrado por categorías. El sistema ordena los resultados mediante un modelo que pondera tanto la relevancia semántica como la coincidencia sintáctica tradicional, logrando un equilibrio óptimo que aprovecha las fortalezas de ambos enfoques para mostrar los productos más pertinentes al usuario.

2.2. Búsqueda Conversacional con Agente de IA

El asistente conversacional proporciona una interfaz de búsqueda interactiva en lenguaje natural. Este agente mantiene el contexto durante la conversación, refina progresivamente los requerimientos del usuario mediante preguntas específicas, y utiliza la API de búsqueda híbrida como herramienta subyacente para ofrecer resultados personalizados. Resulta especialmente útil para usuarios sin conocimientos específicos sobre los productos que buscan.

2.3. Sistema de Recomendación con Agente de IA

Este sistema analiza los productos seleccionados en el carrito para sugerir artículos complementarios o alternativos. Opera automáticamente al visualizar el carrito, generando recomendaciones basadas en productos frecuentemente adquiridos conjuntamente y alternativas similares. Presenta explicaciones concisas sobre cada recomendación y adapta las sugerencias dinámicamente según el contenido del carrito y el historial de la conversación con el usuario.

3. Arquitectura del Sistema

Esta sección describe los componentes principales del sistema, sus responsabilidades e interacciones, así como las tecnologías propuestas para su implementación.

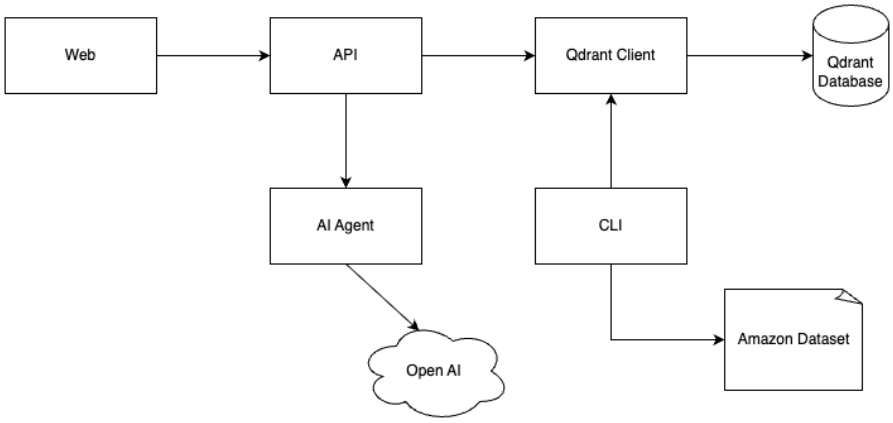

Figura 1: Diagrama de alto nivel de la arquitectura del sistema.

3.1. Web

El componente Web constituye la interfaz de usuario principal del sistema, proporcionando una experiencia de usuario moderna e interactiva para la búsqueda y exploración de productos. La aplicación implementa URLs compartibles donde todos los parámetros de búsqueda y filtrado están codificados en la URL, permitiendo a los usuarios compartir resultados específicos y reproducir exactamente los mismos resultados.

Se eligió Next.js 15 [12] con React por la experiencia previa del equipo y las ventajas que ofrece para optimizar el tiempo de las consultas. Next.js 15 ofrece funcionalidades avanzadas en el App Router, como Server Components y Server Actions que fueron aprovechadas para mejorar el rendimiento y la experiencia de usuario.

3.2. API

Este componente constituye la capa de servicios que comunica el frontend con los distintos motores de búsqueda y recomendación. Incluye endpoints para listar productos (recibiendo parámetros de filtros y búsqueda), obtener información detallada de productos específicos, gestionar conversaciones con el agente AI, y generar recomendaciones de productos.

Se implementó utilizando FastAPI [3], un framework moderno de Python que permite un desarrollo eficiente y robusto. La decisión se basó en su sencillez y la simple integración con LangGraph para la implementación del módulo Agent. FastAPI ofrece validación automática de datos mediante Pydantic [4], tipado estático que reduce errores en tiempo de desarrollo, y documentación automática con OpenAPI que sirvió como recurso valioso para la paralelización del desarrollo API-Frontend.

3.3. Agent

El módulo Agent implementa un asistente AI conversacional que interactúa con los usuarios para entender sus preferencias y necesidades, realizar búsquedas de productos, y presentar resultados de manera contextual y útil.

Se implementó utilizando LangGraph [5], un framework para construir aplicaciones de AI con flujos de trabajo complejos y estados persistentes. La elección se basó en su capacidad para manejar conversaciones multi-turno y mantener el contexto del usuario a lo largo de la interacción. El agente implementa un grafo de estados que incluye nodos para la recolección de preferencias, búsqueda de productos y presentación de resultados, utilizando OpenAI GPT-4 [6] para el procesamiento del lenguaje natural y la generación de respuestas contextuales.

3.4. Qdrant Client

El Qdrant Client es el componente responsable de la gestión de la base de datos vectorial y la implementación de búsquedas híbridas que combinan múltiples estrategias para maximizar la relevancia de los resultados.

Se decidió utilizar Qdrant [7] como base de datos vectorial por su funcionalidad avanzada para la búsqueda híbrida de productos. Qdrant ofrece capacidades únicas

que fueron aprovechadas en el sistema: búsqueda híbrida que combina embeddings densos (para capturar significado semántico) y esparsos (para coincidencias textuales precisas), modelos de embeddings integrados utilizando fastembed [11] con modelos optimizados, filtrado avanzado que permite filtrar resultados por categorías, precios y otros atributos, y reranking para mejorar la calidad de los resultados.

3.5. CLI

La interfaz de línea de comandos (CLI) proporciona una herramienta de administración y desarrollo que facilita la carga de datos, el testeo de funcionalidades, y la gestión del sistema.

Se implementó utilizando Typer [9], una biblioteca moderna de Python para crear aplicaciones de línea de comandos. La elección se basó en su simplicidad, integración con FastAPI (ambos creados por el mismo autor), y su capacidad para generar documentación automática de comandos. La CLI resultó muy útil tanto para la carga de datos como para el testeo previo al desarrollo de los endpoints API, proporcionando funcionalidades como gestión de colecciones, carga de datos, búsqueda de productos, administración de productos y testeo de conexión. Utiliza Rich [10] para proporcionar una interfaz visual atractiva con tablas formateadas, barras de progreso, y mensajes de estado coloridos.

