

E-Commerce Product Recommender

An intelligent product recommendation system that combines traditional recommendation algorithms with Large Language Model (LLM) powered explanations, featuring a Knowledge Graph-based RAG (Retrieval-Augmented Generation) architecture.

■ Objective

Build a hybrid recommendation system that not only suggests products based on user behavior and catalog analysis but also provides intelligent, contextual explanations for why each product is recommended using advanced LLM capabilities.

■ Key Features

- **Intelligent Product Recommendations**: Context-aware suggestions based on uploaded product catalogs
- **LLM-Powered Explanations**: Natural language justifications for each recommendation
- **Knowledge Graph Architecture**: Product catalogs are transformed into interconnected knowledge graphs for semantic understanding
- **RAG-Based Retrieval**: Retrieval-Augmented Generation ensures accurate, context-aware recommendations
- **Real-time Search**: Amazon product search integration via SerpAPI
- **Interactive Dashboard**: Modern Next.js frontend with file upload and product visualization
- **Vector Embeddings**: Products converted to high-dimensional embeddings for semantic similarity matching

■■ System Architecture

```
<p align="center">


</p>
```

Knowledge Graph RAG Pipeline

1. **Catalog Ingestion**: Product catalogs (CSV/Excel) are uploaded and parsed
2. **Knowledge Graph Construction**:
 - Products, attributes, and relationships are extracted
 - A semantic knowledge graph is built representing product interconnections
 - Entity relationships capture features, categories, pricing, and specifications
3. **Vector Embedding Generation**:

- Each product and its metadata are converted into dense vector embeddings
- Embeddings capture semantic meaning beyond keyword matching

4. **LLM Fine-tuning Context**:

- The knowledge graph provides contextual grounding for the LLM
- RAG architecture retrieves relevant product nodes before generation

5. **Recommendation Generation**:

- Query embeddings are matched against the knowledge graph
- Top-k similar products are retrieved using vector similarity
- LLM generates explanations based on retrieved context

Technical Stack

Backend (Python FastAPI)

- FastAPI for REST API endpoints
- Vector database for embeddings storage
- Knowledge graph construction and querying
- LLM integration (Gemini API)
- SerpAPI for real-time product search
- Sentence transformers for embeddings

Frontend (Next.js + React)

- Next.js 14+ with TypeScript
- Framer Motion for animations
- Tailwind CSS for styling
- File upload with progress tracking
- Real-time product visualization

■ Project Structure

..
E-COMMERCE-PRODUCT-RECOMMENDER/

```
■
■■■ backend/
■ ■■■ __pycache__/
■ ■■■ uploads/ # Uploaded product catalogs
■ ■■■ .env # Environment variables (API keys)
■ ■■■ database.py # Database models and connections
■ ■■■ embeddings.py # Vector embedding generation
■ ■■■ gemini_client.py # LLM API client (Gemini)
■ ■■■ kg_builder.py # Knowledge Graph construction
■ ■■■ llm_explainer.py # LLM explanation generation
■ ■■■ main.py # FastAPI application entry point
```

```
■ ■ ■ ■ requirements.txt # Python dependencies  
■ ■ ■ ■ retriever.py # RAG retrieval logic  
■ ■ ■ ■ serpapi_client.py # SerpAPI integration  
■  
■ ■ ■ frontend/  
■ ■ ■ .next/  
■ ■ ■ node_modules/  
■ ■ ■ public/  
■ ■ ■ src/  
■ ■ ■ ■ app/  
■ ■ ■ ■ components/  
■ ■ ■ ■ ■ dashboard/ # Dashboard components  
■ ■ ■ ■ ■ hero/ # Landing page hero section  
■ ■ ■ ■ ■ intro/ # Introduction section  
■ ■ ■ ■ ■ models/ # 3D models/visualizations  
■ ■ ■ ■ ■ ui/ # Reusable UI components  
■ ■ ■ ■ lib/  
■ ■ ■ ■ components.json  
■ ■ ■ ■ eslint.config.mjs  
■ ■ ■ ■ next.config.ts  
■ ■ ■ ■ package.json  
■ ■ ■ ■ postcss.config.mjs  
■ ■ ■ ■ tsconfig.json  
■ ■ ■ ■ README.md  
...  
...
```

■ Technical Implementation

Backend API Endpoints

1. Upload Catalog

...

POST /upload_catalog

- Accepts: CSV/Excel files
- Process: Parses catalog, builds knowledge graph, generates embeddings
- Returns: Analysis result with best product recommendation

...

2. Get Recommendations

...

