Cultural Cooking Assistant: NLP and Language Models for Recipe Recommendation and Adaptation

Abdulaziz Al-Shayef
Computer Science
UNC Chapel Hill
alshayef@cs.unc.edu

Nabeel Abdul Rahman
Computer Science
UNC Chapel Hill
nrahman@unc.edu

***Abstract*—Choosing what to cook with only the ingredients at hand can be time-consuming and creatively limiting, especially for college students juggling academics, work, and social commitments. We present the Cultural Cooking Assistant, a unified NLP and generative language model pipeline that delivers personalized recipe recommendations and culturally enriched adaptations based on user-specified ingredients. Our approach integrates TF-IDF ingredient vectorization for rapid retrieval, Apriori-based association rule mining to uncover cuisine-specific ingredient patterns, Word2Vec embeddings for semantically guided substitutions, and a fine-tuned GPT-2 model to generate clear, step-by-step cooking instructions.**

I. Introduction

Planning daily meals can be a significant source of stress for college students, who must juggle demanding coursework, part-time jobs, and social commitments. Time constraints often force students to rely on quick but repetitive options, such as instant noodles, microwaveable dinners, or the same simple stir-fry, leading to culinary monotony and diminished nutritional variety. Furthermore, limited budgets and unfamiliarity with diverse ingredients can discourage experimentation, preventing students from exploring the rich tapestry of global cuisines.

Traditional recipe platforms and search engines require users to manually sift through hundreds of options and adapt them to available ingredients, a process that can be both time-consuming and overwhelming. While many apps provide meal kit delivery or grocery recommendations, they often do not account for what users already have on hand, nor do they offer personalized cultural variations to accommodate diverse tastes or dietary restrictions.

To address these challenges, we developed the Cultural Cooking Assistant: an intelligent, end-to-end system that streamlines meal planning, maximizes ingredient utilization, and fosters cultural exploration. At its core, the assistant accepts a list of ingredients and an optional target cuisine, then retrieves matching recipes, suggests culturally relevant ingredient substitutions, and generates step-by-step cooking instructions tailored to the user's pantry.

The system's high-level architecture comprises four key components. First, a TF-IDF–based retrieval module ranks recipes by similarity to the input ingredients. Second, an association-rule mining pipeline (Apriori) discovers cuisine-specific ingredient patterns, enabling rule-based cultural substitutions. Third, Word2Vec embeddings supplement the substitution process by identifying semantically similar ingredients for greater flexibility. Finally, a fine-tuned GPT-2 model produces coherent and context-aware instructional text, ensuring that even novice cooks can follow the generated recipes with confidence.

By integrating retrieval, adaptation, and generation within a single framework, the Cultural Cooking Assistant reduces the cognitive load of meal preparation, encourages creative use of existing ingredients, and introduces students to new cultural flavors, all through a simple, user-friendly interface.

II. Our Approach

Our methodology comprises five core components, each tailored to maximize performance and interpretability.

A. Dataset Collection and Preprocessing

Our pipeline begins with a raw dump of over 20,000 recipes in JSON format from the Kaggle "Food.com Recipes and Interactions" dataset, covering 20 distinct world cuisines, ingredients, and instructions. We first load the JSON into a pandas DataFrame (`pd.read_json`), then perform an initial exploratory analysis, checking for missing fields, duplicate recipes, and the distribution of cuisines and ingredient counts per recipe.

To prepare the data for NLP, we apply the following steps in sequence:

1) **Normalization**
   - **Lowercasing & stripping**: Convert all ingredient and instruction text to lowercase; remove leading/-trailing whitespace.
   - **Punctuation & numeric removal**: Use regular expressions to strip out punctuation (commas, periods, parentheses) and numerical quantities (e.g., "2 cups"). This reduces noise and prevents tokens like "2" from polluting the vocabulary.
2) **Tokenization & Lemmatization**
   - **NLTK word_tokenize**: Split each ingredient string and instruction sentence into tokens.
   - **WordNet lemmatizer**: Reduce tokens to their base forms (e.g. "tomatoes" $\rightarrow$ "tomato", "chopped" $\rightarrow$ "chop") so semantically identical words map to the same token.

3) **Stop-word Filtering & Domain-specific Pruning**
   - **Standard stop-words**: Remove common English stop-words (e.g. “and”, “of”, “the”) using NLTK’s list.
   - **Culinary stop-words**: Filter out generic terms that add little semantic value (e.g. “fresh”, “large”, “small”).
4) **Multi-word Ingredient Standardization**
   - Identify common compound ingredients (e.g., “tomato paste”, “olive oil”) and join them with underscores (`tomato_paste`) to ensure that they remain single tokens.
   - Apply a small synonym map (e.g., ’chili powder’ to ’chili powder’) to unify regional spelling variants.
5) **Data Structuring & Storage**
   - Store the cleaned tokens back into new columns: `tokens` for ingredients and `instr_tokens` for instructions.
   - Serialize the processed DataFrame to a Parquet file for fast loading in later pipeline stages.