3.6. Dataset

Para la información de productos, se utilizó el conjunto de datos de Amazon disponible en Kaggle [8]. Este conjunto de datos ofrece una amplia variedad de atributos por producto, incluyendo:

- Título y descripción detallada
- Categorías y subcategorías
- Precio y disponibilidad
- Valoraciones y número de reseñas
- Imágenes de productos

- Especificaciones técnicas

La riqueza de estos atributos permitió implementar las modalidades de búsqueda previamente mencionadas, así como proporcionar información detallada para el agente AI y el sistema de recomendaciones.

4. Search

Esta sección describe la implementación técnica del sistema de búsqueda híbrida, desde el procesamiento inicial de datos hasta la ejecución de consultas complejas con filtros y paginación.

4.1. Procesamiento de Datos Inicial

El sistema implementa un pipeline robusto de procesamiento de datos que transforma el conjunto de datos original de Amazon en un formato optimizado para búsqueda vectorial. El dataset inicial contiene aproximadamente 550,000 productos, muchos de los cuales presentaban inconsistencias en los datos, imágenes no funcionales o información que requería normalización y limpieza antes de poder ser utilizada efectivamente.

4.1.1. Limpieza y Normalización

El proceso de limpieza se ejecuta a través de la función `clean_data` en el módulo `utils.py`, implementando las siguientes transformaciones:

- **Eliminación de valores nulos**: Se descartan productos sin campos esenciales (id, imagen, nombre, categoría, precios)
- **Filtrado de datos inconsistentes**: Se eliminan registros con valoraciones inválidas como `Get`, `FREE` o que contengan símbolos no numéricos
- **Normalización de valoraciones**: Conversión de conteos de ratings de formato string con comas a enteros
- **Conversión monetaria**: Transformación automática de precios en rupias indias (INR) a dólares estadounidenses usando tasa de cambio fija
- **Validación de URLs de imágenes**: Verificación concurrente de la accesibilidad de las imágenes de productos

4.2. Base de Datos Vectorial: Qdrant

La elección de Qdrant como base de datos vectorial se fundamenta en sus capacidades avanzadas para búsqueda híbrida y su arquitectura optimizada para aplicaciones de recuperación de información.

4.2.1. Justificación Técnica

Qdrant ofrece ventajas específicas que fueron determinantes para la implementación:

- **Búsqueda híbrida nativa**: Soporte integrado para combinar embeddings densos y esparsos en una sola consulta
- **Modelos de embeddings integrados**: Utilización de FastEmbed para generar representaciones vectoriales sin dependencias externas
- **Filtrado avanzado**: Capacidad de aplicar filtros complejos por categorías, rangos de precios y otros atributos sin impacto significativo en el rendimiento
- **Fácil de usar**: Interfaz intuitiva y documentación clara que facilita la implementación y mantenimiento.

4.2.2. Configuración de la Colección

La colección se configura con parámetros específicos para optimizar el rendimiento de búsqueda:

- **Vectores densos**: Dimensión 384 con distancia coseno para capturar similitud semántica
- **Vectores esparsos**: Implementación BM25 para coincidencias textuales exactas
- **Índices de payload**: Indexación automática de campos categóricos para filtrado eficiente

4.3. Modelos de Embeddings

El sistema implementa una arquitectura dual de embeddings que combina representaciones densas y esparsas para maximizar la calidad de los resultados de búsqueda.

4.3.1. Embeddings Densos: Sentence-Transformers

Los embeddings densos utilizan el modelo `sentence-transformers/all-MiniLM-L6-v2`, seleccionado por su balance óptimo entre calidad y eficiencia:

- **Dimensionalidad**: 384 dimensiones que capturan representaciones semánticas ricas con un balance entre precisión, eficiencia y tamaño de los vectores.
- **Entrenamiento**: Pre-entrenado en grandes corpus multilingües.
- **Ventajas**: Excelente para capturar similitudes conceptuales, sinónimos y relaciones semánticas.

El proceso de generación de embeddings densos procesa tanto el título como la descripción del producto, creando una representación vectorial que captura el significado semántico completo del artículo.

4.3.2. Embeddings Esparsos: BM25

Los embeddings esparsos implementan el algoritmo BM25 a través del modelo `Qdrant/bm25`, creando representaciones vectoriales donde cada dimensión corresponde a un token específico del diccionario de la colección.

4.3.3. Estructura del Vector Esparso

Un embedding esparso se representa como un conjunto de pares (índice, valor) donde:

- **Índice**: Posición del token en el diccionario global de Qdrant para la colección
- **Valor**: Score BM25 que combina la frecuencia del término (TF) con la frecuencia inversa de documento (IDF)
- **Esparsidad**: Solo se almacenan dimensiones con valores no-cero, optimizando el espacio de almacenamiento

4.3.4. Cálculo de Relevancia

El score BM25 para cada token se calcula mediante la fórmula:

- **Componente TF**: Frecuencia del término normalizada por la longitud del documento usando parámetros k1=1.2 y b=0.75
- **Componente IDF**: Calculado a nivel de colección basado en la frecuencia del término en todos los documentos
- **Actualización dinámica**: Los valores IDF se recalculan automáticamente cuando se añaden nuevos productos a la colección

Esta implementación permite:

- **Coincidencias exactas**: Identificación precisa de términos específicos en títulos y descripciones
- **Relevancia estadística**: Ponderación basada en la rareza del término en la colección completa
- **Robustez**: Resistencia a variaciones en la formulación de consultas
- **Eficiencia**: Almacenamiento optimizado que solo mantiene tokens relevantes

4.4. Búsqueda Híbrida

La implementación de búsqueda híbrida constituye el núcleo del sistema, combinando las fortalezas de ambos tipos de embeddings para proporcionar resultados superiores.

4.4.1. Principio de Funcionamiento

La búsqueda híbrida opera ejecutando simultáneamente dos consultas independientes sobre la misma colección de productos:

- **Consulta densa**: Utiliza el embedding denso de la query para encontrar productos semánticamente similares mediante búsqueda por similitud coseno

- **Consulta esparsa**: Aplica el embedding esparso BM25 de la query para identificar productos con coincidencias textuales exactas
- **Ejecución paralela**: Ambas consultas se procesan simultáneamente en Qdrant, optimizando el tiempo de respuesta
- **Aplicación de filtros**: Los filtros de categoría y precio se aplican de manera idéntica a ambas consultas

4.4.2. Ventajas de la Aproximación Dual

Esta estrategia dual permite capturar diferentes aspectos de la relevancia:

- **Cobertura semántica**: Los embeddings densos identifican productos conceptualmente relacionados aunque no compartan términos exactos
- **Precisión léxica**: Los embeddings esparsos garantizan que productos con términos clave específicos aparezcan en los resultados
- **Compensación mutua**: Cuando una aproximación falla, la otra puede proporcionar resultados relevantes
- **Robustez ante consultas**: El sistema funciona eficazmente tanto para búsquedas conceptuales como específicas

4.4.3. Algoritmo de Fusión

El sistema implementa una estrategia de fusión que opera en dos niveles:

- **Nivel de puntuación**: Combinación ponderada de scores densos y esparsos usando Reciprocal Rank Fusion (RRF)
- **Normalización**: Ajuste de escalas entre diferentes tipos de scores para comparación equitativa

4.5. Sistema de Filtrado

El sistema implementa filtros que se aplican tanto a las consultas densas como esparsas durante la búsqueda híbrida.

