A HEALTH AND BUDGET FOCUSED AI POWERED ADAPTIVE MEAL RECOMMENDATION APPLICATION INTEGRATING PREDICTIVE NUTRITIONAL ANALYTICS AND CONTINUOUS WEEKLY DIETARY PROFILING FOR HYPER PERSONALIZED WELLNESS OPTIMIZATION

Quinn Desimone [1], Julian Avellaneda [2]

[1] Dexter Southfield School, 20 Newton St, Brookline, MA 02445
[2] California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

Nutrition tracking tools often focus narrowly on macronutrients, and they completely neglect micronutritional accuracy, ingredient complexity, and personalization. Further, many applications also fail to address issues such as meal cost. This project addresses these gaps by developing an AI-powered adaptive meal recommendation application. This system integrates ingredient inventories, user health profiles, and predictive analytics to generate individualized meal plans. Being built with Flutter, Firebase, and large language models, the system provides real-time ingredient parsing, dietary restriction enforcement, and tailored nutrition guidance. This project was not simple, however. Several challenges were encountered during development. Thus includes inconsistent food classification, data synchronization issues across app pages, and ensuring the AI generated consistent outputs in the required JSON schema. These challenges were resolved through a myriad of methods. Including, backend integration, stricter prompting, and external database support. Two experiments were completed to evaluate blind spots: dietary restriction enforcement (mean accuracy 90.7) and nutritional information generation across food categories (perfect accuracy for meat, lower for complex foods). Results confirm strong baseline reliability but highlight the need for refinement in parsing compound ingredients and micronutrients. Ultimately, this system demonstrates the feasibility of delivering hyper-personalized, health-focused meal recommendations that advance beyond existing static or generalized solutions.

KEYWORDS

Health, Nutritional Info, Mobile application development via flutter, Inventory food management, Meal recommendation via health issues

1. INTRODUCTION

The balance between macronutrients, cost, and micronutrients is one area where modern society exhibits a severe lack of nutritional literacy. The public conversation frequently ignores vitamins, minerals, and bioavailability factors that are critical to long-term health in favor of

David C. Wyld et al. (Eds): SIGI, CSTY, AI, NMOCT, BIOS, AIMLNET, MaVaS, BINLP – 2025
pp. 85-97, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.151908

concentrating only on calories, protein, fats, and carbs [3]. This discrepancy has led to misunderstandings about what makes a "healthy" diet. For instance, despite being promoted as healthful, protein shakes, energy drinks, and goods like Fairlife milk usually have high sugar content, additives, or unbalanced nutrient profiles. The popularity of these "convenience health foods" is a result of industry incentives as well as consumer demand for ease of use, but it ultimately leads to unnoticed nutritional deficiencies (Monteiro et al., 2019).

The lack of understanding about how to combine readily available ingredients to create nutrientdense, well-balanced meals is equally problematic. Many people rely too much on supplements, taking pills or multivitamins without realizing that whole foods and synthetic alternatives differ greatly in terms of bioavailability, or the body's capacity to absorb and use nutrients. For example, some fat-soluble vitamins need dietary fats to be absorbed. Another, iron from plant sources is less bioavailable than iron from meat. These subtleties are frequently disregarded, which results in pervasive inefficiencies in eating habits.

The effects of these disparities go well beyond personal well-being. Populations that routinely make poor dietary choices are at risk for chronic illnesses, higher medical expenses, and a lower standard of living overtime. Poor diet is linked to diseases like obesity, diabetes, and cardiovascular disease, which continue to be the world's leading causes of death, according to the CDC. Therefore, a key public health priority is creating systems that encourage macro- and micronutrient literacy while pointing people in the direction of cost friendly, efficient meal planning.

NutrifyAI (Han et al., 2024) uses computer vision (YOLOv8) and the Edamam API to detect food in real time and provide nutritional insights. While this is effective for static image-based scenarios, it lacks adaptability to non-visual input and does not enforce user-specific dietary constraints. Our system improves on this by accepting flexible ingredient input and enforcing personalized restrictions through dynamic filtering.

