Report 4: RAG, UI, and Web Analytics

SNumbers: u264332, u264443, u264202
Names: Levente Olivér Bódi, Riccardo Zamuner, Giada Izzo
Github repository: https://github.com/Levilevi01/Info_Retrieval/tree/main
Repository TAG: **IRWA-2025-part-4**

Introduction

In this final part of the project, we transformed our information retrieval algorithms into a fully functional web application. We implemented a User Interface (UI) using Flask, integrated a Retrieval-Augmented Generation (RAG) system to provide AI-powered summaries, and developed a Web Analytics dashboard to track user behavior (clicks, dwell time, and sessions). This report details our design choices, the improvements made to the RAG pipeline, and our analytics data model.

Part 1: User Interface

We utilized the Flask web framework to create a lightweight, responsive application. The architecture consists of three main views:

1. Search Page (/)

The entry point is a clean, minimalist search page containing a central search box. When a user enters a query (e.g., 'men jeans') and submits, the application triggers a GET request to the /search endpoint, passing the query as a URL parameter.

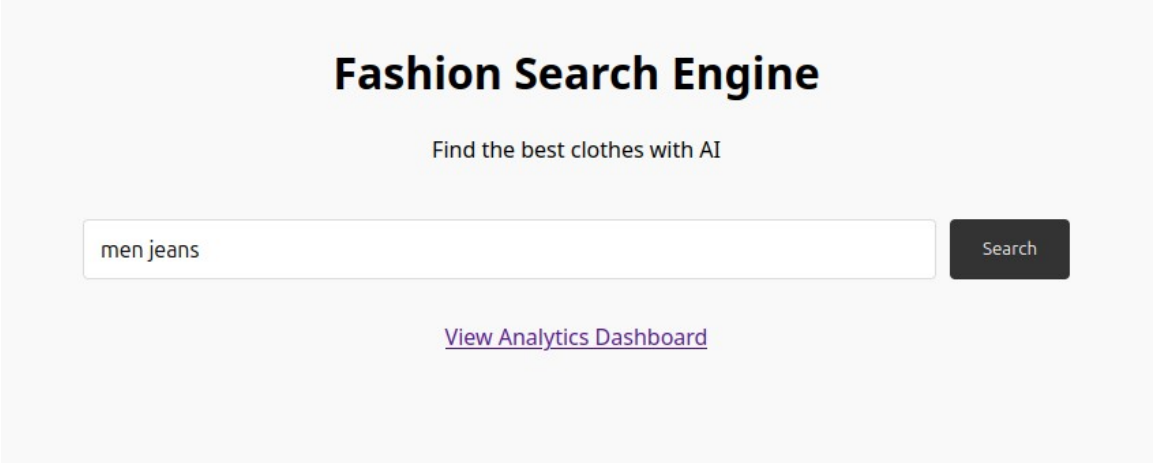

Figure 1: The minimalist Search Home Page.

2. Results Page (/search)

This page displays the retrieved documents ranked by our search engine (utilizing BM25 with custom boosting). Key features include:

- **RAG Summary:** A generated text block at the top providing a quick answer to the user's intent.
- **Result Cards:** Each product is displayed with its Title, Selling Price, Discount (highlighted in green), Rating, and a truncated Description.
- **Tracking:** Click events are tracked asynchronously when a user selects a product.