4.5.1. Filtros Disponibles

- **Categoría principal**: Filtrado por categorías como Electronics, Clothing, Home, etc.
- **Subcategoría**: Filtrado de segundo nivel (requiere categoría principal definida)
- **Rango de precios**: Filtros por precio mínimo y/o máximo en dólares

4.5.2. Implementación

Los filtros utilizan los índices automáticos de Qdrant sobre los campos de payload, aplicándose mediante operadores lógicos AND para combinar múltiples condiciones de filtrado.

4.6. Paginación

La implementación de paginación está optimizada para mantener rendimiento consistente independientemente del tamaño del conjunto de resultados.

4.6.1. Implementación

El sistema utiliza paginación basada en offset con optimizaciones específicas:

- **Offset y limit**: Parámetros estándar para control granular de resultados
- **Validación de parámetros**: Verificación de valores positivos y rangos válidos
- **Metadatos de paginación**: Información adicional sobre total de resultados y páginas disponibles

4.7. Interfaz de Línea de Comandos

La CLI proporciona herramientas para gestión de datos y testing del sistema, implementada con Typer y Rich para hacer uso del sistema de búsqueda desde la terminal.

4.7.1. Comandos Principales

- **create-collection**: Creación de colecciones con configuración automática
- **load-products**: Carga masiva de productos desde CSV
- **search-products**: Búsqueda híbrida con filtros y paginación
- **delete-collection**: Eliminación de colecciones
- **test-connection**: Verificación de conectividad con Qdrant

Esta implementación integral del sistema de búsqueda proporciona una base sólida para operaciones de recuperación de información eficientes y escalables, combinando técnicas modernas de NLP con optimizaciones específicas para comercio electrónico.

5. Agent

Esta sección describe la implementación técnica del agente conversacional de IA, desde la arquitectura del grafo de estados hasta la gestión de preferencias y presentación de productos.

5.1. Arquitectura del Grafo de Estados

El agente conversacional está implementado utilizando LangGraph, que proporciona una arquitectura basada en grafos de estados para manejar conversaciones complejas con múltiples etapas de procesamiento.

5.1.1. Estado del Grafo

El sistema mantiene un estado global (`GraphState`) que persiste a lo largo de toda la conversación:

- **Historial de conversación**: Lista de mensajes intercambiados entre usuario y asistente
- **Preferencias del usuario**: Estructura que almacena criterios de búsqueda recolectados dinámicamente
- **Productos encontrados**: Lista de productos resultantes de la búsqueda híbrida

5.1.2. Nodos del Grafo

La arquitectura implementa tres nodos principales que procesan diferentes aspectos de la conversación:

- **collect_preferences**: Recolecta y refina las preferencias del usuario mediante diálogo natural
- **search_products**: Ejecuta búsquedas híbridas utilizando las preferencias recolectadas
- **present_products**: Genera presentaciones personalizadas de los productos encontrados

5.2. Recolección de Preferencias

El sistema implementa un mecanismo sofisticado para extraer preferencias del usuario a través de conversación natural, utilizando técnicas de procesamiento de lenguaje natural.

5.2.1. Estructura de Preferencias

Las preferencias del usuario se modelan mediante la clase `UserPreferences`:

- **Query**: Consulta textual que describe el tipo de producto deseado
- **Categoría principal**: Categoría de alto nivel (Electronics, Clothing, Home, etc.)
- **Rango de precios**: Límites mínimo y máximo en dólares estadounidenses
- **Atributos específicos**: Color y marca preferidos por el usuario

5.2.2. Validación de Suficiencia

El sistema implementa una función de validación (`has_sufficient_preferences`) que determina cuándo se han recolectado suficientes datos para ejecutar una búsqueda efectiva:

- **Campos mínimos**: Requiere al menos 3 campos completados de las 6 opciones disponibles
- **Campo obligatorio**: La query textual es siempre requerida para iniciar búsquedas
- **Validación dinámica**: Evalúa la completitud en cada iteración del diálogo

5.2.3. Prompts Especializados

La recolección utiliza prompts ingeniería específicamente diseñados para extraer información estructurada:

- **Manejo multiidioma**: Respuestas en el idioma del usuario, datos almacenados en inglés

- **Inferencia categórica**: Mapeo automático de descripciones a categorías válidas
- **Preservación de contexto**: Mantiene preferencias previamente recolectadas
- **Resolución de conflictos**: Solicita clarificación ante información contradictoria

5.2.4. Limitaciones del Sistema Multiidioma

A pesar de las especificaciones detalladas en los prompts, el sistema presenta limitaciones en el manejo multiidioma:

- **Inconsistencia en traducción**: La recolección de preferencias no siempre almacena los datos en inglés como se especifica en el prompt, lo que puede causar desajustes con la base de datos de productos
- **Limitaciones del modelo**: El modelo GPT-4.1-nano actual no siempre sigue consistentemente las instrucciones de traducción automática
- **Posible solución**: Un modelo más expresivo y con mayor capacidad de seguimiento de instrucciones podría resolver estos problemas de consistencia idiomática

5.3. Integración con Búsqueda

El agente utiliza el sistema de búsqueda híbrida como herramienta subyacente para obtener productos relevantes basados en las preferencias recolectadas.

5.3.1. Construcción de Consultas

El nodo `search_products` construye consultas optimizadas combinando múltiples campos de preferencias:

- **Query base**: Utiliza la consulta textual principal del usuario
- **Enriquecimiento**: Añade color y marca cuando están disponibles

- **Filtros aplicados**: Categoría principal y rangos de precio se aplican como filtros
- **Límite de resultados**: Configurado para presentar 3 productos por defecto

5.3.2. Manejo de Errores

El sistema implementa estrategias robustas para manejar casos donde la búsqueda no produce resultados:

- **Validación previa**: Verifica que existe una query antes de ejecutar búsquedas
- **Respuestas alternativas**: Proporciona mensajes informativos cuando no hay productos
- **Recuperación graceful**: Mantiene el flujo conversacional ante errores técnicos

5.4. Presentación de Productos

La presentación de productos utiliza un sistema de generación de texto especializado que adapta el contenido según las preferencias del usuario y características de los productos encontrados.

