Building LLM-Augmented Recommendations in Food Delivery Applications.

Introduction: Why Food Recommendation Is Harder Than It Looks

Recommending food is quite tricky. Like movies or music, food preferences are shaped by habits, culture, emotions, and even the weather, but for food these factors change faster and feel more immediate. A spicy ramen might be perfect on a rainy evening but unappealing at noon.

While people are creatures of habit, often ordering the same dishes week after week, they also occasionally crave something new. A recommendation system that gives too familiar options feels boring, while one with options that strays too far is even worse. Good food recommendation systems constantly try to balance comfort and exploration.

Traditional recommendation systems have powered food delivery platforms for years, but the advent of large language models is improving how recommendations are generated and how they feel to users.

This article walks through:

- How food recommendation systems traditionally work
- How companies like Just Eat Takeaway, Swiggy, and DoorDash use LLMs to enhance recommendations.
- Our experiment with LLM-augmented recommendations through a demo food delivery platform

Traditional Recommendation Systems in Food Delivery

Below are the three main traditional recommendation techniques that still form the backbone of most production systems today.

Collaborative Filtering

Collaborative filtering works on the simple principle that users who had similar preferences in the past will like similar things in the future. Below are the two primary forms:

- **User-based:** Recommend items liked by similar users.

  Example: If you and another user often order spicy Indian dishes, a new dish that they like may be recommended to you.

- **Item-based:** Recommend items frequently ordered together or enjoyed by the group of similar users.

  Example: "People who ordered Margherita Pizza also ordered Garlic Bread."

Challenges of this method:

- **Cold start:** New users will have no preference history, making it hard to find similar users to them until they've ordered several times. New restaurants or menu items face the same challenge as they can't be recommended through collaborative signals until enough users have tried them..
- **Sparse data:** Most users order infrequently (maybe only a couple times weekly at most), so the data which might not be enough find users truly similar to you meaning weakly related users are included.
- **Weak explainability:** There is often no explanation as to why a recommendation is shown beyond "users similar to you liked this".

Content-Based Filtering

Content-based filtering focuses on the attributes of food items rather than other users. Each item is described by metadata such as:

- Cuisine (Italian, Indian, Thai)
- Flavour profile (spicy, sweet, savoury)
- Dietary attributes (vegetarian, gluten-free)
- Meal timing (breakfast, dinner, snack)
- Ingredients and preparation (grilled, fried, baked)

The system builds a profile of a user's preferences and recommends items that match.

Example: If a user often orders vegetarian Italian dishes with creamy textures, the system may suggest risotto or pasta alfredo.

Note: This works really well for food because food has rich metadata, and user preferences are relatively stable (e.g dietary, vegan).

Hybrid & ML-Based Approaches

Production recommendation systems often combine multiple techniques to leverage the strengths of different approaches. They integrate **collaborative filtering**, **content-based filtering**, and/or use advanced machine learning models such as **neural networks** or **embedding-based representations**.

For example, users and foods can be represented as **vectors in a latent space**, and techniques like **matrix factorization** and **vector similarity search** can then be used to find similar users and foods. Additionally, **learned ranking models** can sort the final recommendations based on metrics such as **click-through rate, past orders, or other engagement signals**.

Common hybrid strategies include:

- **Weighted combinations:** Combine outputs score from multiple recommendation methods (e.g., collaborative filtering + content-based) using weighted scores to balance their contributions.
- **Switching or cascade approaches:** Dynamically select which method to apply based on situation. For instance, content-based methods may be favoured for new users with limited interaction history (cold-start problem), while collaborative filtering is used for established users.
- **Unified / reinforcement learning models:** Train a reinforcement learning model using data collected from other approaches. This model is then continuously improved over time, as the user either follows through with the recommendation or not.

By blending all these techniques, one can craft an enjoyable and continuously improving user experience with a recommendation system.