GET /recommendations?query={product_name}

- Accepts: Search query string

- Process: Searches Amazon via SerpAPI
- Returns: List of products with images, prices, and URLs

```

#### #### 3. Explain Recommendation

```

POST /explain

- Accepts: Product details JSON
- Process: LLM generates contextual explanation
- Returns: Natural language explanation

```

#### ### Knowledge Graph Construction

The `kg\_builder.py` module transforms product catalogs into semantic knowledge graphs:

```python

Example Knowledge Graph Structure

```
{  
  "nodes": [  
    {"id": "product_1", "name": "Realme Buds Air 5 Pro", "category": "Electronics"},  
    {"id": "feature_anc", "name": "Active Noise Cancellation"}  
],  
  "edges": [  
    {"source": "product_1", "target": "feature_anc", "relation": "has_feature"}  
]  
}
```

Gemini API Integration

The `gemini_client.py` module powers intelligent product analysis using Google's Gemini 2.5 Flash model. The system:

Processing Pipeline:

1. **Catalog Loading**: Accepts CSV, Excel, or JSON product files
2. **Data Optimization**: Limits analysis to 5 products and essential columns to avoid token overflow
3. **Prompt Engineering**: Constructs structured prompts with product details
4. **LLM Analysis**: Gemini evaluates products based on value, features, and ratings
5. **Structured Output**: Returns JSON with best product, reasoning, and alternatives

Configuration:

- **Model**: Gemini 2.5 Flash for fast, accurate analysis
- **Temperature**: 0.3 for consistent, factual recommendations

- **Output Format**: JSON for seamless frontend integration
- **Error Handling**: Comprehensive validation and fallback mechanisms

Vector Embeddings

Products are embedded using sentence transformers:

```
```python
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('all-MiniLM-L6-v2')
product_text = f"{name} {category} {features} {price}"
embedding = model.encode(product_text)
````
```

■ Setup Instructions

Prerequisites

- Python 3.9+
- Node.js 18+
- Gemini API Key
- SerpAPI Key

Backend Setup

1. Navigate to backend directory:

```
```bash
cd backend
````
```

2. Create virtual environment:

```
```bash
python -m venv venv
source venv/bin/activate # On Windows: venv\Scripts\activate
````
```

3. Install dependencies:

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

4. Create ` `.env` file:

```
```env
GEMINI_API_KEY=your_gemini_api_key_here
SERPAPI_KEY=your_serpapi_key_here
````
```

...

5. Run the FastAPI server:

```
```bash
uvicorn main:app --reload --host 0.0.0.0 --port 8000
````
```

Frontend Setup

1. Navigate to frontend directory:

```
```bash
cd frontend
````
```

2. Install dependencies:

```
```bash
npm install
````
```

3. Run development server:

```
```bash
npm run dev
````
```

4. Open browser at `http://localhost:3000`

■ Usage

1. Search Amazon Products

- Enter a product name in the search bar (e.g., "wireless earbuds")
- View top recommendations with images, prices, and Amazon links
- Click any product for AI-generated explanation

```
<p align="center">

</p>
```

2. Upload Custom Catalog (Primary Feature)

- Click or drag-and-drop CSV/Excel file with product data
- Required columns: `name`, `price`, `rating`, `description` (or any 4+ columns)
- System automatically:
 - Builds knowledge graph from catalog relationships
 - Generates vector embeddings for semantic search

- Analyzes products using Gemini 2.5 Flash
- Returns best recommendation with detailed reasoning
- Suggests alternative products with images

```
<p align="center">

</p>
```

3. View LLM Recommendations

- **Main Recommendation**: Displays product image, name, and key benefits
- **Key Benefits**: 3 data-driven reasons explaining the recommendation
- **Suggested Recommendations**: Alternative products with images (fetched via SerpAPI)

```
<p align="center">


</p>
```

4. Product Explanation (Search Results)

- Click on any searched product from Amazon
- AI generates contextual explanation considering:
- Product features and specifications
- Price-to-value ratio
- Competitive alternatives
- User ratings and reviews

■ LLM Usage

The system uses **Gemini 2.5 Flash** for intelligent product analysis and recommendations. The LLM analyzes catalogs using a structured prompt that evaluates products based on:

- **VALUE**: Price-to-performance ratio
- **FEATURES**: Technical specifications and capabilities
- **RATING**: User satisfaction scores

Catalog Analysis Configuration

```
```python
response = client.models.generate_content(
 model="gemini-2.5-flash",
 contents=prompt,
 config={
```

```
"temperature": 0.3, # Low temperature for factual, consistent analysis
"top_p": 0.9, # Nucleus sampling for quality
"max_output_tokens": 4000, # Sufficient for detailed explanations
"response_mime_type": "application/json", # Structured output
"candidate_count": 1,
},
)
...
```
```

Example Catalog Analysis Prompt

The system uses **Gemini 2.5 Flash** with a **Knowledge Graph-enhanced RAG approach**. When a catalog is uploaded, the LLM receives enriched context derived from the knowledge graph:

111

You are analyzing a product catalog through a knowledge graph representation.

The graph captures product entities, their attributes, feature relationships, and competitive positioning.