5.4.1. Generación Contextual

El nodo `present_products` genera descripciones personalizadas utilizando:

- **Contexto de preferencias**: Incluye los criterios específicos del usuario
- **Datos de productos**: Información detallada de cada artículo encontrado
- **Orden preservado**: Mantiene la secuencia de relevancia de la búsqueda
- **Alineación con necesidades**: Explica cómo cada producto satisface los criterios

5.4.2. Formato de Presentación

Las presentaciones siguen una estructura consistente que optimiza la experiencia del usuario:

- **Introducción general**: Mensaje amigable que resume las preferencias identificadas
- **Descripciones individuales**: Análisis detallado de cada producto con características clave
- **Información de precios**: Precios actuales y descuentos cuando aplican
- **Valoraciones sociales**: Ratings y número de reseñas para validación social

5.5. Configuración y Parámetros

El sistema utiliza un conjunto de parámetros configurables que optimizan el comportamiento del agente para diferentes escenarios de uso.

5.5.1. Configuración de OpenAI

- **Modelo**: GPT-4.1-nano para balance entre calidad y eficiencia
- **Temperatura**: 0.0 para respuestas determinísticas y consistentes
- **Structured outputs**: Utiliza Pydantic para validación automática de respuestas

5.5.2. Parámetros de Conversación

- **Contexto de mensajes**: Mantiene los últimos 10 mensajes para contexto relevante
- **Límite de recursión**: Máximo 10 iteraciones para prevenir bucles infinitos
- **Gestión de hilos**: Identificador único para mantener contexto conversacional

5.6. Continuidad Conversacional y Refinamiento

El sistema mantiene el contexto completo de la conversación, permitiendo refinamientos iterativos de búsquedas basados en interacciones previas.

5.6.1. Mantenimiento de Contexto

Una vez que se ha realizado una búsqueda inicial y se han presentado productos al usuario, el sistema permite:

- **Refinamiento de preferencias**: El usuario puede modificar criterios específicos (como rango de precios) manteniendo el resto de preferencias previamente establecidas
- **Búsquedas iterativas**: Nuevas consultas utilizan el contexto previo para generar resultados más refinados
- **Preservación selectiva**: Solo se actualizan los campos de preferencias que el usuario modifica explícitamente
- **Historial persistente**: El sistema mantiene el registro completo de la conversación para referencias futuras

5.6.2. Ejemplo de Refinamiento

El flujo típico de refinamiento funciona de la siguiente manera:

- **Búsqueda inicial**: Usuario solicita .ªuriculares inalámbricosçon presupuesto de $50-100
- **Presentación de resultados**: Sistema muestra 3 productos relevantes
- **Refinamiento del usuario**: "Muéstrame opciones más baratas, hasta $50"
- **Nueva búsqueda contextual**: Sistema mantiene .ªuriculares inalámbricos"pero actualiza price_max a $50
- **Resultados refinados**: Nuevos productos que satisfacen los criterios actualizados

Esta capacidad de refinamiento iterativo mejora significativamente la experiencia del usuario al permitir ajustes precisos sin necesidad de reiniciar la conversación completa.

5.7. Flujo de Ejecución

El agente implementa un flujo de ejecución determinístico que garantiza una experiencia conversacional coherente y eficiente.

5.7.1. Enrutamiento Condicional

La función `route_after_collection` implementa lógica de decisión que determina el siguiente paso basado en el estado de las preferencias:

- **Preferencias insuficientes**: Continúa en modo recolección para obtener más información
- **Preferencias suficientes**: Procede a ejecutar búsqueda de productos
- **Evaluación dinámica**: Reevalúa en cada iteración del diálogo

5.7.2. Manejo de Excepciones

El sistema implementa captura robusta de errores que mantiene la funcionalidad ante fallos:

- **Errores de API**: Manejo graceful de fallos en llamadas a OpenAI
- **Errores de búsqueda**: Recuperación ante problemas con el motor de búsqueda
- **Mensajes informativos**: Comunicación clara de problemas al usuario
- **Continuidad conversacional**: Mantiene el contexto ante errores técnicos

Esta implementación integral del agente conversacional proporciona una experiencia de usuario natural y eficiente, combinando técnicas avanzadas de procesamiento de lenguaje natural con integración robusta al sistema de búsqueda híbrida.

6. Recommendation

El sistema de recomendación de Amazon Copilot tiene como objetivo asistir al usuario en la selección de productos complementarios o alternativos a los que ya ha añadido a su carrito. Aprovechando técnicas avanzadas de inteligencia artificial y búsqueda híbrida, el sistema analiza el contenido actual del carrito y genera sugerencias personalizadas en tiempo real. A continuación se describe el flujo general y el funcionamiento interno del módulo de recomendación.

6.1. Estructura (flujo)

El flujo puede resumirse en cuatro pasos sencillos:

1. **Envío del carrito.** El *frontend* hace una petición a un *endpoint* y envía la lista de productos que se encuentran actualmente en el carrito al servicio de recomendaciones.

2. **Creación de ideas.** A partir de esos productos se generan palabras clave que describen posibles artículos complementarios. Esto se hace pasándole la lista de productos a OpenAI, que devuelve una lista de artículos sugeridos.

3. **Búsqueda de productos.** Con esta lista se realiza una consulta a Qdrant para obtener, para cada sugerencia, el artículo más similar disponible en nuestro sistema.

4. **Respuesta.** El servicio devuelve una lista de los productos más similares a los sugeridos por OpenAI que existen en nuestro catálogo.

6.2. Funcionamiento

El sistema de recomendación analiza los productos seleccionados en el carrito para sugerir artículos complementarios o alternativos. Opera automáticamente al visualizar el carrito, generando recomendaciones basadas en productos frecuentemente adquiridos conjuntamente y alternativas similares. Si se elimina o agrega un producto al carrito esta lista de sugerencias se refresca.

7. UX/UI

7.1. Flujo de Interacción del Usuario

El flujo de interacción de Amazon Copilot está diseñado para proporcionar una experiencia de usuario intuitiva y eficiente. La aplicación presenta tres páginas principales que guían al usuario a través de su jornada de compra.

7.1.1. Página Principal - Catálogo de Productos

La página principal (*Home*) sirve como punto de entrada a la aplicación, presentando un catálogo completo de productos con funcionalidades avanzadas de búsqueda y filtrado.