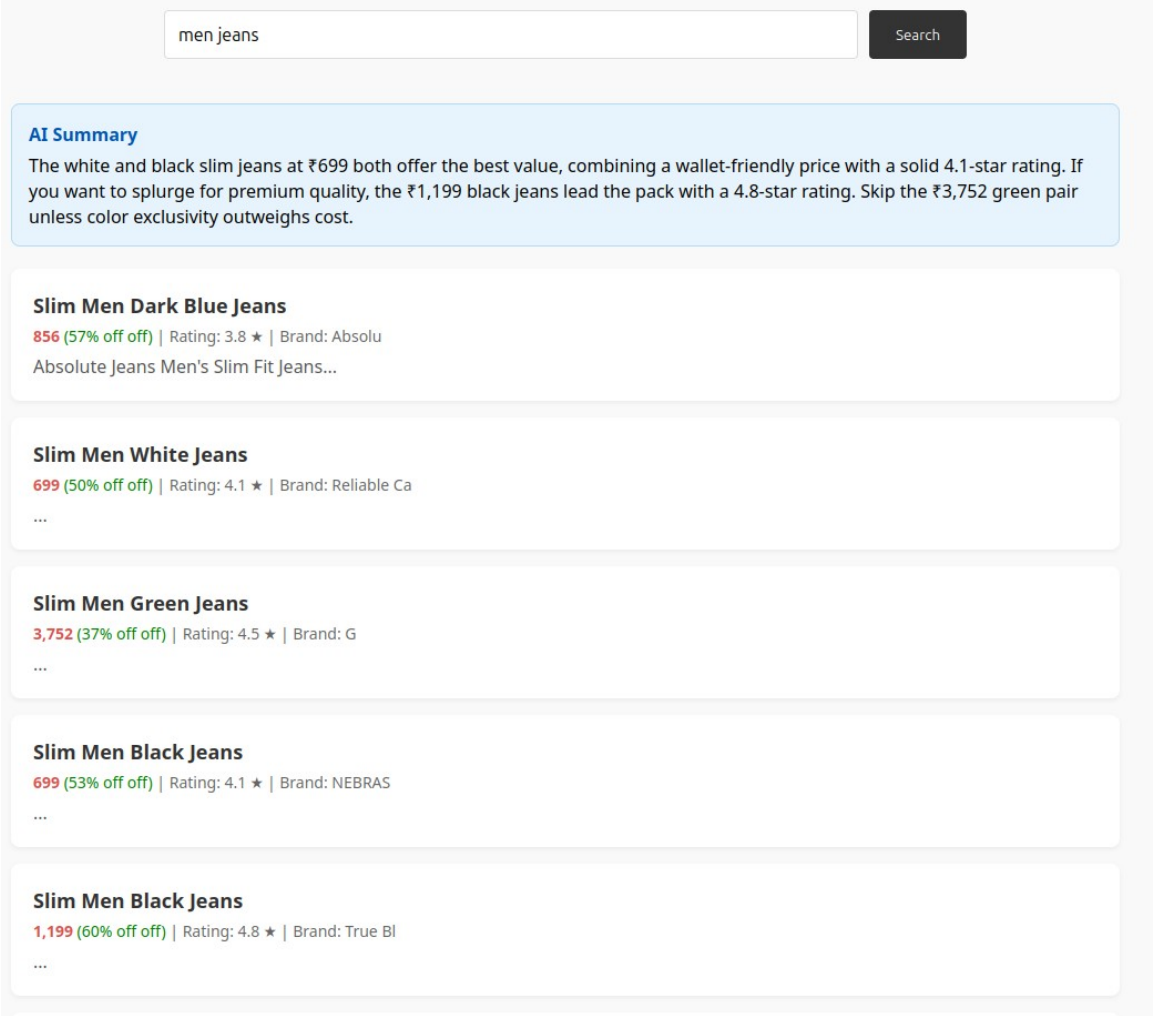

Figure 2: Search Results with AI Summary and Product Cards.

3. Product Details Page (/product/<id>)

This page presents the full metadata of the document. We implemented a 'Buy on Original Site' button that links to the external URL. Crucially, this page includes JavaScript logic to track 'Dwell Time' using the navigator.sendBeacon API, ensuring accurate engagement data is sent even when the user closes the tab.