After preprocessing, our corpus contains a clean, deduplicated set of recipes with standardized token lists, which is ready for vectorization, association-rule mining, and language-model fine-tuning.

B. Ingredient-based Retrieval

To match a user’s available ingredients with suitable recipes, we transform each recipe’s ingredient list into a vector representation using Term Frequency–Inverse Document Frequency. In this process, every unique ingredient in our vocabulary contributes one dimension, producing a high-dimensional but sparse matrix whose rows correspond to recipes. When a user submits a list of ingredients, we apply the same TF-IDF transformation and compute the cosine similarity between the user vector and every recipe vector. The recipes with the highest similarity scores are returned as the most relevant matches. In practice, this retrieval completes in well under one second, even on modest hardware, thanks to efficient sparse matrix operations and indexed similarity searches. Users, therefore, experience near-instantaneous feedback, allowing them to iterate on ingredient choices without delay.

C. Cultural Ingredient Substitution

To infuse retrieved recipes with authentic cultural flavors, we first examine each cuisine’s characteristic ingredient patterns. By encoding recipes for a given cuisine as binary feature vectors and running the Apriori algorithm with a minimum support threshold of ten percent, we identify frequent sets of ingredients that define that culinary tradition. We then extract association rules with high confidence (eighty percent or greater), focusing on those whose antecedents contain multiple ingredients. For example, the simultaneous presence of cumin and turmeric implies the use of garam masala. In addition to these rule-based substitutions, we train a Word2Vec model on the entire recipe corpus so that semantically similar ingredients can be suggested when no strong rule applies. Each candidate substitute is scored by combining its embedding similarity with its empirical frequency in the target cuisine, ensuring that both semantically and culturally appropriate replacements are offered. This hybrid approach balances the precision of explicit association rules with the flexibility of learned semantic relationships. This part proved to be a bit challenging, and we weren’t able to function as intended. We plan on attempting this again as part of future work.

D. Recipe Generation with GPT-2

Once an adapted ingredient list is finalized, we generate detailed cooking instructions by fine tuned a GPT-2 small model on a structured recipe corpus. Each training example begins with an “Ingredients:” prompt listing the cleaned tokens, followed by a “Steps:” sequence of numbered instructions. Training proceeds for one epoch with a block size of 256 tokens and a batch size of two, which we found sufficient to capture recipe structure without overfitting. The generated output retains clear step delineation and cooking conventions, such as preheating, sautéing, and simmering, so that even novice cooks can follow along confidently.

E. Integration and Interface

All components are woven together within a FastAPI backend that exposes three main endpoints: one for retrieving matching recipes, another for applying cultural substitutions, and a third for generating full cooking instructions. When a user enters ingredients and selects a cuisine in the web interface, asynchronous calls first retrieve candidate recipes, then transform them via the substitution pipeline, and finally request step-by-step instructions from the language model. On the frontend, a lightweight HTML and JavaScript application displays retrieved recipes side-by-side with their culturally adapted counterparts, and it renders the generated instructions in a clean, scrollable panel. This design ensures a seamless user experience, allowing students to move from ingredient entry to cooking with just a few clicks, all while exploring diverse culinary traditions.

III. Challenges

Throughout development, we confronted several obstacles that required iterative refinement of our methods.

First, ensuring substitution relevance proved difficult because our association rules often captured only the most dominant ingredient pairings, and Word2Vec embeddings tended to suggest globally common items rather than culturally specific ones. This behavior arose from skewed ingredient frequencies in the dataset and the embedding training objective, which emphasizes overall co-occurrence rather than cuisine-specific relationships. To remedy this, we increased the minimum support and confidence thresholds for Apriori mining, applied frequency-weighted filtering to Word2Vec candidates based on per-cuisine usage, and prioritized multi-ingredient antecedent rules to reinforce meaningful cultural patterns. These adjustments led to a notable increase in substitution precision without excessively narrowing the candidate set.

Second, underrepresented cuisines suffered from sparse, high-confidence rules due to limited data coverage. The smaller recipe counts in these categories prevented the Apriori algorithm from extracting reliable patterns, resulting in few or no substitutions for certain target cuisines. We addressed this by changing to a different dataset, expanding ingredient synonym lists to unify analogous terms, and incorporating a fallback mechanism that applies only the most semantically similar embeddings when rule-based options are absent. This hybrid strategy improved substitution coverage while maintaining cultural authenticity.