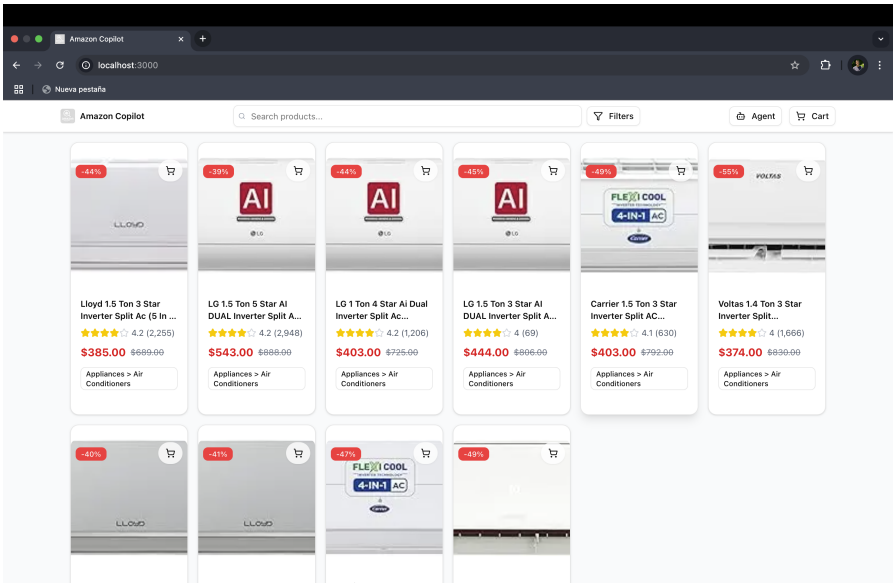

Figura 2: Página principal - Catálogo de Productos

La interfaz incluye:

- **Navbar superior**: Contiene el logo de la aplicación, barra de búsqueda y botones de navegación hacia el carrito y el asistente AI
- **Filtros dinámicos**: Panel lateral con categorías obtenidas dinámicamente del backend

- **Grid de productos**: Visualización en formato de tarjetas con información esencial (imagen, título, precio, descuento)
- **Paginación**: Controles para navegar entre páginas de resultados

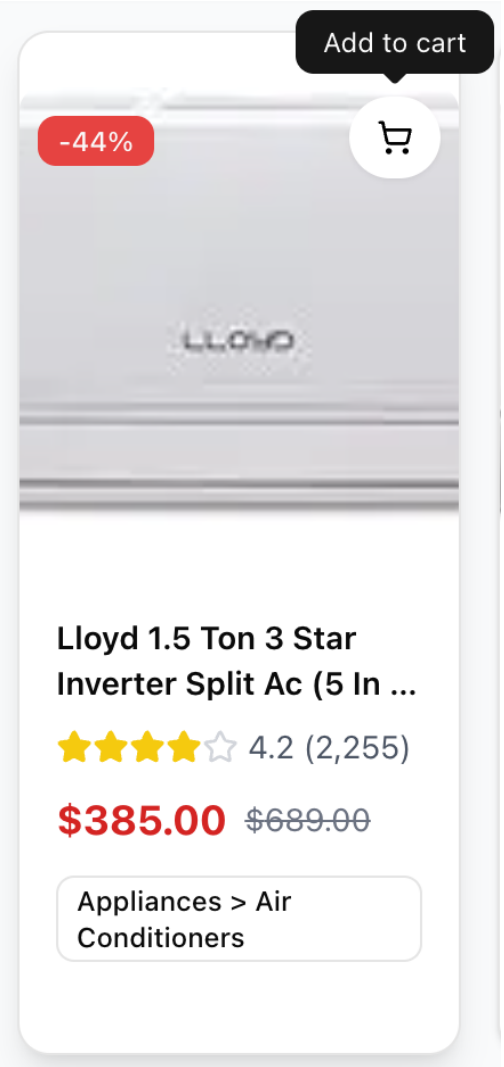

Figura 3: Detalle de un producto en la página principal

7.1.2. Página del Carrito

La página del carrito (*Cart*) proporciona una vista consolidada de los productos seleccionados y funcionalidades adicionales de recomendación.

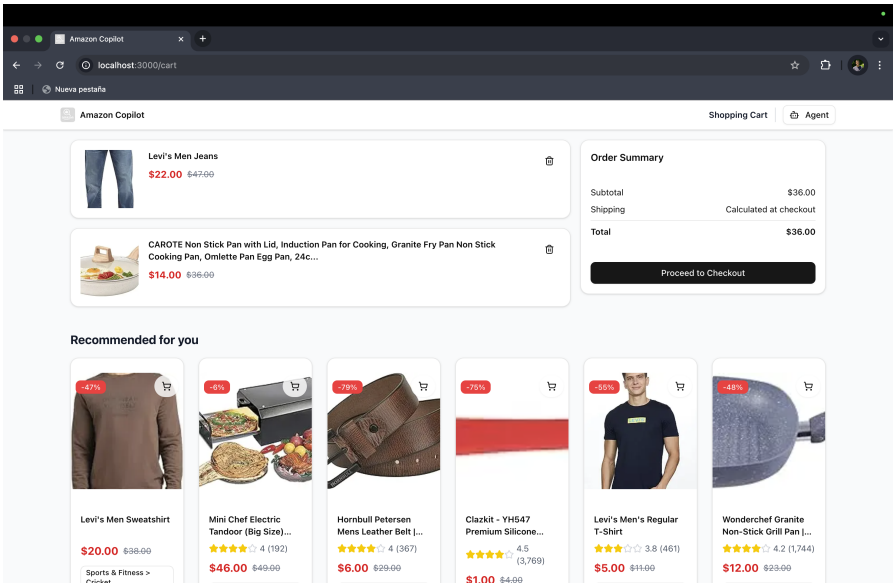

Figura 4: Página del carrito

Esta página se estructura en tres secciones principales:

- **Lista de productos**: Muestra todos los items agregados al carrito con opciones de eliminación
- **Resumen de orden**: Panel lateral con el total de la compra y botón de checkout
- **Recomendaciones**: Sección inferior que sugiere productos relacionados basados en el contenido del carrito

7.1.3. Página del Asistente AI

La página del asistente (*Agent*) implementa una interfaz de chat conversacional para asistir al usuario en sus consultas sobre productos.

