Technical Report: Cook Compass

Group 20: Trayan Tsonev, Dominik Vallazza, Elias Hirsch, Juraj Simkovic

Date: January 21, 2026

Abstract

Cook Compass is a Generative AI application designed to solve the "dinner dilemma" for students and home cooks. The problem is that traditional search engines prioritize SEO-optimized, life-story-heavy content over concise cooking instructions, and generic chatbots often hallucinate culinary nonsensical recipes. We developed a Retrieval-Augmented Generation (RAG) system that allows users to input natural language queries (e.g., ingredients, constraints) and retrieves verified recipes from a structured dataset. The system filters for relevance and dietary compliance, aiming to increase the success rate of finding a cookable meal compared to standard browsing.

Description

User Journey

The ideal user journey consists of three steps:

1. **Input:** The user opens the Streamlit web interface and is greeted by the assistant. They input a natural language query based on their current situation, such as "I have spinach and tomatoes, but no oven. I want a vegetarian pasta dish.".
2. **Processing:** The system rewrites this query to optimize it for vector search, retrieves the most relevant recipes from the underlying database, and passes them as context to the Large Language Model (LLM).
3. **Result:** The user receives a structured response containing the name, ingredients, and instructions for a specific recipe that matches their constraints, along with a polite explanation.

Technical Approach

We implemented a RAG (Retrieval-Augmented Generation) architecture to ground the LLM's responses in factual data.

- Framework: We used Haystack 2.0 for the orchestration of the pipeline. It allowed us to modularly connect embedding, retrieval, and generation steps.
- Data Ingestion: We utilized a subset of the Food.com dataset (`small_recipes.csv`). A custom ingestion script (`ingest.py`) cleans the data (parsing string-lists into actual text) and creates embeddings using ChromaDB as the vector store.
- Retrieval: We employ dense retrieval using the `BAAI/bge-base-en-v1.5` embedding model. This captures semantic similarity (e.g., matching "no oven" to "stovetop") better than keyword search.
- Keyword Filtering: To improve precision for dietary and constraint-based queries, we use a two-stage filtering system. After vector retrieval returns the top-k semantically similar recipes, the LLM extracts relevant keywords from the user query—including dietary tags (vegan, vegetarian, gluten-free, kosher, lactose-free, low-carb, high-

protein, diabetic) and lifestyle constraints (inexpensive, 15-minutes-or-less, healthy). The system then filters results to include only recipes whose metadata contains all identified keywords. For example, "quick vegan protein meals" returns only recipes tagged with vegan, high-protein, and 15-minutes-or-less. If no recipes match all constraints, the system falls back to the original top-k results to avoid empty responses. This approach combines semantic retrieval flexibility with strict metadata filtering.
- Generation: For the generation step, we utilized the `Qwen/Qwen2.5-7B-Instruct` model via the Hugging Face Serverless API. We chose this for its cost-efficiency.

Challenges

We had problems with the size of the data set because we underestimated the resources that would have been necessary for this. Therefore, we had to limit ourselves to a greatly reduced sample of the data.

Evaluation

We adopted an LLM-as-a-Judge approach to evaluate the system, as manual review of generative output is unscalable. Using a Jupyter Notebook (`notebooks/LLM_as_judge+visualise_results.ipynb`), we tasked a stronger model (`Llama-3-8B`) to act as an impartial judge.

The judge evaluated user-response pairs on three criteria (0-10 scale):

- Relevance: Did the recipe use the requested ingredients?
- Healthiness: Was the recipe nutritional balanced?
- Taste: Was the culinary combination plausible?

Results: Based on 80 test queries evaluated by `Llama-3.1-8B-Instruct` :

- **Relevance: 7.97/10** - The system scored high on direct ingredient queries.
- **Healthiness: 6.91/10** - Moderate scores could be reflecting the varied nutritional quality present in the Food.com dataset itself.
- **Taste: 7.83/10** - Strong scores indicating that by retrieving from a curated dataset rather than letting the LLM hallucinate from scratch, we maintained high culinary quality.

The evaluation pipeline generated queries using three templates: standard keyword requests (35%), personal dietary conditions (35%), and ingredient-based queries with constraints (30%).