IV. Alternative Approaches

Beyond our current TF-IDF+Apriori+Word2Vec+GPT-2 pipeline, two promising directions could have fundamentally changed both implementation complexity and end results:

1) **Transformer-based Retrieval & Substitution**
   Rather than relying on TF-IDF for ingredient matching and Apriori/Word2Vec for substitutions, one could employ pre-trained transformer models (e.g. Sentence-BERT) to embed entire ingredient lists and recipes into a dense semantic space.
   - **How it works**: You'd pass each recipe's ingredient string through a fine-tuned SBERT encoder to produce a fixed-length vector; user queries receive the same treatment. Similarity search then becomes a nearest-neighbor lookup in that continuous space, capturing synonymy and subtle semantic relations (e.g. treating "cilantro" and "coriander" as near-identical). For substitutions, the same embedding space can retrieve the top cuisine-specific tokens closest to a target ingredient.
   - **Potential outcome**: Retrieval precision would likely improve for paraphrased or rare ingredient combinations, and substitutions could become more contextually appropriate without hand-tuning support/confidence thresholds. However, this comes at the cost of heavier infrastructure (GPU-accelerated embedding and ANN indexing) and reduced interpretability compared to explicit Apriori rules.
2) **LLM-Centered Substitution & Generation (GPT-4)**
   Instead of a two-stage substitution pipeline with GPT-2 generation, one could leverage a single, more powerful LLM (such as GPT-4) via prompt engineering to handle both cultural adaptation and instruction generation in one pass.
   - **How it works**: Given a prompt like:
     "Here are ingredients: chicken, rice, salt, pepper. Please rewrite this recipe in an Indian style, suggesting ingredient swaps and providing step-by-step cooking instructions."
     GPT-4 could output both the modified ingredient list and full instructions directly.
   - **Potential outcome**: This unified approach simplifies the codebase, eliminates the need for separate rule mining and embedding models, and may produce more creative or nuanced cultural spins. On the downside, it requires paid API access or substantial on-premise resources, introduces greater risk of hallucinated substitutions, and reduces repeatability, where every API call might yield slightly different results.

Each of these alternatives trades off interpretability, infrastructure complexity, and cost against retrieval/substitution accuracy and generative richness. Depending on project constraints (compute budget, maintainability, desired control), one might favor a hybrid model: transformer embeddings for retrieval and LLM prompts for final adaptation, thereby combining the strengths of both paradigms.

V. Future Work

In the coming months, we will deepen personalization by integrating collaborative filtering techniques that learn from real usage patterns rather than static ingredient lists. We plan to try out different methods to improve the mapping and substitution system so that users can properly create a new cuisine mix. As students rate and save their favorite recipes, and note which substitutions they actually try. The system will identify clusters of users with similar tastes and automatically surface new dishes that those peers enjoyed. Over time, this will allow the assistant to suggest not only relevant recipes but also preferred cooking styles, spice levels, and portion sizes tailored to each individual's lifestyle and dietary constraints.

At the same time, we plan to substantially enrich our recipe corpus. Beyond the Kaggle dataset, we will ingest community-submitted recipes, open-source cookbook archives, and regional food blogs, ensuring robust coverage of underrepresented cuisines and emerging food trends. This effort will be accompanied by semi-automated data curation, using named-entity recognition to tag ingredients and cooking methods, and by periodic retraining of our Apriori and Word2Vec models to capture newly added patterns. The result will be a living, growing knowledge base that reflects the evolving tastes of our user community.

On the front-end, we envision a mobile application augmented with voice recognition so that students can add ingredients simply by speaking them aloud or scanning package barcodes. A built-in grocery-list generator will translate recipe plans into shopping items, and a calendar view will help juggle meal prep alongside exams and work shifts. Push notifications, which can be timed to remind users when pantry staples are running low, will turn reactive meal planning into a proactive, stress-free experience.

Finally, we will conduct a semester-long field study within multiple campus dorms. By combining quantitative metrics, such as frequency of use, diversity of cuisines tried, and substitution acceptance rates, with qualitative feedback from focus groups and cooking diaries, we will evaluate how sustained interaction with the Cultural Cooking Assistant influences students' cooking confidence, dietary variety, and openness to new cultures. Insights from this longitudinal research will drive iterative improvements in both our recommendation algorithms and the user interface, guiding us toward a truly adaptive, student-centered culinary companion.

VI. Conclusion

The Cultural Cooking Assistant effectively combines NLP retrieval, cultural pattern mining, and language generation to support ingredient-based recipe discovery and cultural adaptation. Continued enhancements and broader deployment hold promise for enriching the culinary experiences of diverse student populations.

References

[1] X. Chen and L. Zhu. ”Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images.” CVPR, 2019.

[2] T. Mikolov et al. ”Efficient Estimation of Word Representations in Vector Space.” arXiv:1301.3781, 2013.

[3] A. Radford et al. ”Language Models are Unsupervised Multitask Learners.” OpenAI Technical Report, 2019.