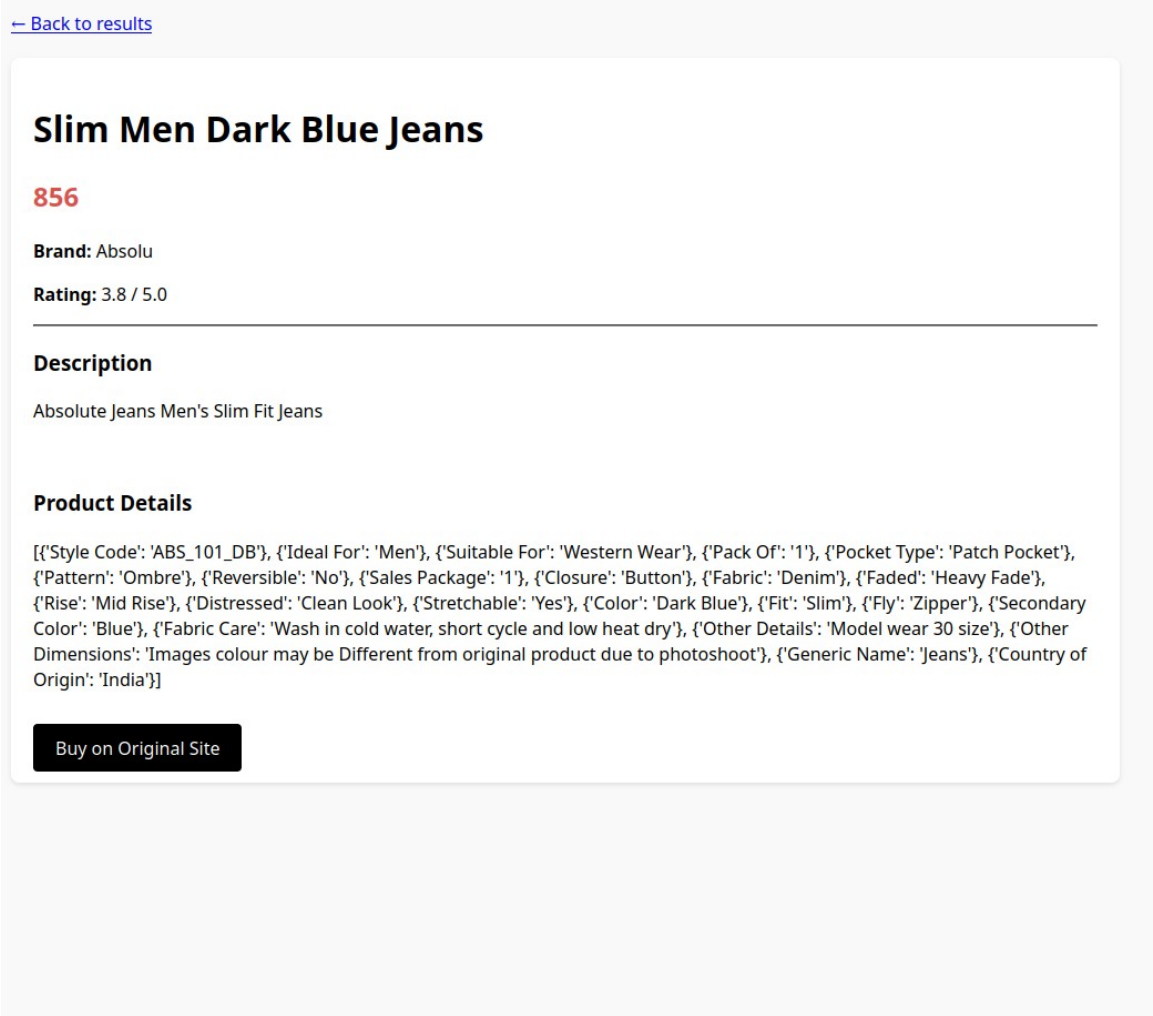

Figure 3: Product Details Page displaying full metadata.

Part 2: RAG Improvements

We implemented a RAG system using the Groq API (model: moonshotai/kimi-k2-instruct). Below is our analysis of the baseline versus our improved implementation.

Baseline Weaknesses

The baseline RAG implementation was generic. It simply fed the top results to the LLM without pruning, leading to context window overflows. It also lacked specific instructions, resulting in vague summaries that did not help the user make a purchasing decision.

Implemented Improvements

1. **Context Pruning (Top 5 Results):** We restricted the context passed to the LLM to only the top 5 retrieved documents. Pros: Reduces API latency and cost; prevents 'lost in the middle' phenomenon where LLMs ignore data in large contexts.
2. **Metadata Injection:** We explicitly formatted the context string to include 'Price', 'Rating', and 'Title' alongside the description. Pros: Allows the LLM to perform reasoning tasks, such as identifying the 'best value' or 'highest rated' item, rather than just summarizing text.
3. **Refined Prompt Engineering:** We adopted a persona-based prompt ('Expert Fashion Shopping Assistant') and imposed constraints ('provide a 3-sentence summary', 'highlight best value'). Pros: The output is now structured, concise, and actionable for the user.

Part 3: Web Analytics

We developed a custom analytics module to track user behavior without relying on external cookies.

1. Data Collection

We capture data at three distinct points:

- **Sessions:** Initialized on the home page, tracking User-Agent and IP address to identify unique visitors.
- **Searches:** Every query is logged with a timestamp and the number of results found.
- **Interactions (Clicks & Dwell):** We log the rank of the clicked document. Dwell time is calculated client-side (Exit Time - Entry Time) and sent to the server via a POST request to /track_dwell.