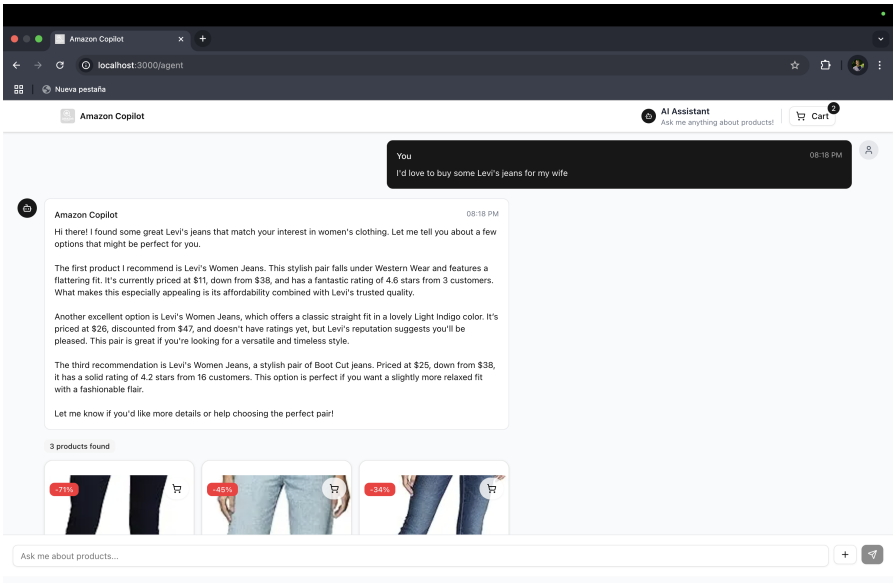

Figura 5: Página del asistente AI

Características principales:

- **Interfaz de chat**: Diseño limpio con burbujas de mensaje diferenciadas por rol
- **Indicador de carga**: Feedback visual durante el procesamiento de mensajes
- **Productos sugeridos**: Visualización de productos recomendados por el asistente
- **Navegación contextual**: Enlaces directos a productos mencionados en la conversación

7.2. Búsqueda y Filtrado Dinámico

El sistema de búsqueda y filtrado implementa una arquitectura que optimiza tanto la experiencia del usuario como el rendimiento del backend.

7.2.1. Filtros Dinámicos

Los filtros de categorías se obtienen dinámicamente a través del endpoint de categorías, pero con una característica importante: las categorías disponibles se

calculan en función de todos los productos existentes en la base de datos, no solo de los productos mostrados en la página actual.

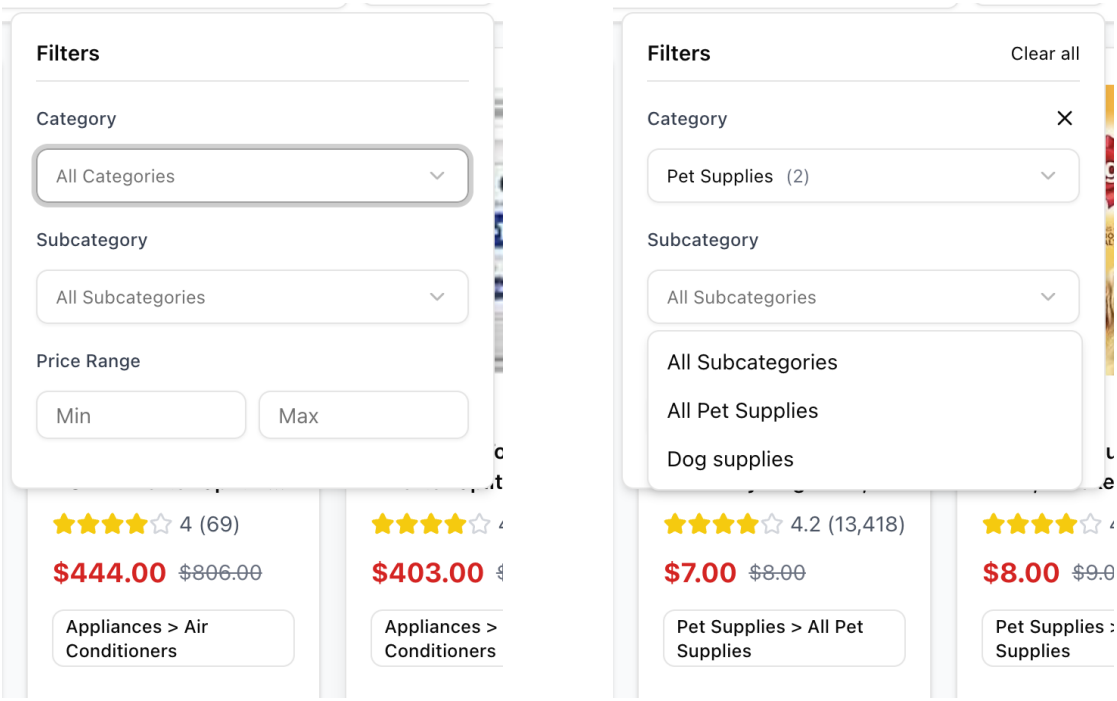

Figura 6: Panel de filtros

Esta implementación ofrece varias ventajas fundamentales:

- **Filtrado en Backend**: Tanto los filtros como la paginación se procesan completamente en el servidor, garantizando que cada consulta se ejecute sobre la totalidad del catálogo de productos disponible
- **Consistencia**: Los filtros siempre reflejan la totalidad del catálogo disponible, independientemente de la página actual o los resultados mostrados
- **Precisión**: Evita situaciones donde un filtro no devuelve resultados debido a limitaciones de la página actual
- **Eficiencia**: El filtrado y paginación se realizan a nivel de base de datos, optimizando las consultas y reduciendo la transferencia de datos innecesaria
- **Escalabilidad**: La arquitectura permite manejar catálogos de gran tamaño sin impactar el rendimiento del frontend

El flujo de procesamiento garantiza que cada búsqueda, filtro o cambio de página resulte en una nueva consulta al backend que opera sobre el conjunto completo de productos, asegurando resultados precisos y consistentes en todo momento.

7.2.2. Query Parameters y Reactividad

La aplicación utiliza query parameters para triggerear nuevas consultas, implementando un sistema reactivo que actualiza automáticamente los resultados cuando el usuario modifica los filtros o la búsqueda.

