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, specifically focusing on vegetarian cuisine. 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.
- Generation: For the generation step, we utilized the `Qwen/Qwen3-VL-8B-Instruct` model via the Hugging Face Serverless API. We chose this for its cost-efficiency.

Challenges

TODO (Describe the challenges / technical obstacles you have faced (what didn't just magically work on the first try)

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 (`LLM_as_judge.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: TODO (average scores from notebook).

- Relevance: The system scored high (~8/10) on direct ingredient queries but struggled slightly with negative constraints (e.g., "no garlic").
- Taste/Healthiness: Scores averaged 7-9/10, indicating that by retrieving from a curated dataset (Food.com) rather than letting the LLM hallucinate from scratch, we maintained high culinary quality.

TODO (average scores from notebook).

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 "Vegetarian" constraint is currently soft-enforced via the dataset and prompt. A production version would need:

1. Hard Filtering: Implementing metadata filtering in ChromaDB to strictly enforce dietary flags (Vegan/Vegetarian) before the LLM sees the data.
2. Personalization: Adding a user memory module so the bot remembers allergies or dislikes across sessions.
3. Hybrid Retrieval: Finishing the implementation of the HybridRetriever class to better handle specific recipe names alongside semantic descriptions.