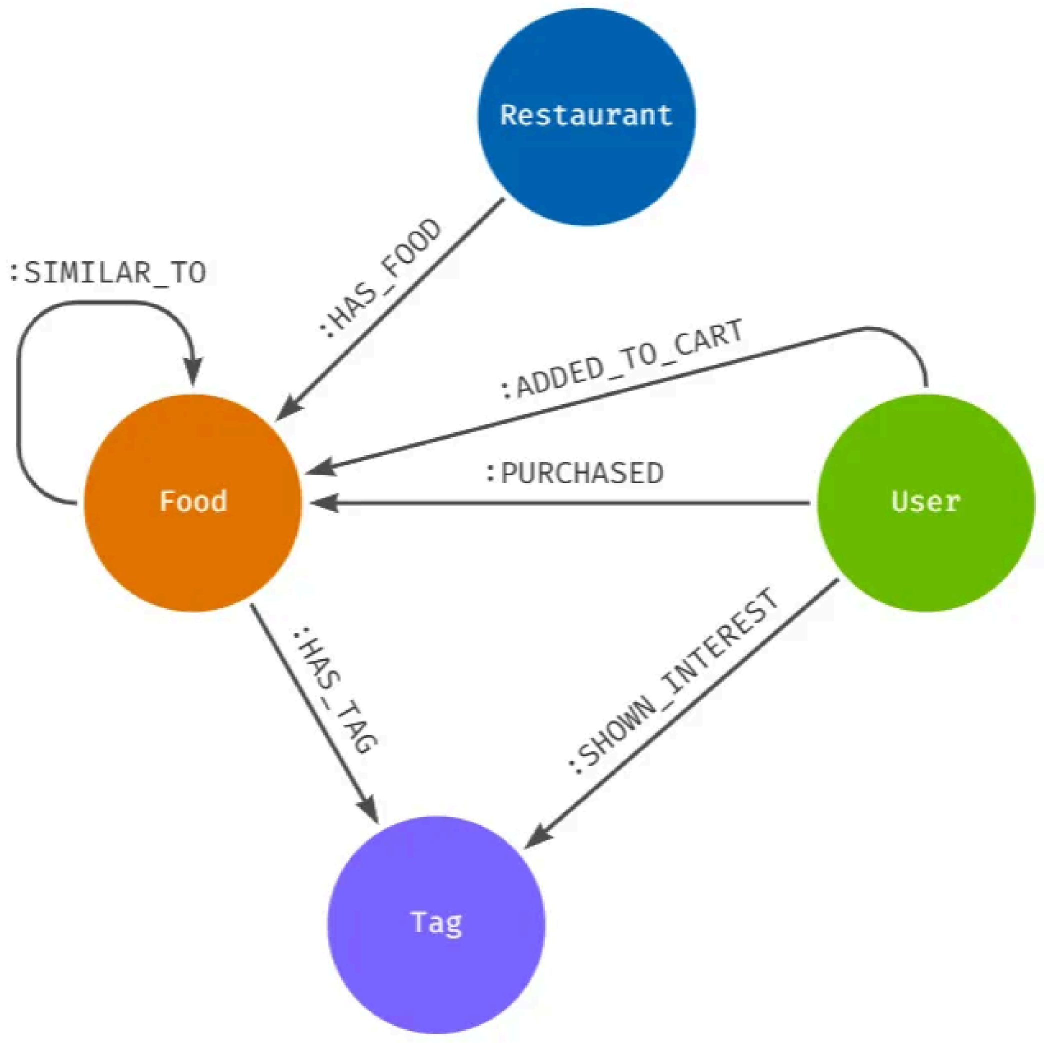
Why having good AI Models Matter for Recommendation Systems:

- **Fast embedding generation** for real-time similarity search across thousands of menu items
- **Consistent API reliability** for production-grade user experiences with guaranteed SLAs
- **Cost-effective inference** to handle high query volumes without breaking the budget, with transparent per token pricing and no hidden costs
- **Automatic scaling** that grows with your business from prototype to millions of daily users

Exploring These Ideas in Practice: Building a food recommendation app

We built a demo food delivery platform called **food recommendation app** to explore augmenting an existing recommendation system with LLMs. For recommendations, we chose a simple and popular approach: **Content-Based Filtering**, implemented using a **Neo4j graph database**.

The Content-Based Filtering Graph Recommendation System



We started by creating mock data consisting of about **30 restaurants** and **50 different food cuisines** across different cultures (Thai, Indian, German, etc.). Each food item includes a description and, most importantly, a set of **tags**. These tags serve as the backbone of our recommendation system.

Examples of these tags include things like: `vegetarian`, `spicy`, `rice`, `chicken`, `snack`, and so on. You can think of them as keyword metadata attached to each food.

To build the recommendation system, we first populated our Neo4j graph with all the data we would need later.

We created:

- **Food nodes**
- **Restaurant nodes**
- **Tag nodes**

Then we connected them using relationships:

- A **Restaurant** is connected to all the **Food** items it serves
- A **Food** node is connected to all of its **Tag** nodes

In addition to this, there is one more important relationship we create during pre-population: **food-to-food similarity**.

Food Similarity Relationship

We use the tags attached to each food and graph traversal to determine similarity. Two foods are considered similar if their **similarity strength is above 40%**.

The similarity strength is calculated as:

Similarity = (Number of Shared Tags) / max(Tags of Food A, Tags of Food B)

Example

Consider two dishes:

- **Chicken Tikka Masala**

  Tags: `{Indian, Spicy, Chicken, Rice}`

- **Paneer Tikka**

  Tags: `{Indian, Spicy, Paneer, Vegetarian, Grilled}`

They share two tags namely `{Indian, Spicy}`.