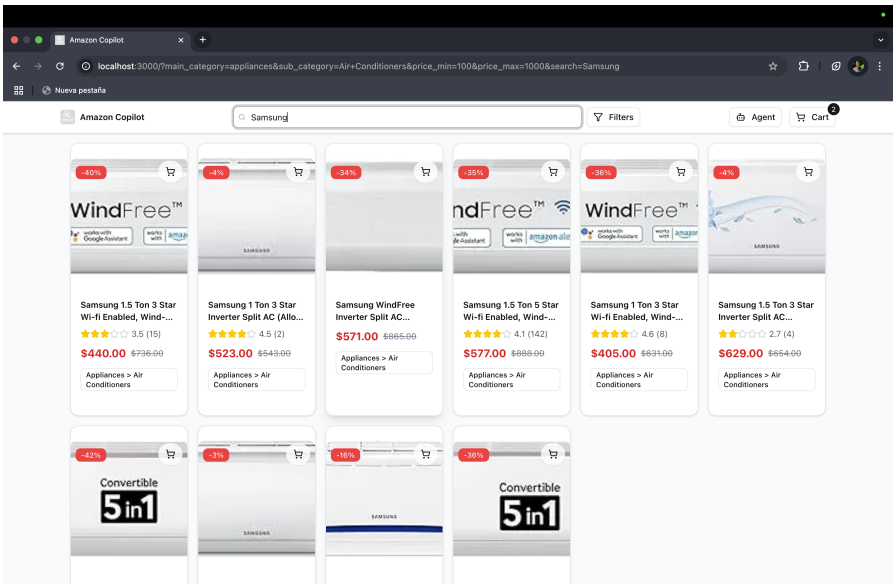

Figura 7: URL con query parameters

El flujo de actualización funciona de la siguiente manera:

1. El usuario modifica un filtro o realiza una búsqueda
2. Los query parameters se actualizan en la URL
3. Next.js detecta el cambio y re-renderiza la página
4. Se ejecuta una nueva consulta al backend con los parámetros actualizados
5. Los resultados se muestran al usuario

Esta implementación ofrece ventajas adicionales:

- **URLs compartibles**: Los usuarios pueden compartir resultados específicos simplemente copiando el enlace, ya que todos los parámetros de búsqueda y filtrado están codificados en la URL
- **Reproducibilidad**: Cualquier usuario puede reproducir exactamente los mismos resultados accediendo al enlace compartido
- **Bookmarking**: Los usuarios pueden guardar búsquedas específicas como favoritos en su navegador

7.3. Arquitectura de Server Components y Suspense

La aplicación aprovecha las capacidades avanzadas de Next.js 15, específicamente los Server Components y Suspense, para optimizar el rendimiento y la experiencia de usuario.

7.3.1. Server Components

Los Server Components permiten que la ejecución de queries comience desde el momento en que el navegador realiza la consulta al servidor web, no después de que se cargue el JavaScript en el cliente.

Beneficios de esta arquitectura:

- **Inicio temprano**: Las consultas se ejecutan en paralelo con la generación del HTML
- **Mejor SEO**: El contenido se renderiza en el servidor, mejorando la indexación
- **Menor JavaScript**: Reduce la cantidad de código que debe descargarse al cliente
- **Mejor rendimiento inicial**: La página se muestra más rápido al usuario

Además, la aplicación aprovecha los **Server Actions** de Next.js para operaciones que requieren interacción del servidor, como el envío de mensajes al asistente AI y la generación de recomendaciones de productos. Los Server Actions permiten ejecutar código del servidor directamente desde componentes del cliente, proporcionando una experiencia fluida sin necesidad de crear endpoints API separados.

7.3.2. Suspense y Estados de Carga

El componente Suspense se utiliza para proporcionar una experiencia de usuario fluida durante la carga de datos, implementando loading states que se muestran mientras se stremea el HTML con los datos.

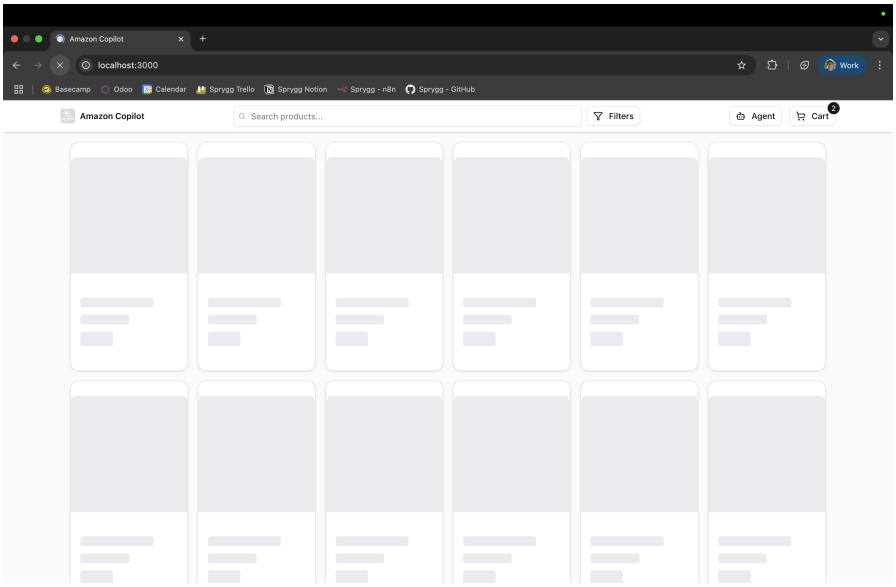

Figura 8: Estado de carga con Suspense

La implementación incluye:

- **Skeleton loading**: Placeholders animados que simulan la estructura del contenido
- **Streaming progresivo**: Los datos se envían al cliente tan pronto como están disponibles
- **Transiciones suaves**: Los estados de carga se reemplazan sin interrupciones visuales

7.4. Gestión de Estado con Contextos

Para el carrito y el chatbot, se implementó un sistema de gestión de estado basado en contextos de React, combinado con almacenamiento local en el navegador.

7.4.1. Contexto del Carrito

El contexto del carrito maneja el estado de los productos seleccionados, almacenándolos en localStorage para persistencia entre sesiones.

Características principales:

- **Persistencia local**: Los productos se mantienen entre recargas de página
- **Sincronización automática**: Cambios en el carrito se reflejan inmediatamente en toda la aplicación
- **Recomendaciones dinámicas**: Los productos del carrito se utilizan para generar recomendaciones personalizadas

7.4.2. Contexto de Conversación

El contexto de conversación gestiona el estado del chat con el asistente AI, implementando una estrategia híbrida de almacenamiento.

La implementación distingue entre dos tipos de datos:

- **Mensajes**: Se almacenan en localStorage únicamente para su visualización en la interfaz
- **Identificador de conversación**: Se mantiene para preservar el contexto de la conversación en el backend

Esta separación permite:

- **Contexto persistente**: El asistente mantiene memoria de mensajes anteriores
- **Experiencia fluida**: Los mensajes se muestran inmediatamente sin necesidad de recargar
- **Eficiencia de almacenamiento**: No se requiere una base de datos adicional para el estado

Esta arquitectura es especialmente adecuada para prototipos y aplicaciones de demostración, donde la simplicidad y velocidad de desarrollo son prioritarias sobre la persistencia a largo plazo de datos de estado.