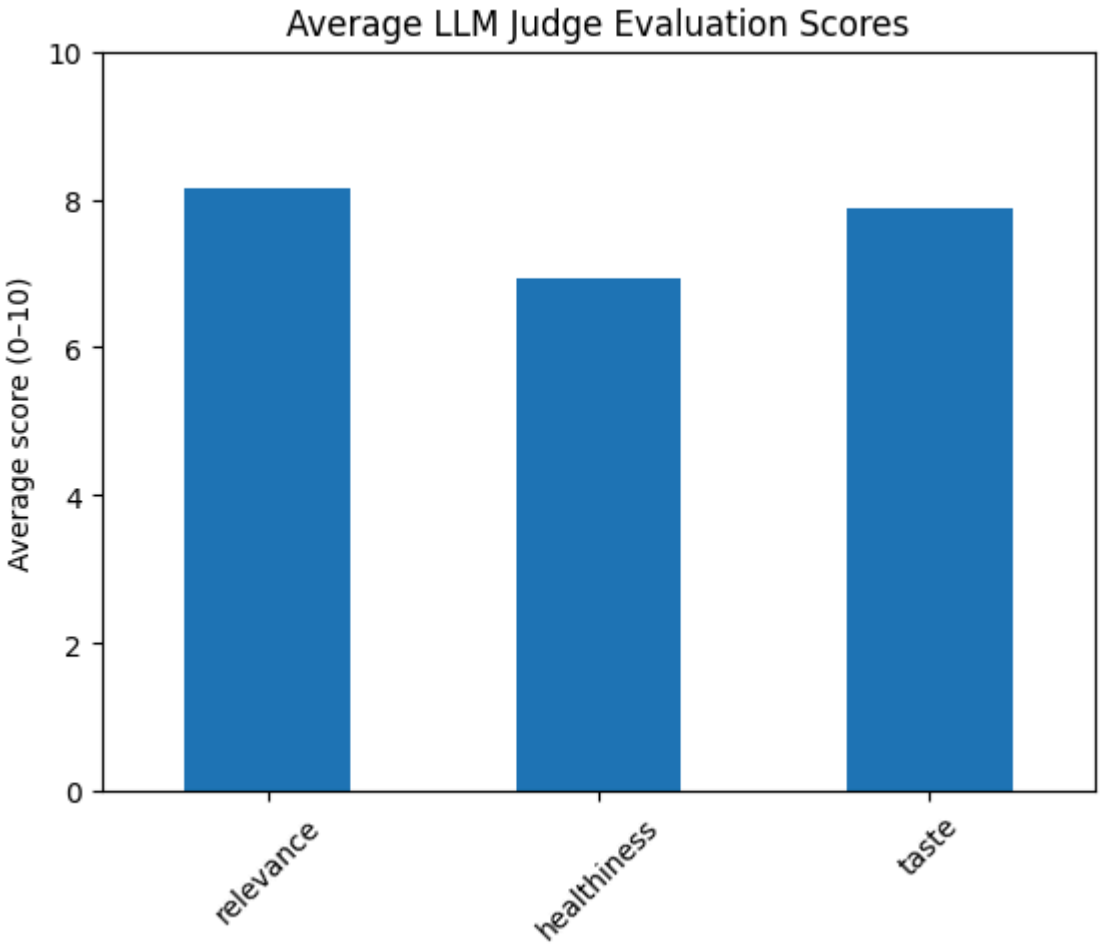

Figure 1: Average scores across Relevance, Healthiness, and Taste.

Did it work?: Partially. The system succeeds at its core goal—returning real recipes instead of hallucinated ones—but has notable limitations:

- Keyword filtering correctly enforces dietary constraints when matching recipes exist.
- No complete fabrications (ingredients/steps mostly sourced from dataset).
- Dataset size bottleneck: With only 450 recipes some reasonable queries return poor matches.
- Evaluation bias: LLM-as-a-Judge is not ground truth—it may score nonsensical but well-formatted responses highly.

The 7-8/10 scores are misleading. They reflect average performance across easy and hard queries. In practice, performance is bimodal: excellent for common vegetarian pasta dishes, poor for niche dietary combinations. The system is a proof-of-concept that validates the RAG approach but is not production-ready without significant dataset expansion.

Reflection

Human Agency

Our system increases human agency. By automating the tedious process of filtering through SEO-bloated blogs and cross-referencing ingredients, we reduce the cognitive load on the user. This empowers users to cook meals with what

they actually have on hand, rather than feeling forced to order takeout or buy unnecessary groceries. The user remains the final decision-maker, but they are presented with better options faster.

Future Development

Yes, we would develop this further. While the current prototype works, the dataset site constraint is currently soft-enforced via the dataset and prompt. A future version would need:

1. Dataset Expansion: The current 450-recipe subset limits variety. Scaling to 5,000-10,000 recipes would improve coverage, especially for underrepresented categories like lactose-free (currently 0 recipes).
2. Multi-Modal Input: Allow users to upload a photo of their fridge/pantry and use vision-language models to automatically extract available ingredients.
3. Personalization: Adding a user memory module so the bot remembers allergies or dislikes across sessions.
4. Nutritional Analysis: Add the nutrition data (calories, protein, carbs) in the UI to help users make informed choices beyond just taste.