{context} ----> Generated from kg_builder.py

KNOWLEDGE GRAPH CONTEXT:

—

Nodes (Products):

- Realme Buds Air 5 Pro [₹3499, Rating: 4.1/5]
 - Features: {50dB ANC, 45ms latency, 38H battery, Hi-Res audio}
 - Category: Premium TWS Earbuds
 - Price Segment: Mid-range
 - boAt Airdopes 141 [₹1299, Rating: 4.0/5]
 - Features: {42H playback, ENx tech, 8mm drivers}
 - Category: Budget TWS Earbuds
 - Price Segment: Entry-level
 - OnePlus Nord Buds 2r [₹2299, Rating: 4.2/5]
 - Features: {12.4mm drivers, 38H battery, IP55}
 - Category: Mid-range TWS Earbuds
 - Price Segment: Mid-range
 - Sony WF-C500 [₹4999, Rating: 4.3/5]
 - Features: {DSEE audio, IPX4, 20H battery}
 - Category: Premium TWS Earbuds
 - Price Segment: Premium
 - JBL Tune 130NC TWS [₹5999, Rating: 4.2/5]
 - Features: {Active NC, 40H playback, 4-mic setup}
 - Category: Premium TWS Earbuds

→ Price Segment: Premium

Relationships:

- "Realme Buds Air 5 Pro" --[better_value_than]--> "Sony WF-C500"
- "Realme Buds Air 5 Pro" --[has_feature]--> "50dB ANC"
- "Realme Buds Air 5 Pro" --[competes_with]--> "OnePlus Nord Buds 2r"
- "boAt Airdopes 141" --[budget_alternative_to]--> "Realme Buds Air 5 Pro"

ANALYSIS TASK:

Using the knowledge graph context above, analyze the semantic relationships between products and recommend the best one based on:

1. VALUE: Price-to-performance ratio within the graph
2. FEATURES: Technical superiority evident from feature nodes
3. RATING: User satisfaction compared to connected products

Provide compelling, data-driven reasoning leveraging the graph relationships.

Keep each reason under 50 words.

REQUIRED OUTPUT FORMAT (JSON):

```
{  
  "best_product": "<exact product name from graph>",  
  "reasoning": [  
    "<feature superiority from graph relationships>",  
    "<price-value comparison using graph edges>",  
    "<rating insight relative to competitive nodes>"  
  ],  
  "alternatives": ["<product name>", "<product name>"]  
}  
---
```

Sample LLM Response

```
```json  
{
 "best_product": "Realme Buds Air 5 Pro",
 "reasoning": [
 "It boasts 50dB Active Noise Cancellation and ultra-low 45ms latency, premium features unmatched by competitors under ₹3000, enhancing both audio immersion and gaming.",
 "At ₹3499, it offers superior value, delivering advanced ANC and sound quality comparable to earbuds priced at ₹4999 like the Sony WF-C500, but with more features.",
 "With a strong 4.1/5 user rating and a robust 38-hour total battery life, it provides a reliable, feature-rich experience that balances performance and endurance."
],
 "alternatives": ["OnePlus Nord Buds 2r", "boAt Airdopes 141"]
}
```

}

...

### ### Real-World Example

Based on the uploaded catalog, the system displays:

\*\*■ Recommended Product: Realme Buds Air 5 Pro\*\*

\*\*Key Benefits:\*\*

1. \*\*50dB ANC & Low Latency\*\*: Premium 50dB Active Noise Cancellation and 45ms latency under ■3000 - perfect for immersive audio and gaming
2. \*\*Best Price-Performance Ratio\*\*: At ■3,499, delivers features comparable to ■4,999 Sony WF-C500 with superior ANC
3. \*\*Reliable All-Day Performance\*\*: Strong 4.1/5 rating with 38-hour battery life balances performance and endurance

\*\*Suggested Recommendations:\*\*

- OnePlus Nord Buds 2r (■2,299)
- boAt Airdopes 141 (■1,299)

## ■ Key TechStack

- \*\*FastAPI\*\*: High-performance async Python backend
- \*\*Gemini API\*\*: Google's LLM for natural language generation
- \*\*SerpAPI\*\*: Real-time Amazon product search
- \*\*Sentence Transformers\*\*: State-of-the-art text embeddings
- \*\*Next.js 14\*\*: React framework with server components
- \*\*Framer Motion\*\*: Smooth animations and transitions
- \*\*Tailwind CSS\*\*: Utility-first CSS framework

## ■ Performance Considerations

- \*\*Embedding Cache\*\*: Pre-computed embeddings for faster retrieval
- \*\*Async Processing\*\*: Non-blocking file uploads and API calls
- \*\*Batch Processing\*\*: Multiple product embeddings generated in parallel
- \*\*Vector Indexing\*\*: FAISS/Pinecone for sub-millisecond similarity search

## ■ Security

- API keys stored in environment variables
- File upload validation and sanitization

- CORS configured for frontend-backend communication
- Rate limiting on API endpoints

## ■ License

MIT License - see LICENSE file for details