8. Expansión

El presente capítulo traza una hoja de ruta para la evolución de Amazon Copilot, agrupando las iniciativas en tres grandes ejes: experiencia de usuario, calidad de las recomendaciones y robustez de la plataforma. Todas las ideas aquí expuestas son complementarias entre sí y pueden abordarse de forma incremental.

8.1. Streaming de respuestas

Implementar *streaming* permitiría que el asistente enviase la respuesta de forma incremental—token a token—mediante *Server-Sent Events* o *WebSockets*. Esto traería dos ventajas principales:

- **Percepción de inmediatez.** El usuario ve cómo el mensaje se «escribe» en pantalla, reduciendo la sensación de espera en consultas largas.
- **Interrupción inteligente.** Al recibir tokens en tiempo real, el cliente puede ofrecer al usuario la opción de cancelar, reformular o profundizar sin tener que esperar al final de la generación.

8.2. Base de datos de telemetría y *feedback*

Registrar la interacción de los usuarios en una base de datos operacional abre la puerta a un ciclo virtuoso de mejora continua:

- **Patrones de comportamiento.** Analizar clics, búsquedas y compras para detectar tendencias estacionales o hábitos de compra.
- **Re-entrenamiento de modelos.** Utilizar los datos recogidos para ajustar la ponderación entre embeddings densos y esparcidos, o para afinar los *prompts*.
- **Personalización.** Mantener perfiles de preferencia (marcas, rangos de precio, colores) y aplicarlos en futuras consultas o recomendaciones.
- **Métricas de producto.** Medir *CTR*, tasa de conversión y tiempo medio de respuesta para orientar decisiones de negocio.

8.3. Embeddings de imágenes y búsqueda multimodal

Extender el índice a vectores visuales dotaría al sistema de capacidades multimodales:

- **Búsqueda inversa.** El usuario podría subir una foto o URL y recibir productos similares en forma, color o estilo.
- **Recomendaciones estéticas.** Combinar señal visual y textual para sugerir artículos que «combinen» con el carrito actual (p. ej. sets de ropa).
- **Comparación rápida.** Mostrar al usuario variaciones visuales (otros colores, modelos o diseños) sin depender solo de descripciones textuales.

8.4. Sistemas de memoria a largo plazo

Implementar memoria persistente permitiría al sistema mantener contexto de usuario más allá de conversaciones individuales:

- **Perfiles de preferencias persistentes.** Mantener un registro histórico de categorías preferidas, marcas favoritas, rangos de precio habituales y patrones de compra del usuario.
- **Memoria semántica.** Almacenar conceptos y relaciones aprendidas durante interacciones previas para mejorar la comprensión contextual en futuras conversaciones.
- **Historial de búsquedas.** Registrar consultas exitosas y productos seleccionados para identificar patrones de comportamiento y preferencias implícitas.
- **Adaptación progresiva.** Ajustar automáticamente los algoritmos de recomendación basándose en el feedback acumulado y las interacciones históricas del usuario.

8.5. Análisis de tendencias y patrones estacionales

Incorporar inteligencia temporal permitiría al sistema anticipar y responder a cambios en la demanda:

- **Detección de tendencias emergentes.** Analizar patrones de búsqueda y compra para identificar productos en auge antes de que se conviertan en tendencias masivas.
- **Recomendaciones estacionales.** Ajustar automáticamente las sugerencias según la época del año, festividades o eventos especiales (Black Friday, Navidad, regreso a clases).
- **Predicción de demanda.** Utilizar datos históricos para anticipar picos de demanda en categorías específicas y optimizar la presentación de productos.
- **Personalización temporal.** Adaptar las recomendaciones considerando no solo las preferencias del usuario, sino también el contexto temporal y las tendencias actuales del mercado.

9. Conclusiones

Amazon Copilot implementa técnicas de recuperación de información y procesamiento de lenguaje natural para comercio electrónico. El sistema utiliza Next.js 15, FastAPI, LangGraph y Qdrant en una arquitectura modular con componentes separados.

Como base del sistema, la búsqueda híbrida combina embeddings densos y esparsos para búsquedas sobre 550,000 productos. Esta implementación en Qdrant permite filtrado por categorías y precios con rendimiento escalable.

Sobre esta base de búsqueda, el agente conversacional usa grafos de estados para mantener contexto en conversaciones multi-turno y recopilar preferencias de usuario. Complementariamente, el sistema de recomendación analiza el contenido del carrito para sugerir productos relacionados.

Para la presentación de estos servicios, la interfaz web utiliza Server Components y Suspense de Next.js 15 para optimizar rendimiento. Asimismo, implementa URLs compartibles y filtrado dinámico procesado en el backend.

Finalmente, el proyecto identifica oportunidades de expansión en streaming de respuestas, telemetría, búsqueda multimodal y sistemas de memoria persistente.

En conjunto, Amazon Copilot integra técnicas de recuperación de información IA y desarrollo web moderno en un sistema funcional para comercio electrónico inteligente.

10. Referencias

[1] Cursor. (2024). *Cursor: The AI-first code editor*. https://www.cursor.com

[2] Replit. (2024). *Replit: The collaborative browser based IDE*. https://replit.com

[3] FastAPI. (2024). *FastAPI: Framework web de alto rendimiento para APIs con Python*. https://fastapi.tiangolo.com

[4] Pydantic. (2024). *Pydantic: Validación de datos para Python*. https://docs.pydantic.dev/latest

[5] LangGraph. (2024). *LangGraph: Orchestration framework for building stateful, multi-actor applications with LLMs*. https://www.langchain.com/langgraph

[6] OpenAI. (2024). *OpenAI: Creating safe AI that benefits humanity*. https://openai.com

[7] Qdrant. (2024). *Qdrant: Vector Database*. https://qdrant.tech/documentation/search-precision/reranking-hybrid-search

[8] Parab, L. (2023). *Amazon Products Dataset*. Kaggle. https://www.kaggle.com/datasets/lokeshparab/amazon-products-dataset/data?select=Amazon-Products.csv

[9] Typer. (2024). *Typer: Library for building CLI applications*. https://typer.tiangolo.com

[10] Rich. (2024). *Rich: Rich text and beautiful formatting in the terminal*. https://rich.readthedocs.io

[11] FastEmbed. (2024). *FastEmbed: Fast, Lightweight Embedding Model*. https://github.com/qdrant/fastembed

[12] Next.js. (2024). *Next.js: The React Framework*. https://nextjs.org