Similarity = 2 / max(4,5)

= 2 / 5
= 0.40 (40%)

Since the similarity is 40%, we create a `SIMILAR_TO` relationship between the two food nodes and store the similarity strength as a property on that relationship.

User Behaviour Tracking

Once pre-population is done, when a user joins the platform, we create a **User node**.

We then start tracking user behavior such as:

- Purchasing a food
- Adding a food to the cart
- Viewing a food's description

For each of these actions:

- We create a relationship from the **User** node to the **Food** node (`PURCHASED`, `ADDED_TO_CART`, etc.)
- Since each food is already connected to tags, we also create a `SHOWN_INTEREST` relationship from the **User** node to the **Tag** nodes of that food

Each `SHOWN_INTEREST` relationship has a **count** property to represent interest strength:

- Purchase ⇒ `+1`
- Add to cart ⇒ `+0.5`
- View description ⇒ `+0.05`

Over time, this builds a clear profile of what the user is interested in.

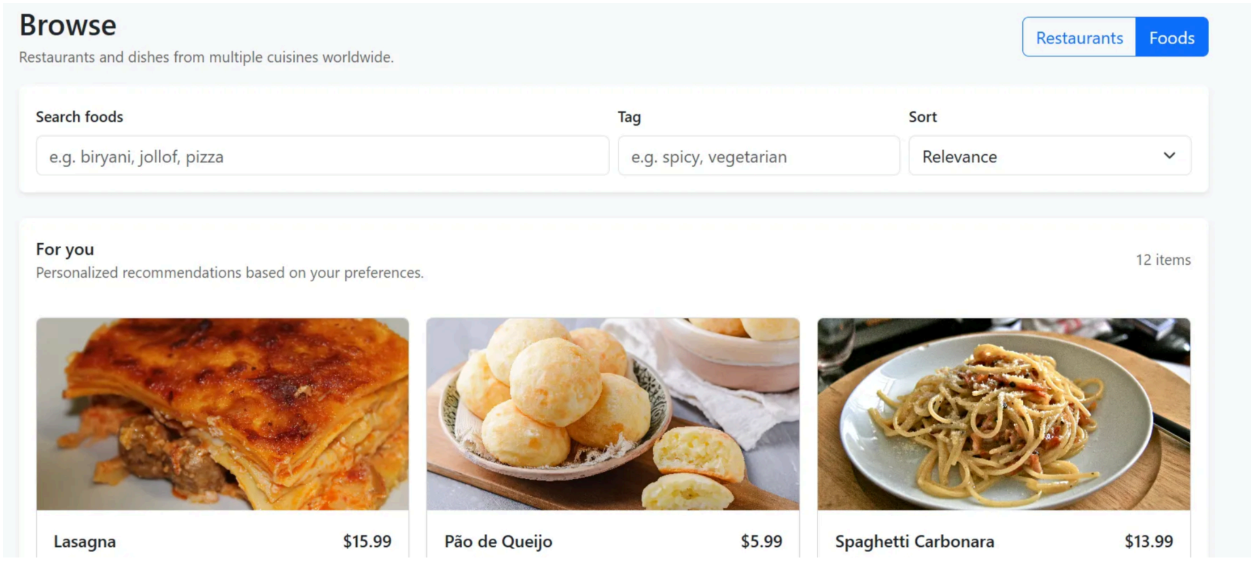
Generating Recommendations

To recommend foods to a user, we follow these steps:

1. **Collect candidate foods**
    - Foods the user has purchased or added to cart
    - Similar foods connected via the `SIMILAR_TO` relationship
2. **Score each food** based on:
    - **Direct interaction boost**
        - Purchased items: `1.2x`
        - Cart items: `1.0x`
    - **Interest score**
        - Sum of `SHOWN_INTEREST` counts for matching tags

        - **Similarity score**
            - Weighted by the `SIMILAR_TO` relationship strength
3. **Rank and return**
    - We return the top-K highest-scoring foods

This final ranked list is what is shown in the **“For You”** section in food recommendation app.



2. The "About Me" Page: Your Food Wrapped

Inspired by Spotify Wrapped, the "About Me" page is a personalized profile generated by the LLM using the user's information, preferences, and backend data. This page evolves continuously as the user interacts with the platform.

- **Tags and Taste Profile**
- **Foods the user has been exploring recently**
- **Picks to try next**, recommended by the LLM with a mix of the familiar and new exploration options

Conclusion: The Future of LLM-Augmented Recommendation

Food recommendation systems have come a long way from collaborative filtering to sophisticated hybrid approaches, but LLMs represent the next leap forward. As we've seen with platforms like **Just Eat Takeaway**, **Swiggy**, **DoorDash**, and through our own **food recommendation app** experiment, LLMs augment traditional recommendation systems in powerful ways by adding natural language understanding, enabling cross-domain recommendations, and generating personalized content, all which creates a more engaging user experience.