2. Data Model (Star Schema Simulation)

We implemented an in-memory 'AnalyticsStore' class acting as a star schema with three main lists:

Entity	Attributes
Session	session_id, user_agent, ip, timestamp
Search	search_id, session_id, query, num_results
Click	session_id, search_id, doc_uid, rank,

	dwell_time

3. Analytics Dashboard

The dashboard visualizes the collected data to provide actionable insights. We implemented two main graphs using Chart.js:

- **Top Queries (Bar Chart):** Visualizes the most frequent terms users search for. This helps in understanding user intent.
- **Activity Timeline (Line Chart):** plots 'Searches' vs. 'Clicks' over time. This metric is crucial for understanding the conversion funnel—if searches are high but clicks are low, the ranking algorithm may need adjustment.

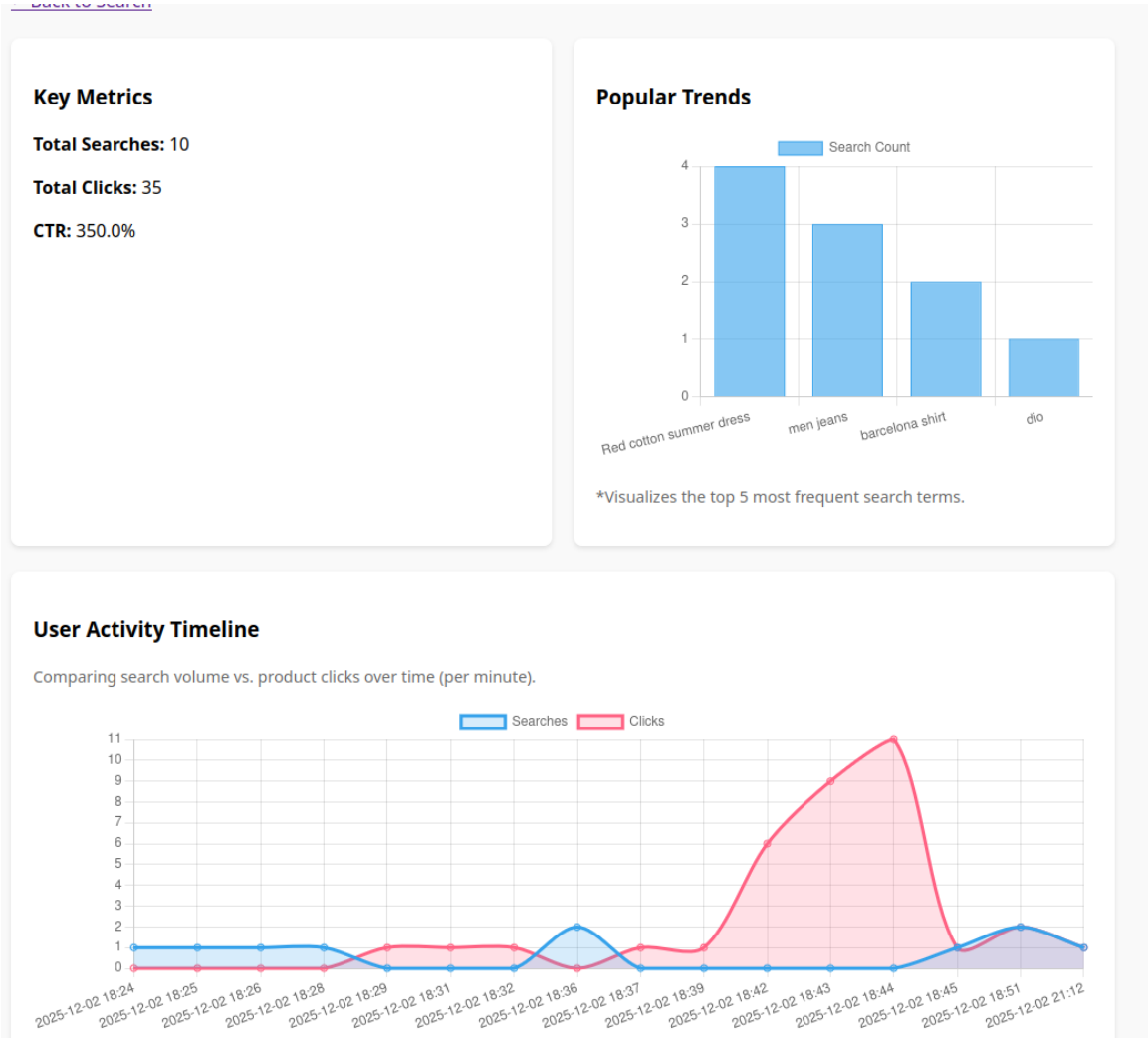

Figure 4: Analytics Dashboard showing Search trends and Activity timeline.