MealRec (Zhou et al., 2022) introduced bundled meal recommendations based on co-rated meal trios. However, its reliance on user reviews and Western meal structures limits real-world dietary applicability. Our system avoids this by assembling meals in real time based on explicit user input, offering culturally agnostic, goal-specific recommendations that reflect actual consumption patterns.

MealRec+ (Li et al., 2024) builds on MealRec with nutritional scoring but lacks real-time interactivity and personalization. It uses static metrics for evaluation and cannot adapt to userspecific needs. In contrast, our system tailors' meals based on individualized health data and supports real-time interaction, feedback loops, and broader deployment potential.

The suggested remedy is an AI-driven adaptive meal recommendation system that combines ingredient inventories, predictive analytics, and user health data to create highly customized meal plans. The system's main component is a large language model (LLM) that can parse data entered by users, including weekly eating patterns, dietary restrictions, allergies, and pantry items. The system creates customized cost-friendly meal recommendations based on this input, taking into account dietary objectives and medical conditions.

This solution addresses two significant drawbacks of conventional approaches, which mainly rely on supplementation through pills or static dietary advice. First, it acknowledges that nutrient requirements differ based on body composition, lifestyle, and health goals, moving the emphasis from broad nutritional guidelines to tailored recommendations. Second, it gives food-based solutions precedence over synthetic ones, which maximizes nutrient absorption and takes bioavailability into account. For instance, the system might suggest salmon with fortified yogurt instead of a vitamin D pill, incorporating nutrients into a meal format that the body can better absorb.

The system is built to learn from user input and feedback over time. The application finds trends and provides increasingly detailed recommendations by monitoring dietary data over several weeks. Dynamic adaptation to shifting tastes, health conditions, or ingredient availability is ensured by this iterative process [4]. Furthermore, real-time nutritional analysis of branded food products is made possible by integration with external APIs like OpenFoodFacts, which increases transparency and accuracy [5].

This approach not only raises user awareness but also offers a more effective and long-lasting substitute for supplementation-based strategies.Theappencourages people to embrace evidencebased, healthier eating practices that are useful in everyday situations by providing them with information and practical suggestions.

Two experiments were conducted to evaluate blind spots in the proposed system. The first experiment tested reliable enforcement of dietary restrictions. Eleven user profiles were generated, each with a specific dietary need. Accuracy was measured across meals, yielding a mean score of 90.7 and a median of 90.0. Simple meals achieved perfect accuracy, while composite meals, such as tacos or mixed bowls, were more error-prone. The granola bar performed worst, and this is likely due to hidden allergens and ambiguous labeling. The findings demonstrated that ingredient complexity is the strongest factor affecting enforcement accuracy.

The second experiment assessed the accuracy of nutritional information generation for different ingredient categories. Meat consistently achieved 100% accuracy, whereas plant-based, grain, and composite foods occasionally missed micronutritional details [6]. This reflects broader challenges in food informatics, where macronutrients are easier to standardize than fluctuating micronutrients. Together, these experiments confirmed that the system is efficient overall, but it can still be improved and more accurately address multi-ingredient parsing, and micronutrient standardization.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Handling Food Classification Variability in Mealie

A key issue identified during the development of the Mealie application was the inconsistent classification of foods. For instance, for chicken breast, there are over 65,000 different options for such food. Each of these options contains differing amounts of preservatives, micro- and macronutrients, and fillers. When generating entire meals, these underlying varieties lead to huge differences in implicated health benefits. A multitude of ways to address this issue include manually specifying the amount of preservatives and fillers within a person’s food, or utilizing an API that contains a working inventory of the numerous versions of chicken breast, allowing a person to specify which brand and exact product they have – this leads to the application having an accurate internal frame that takes in all contexts. The ladder was applied (Pennington & Fisher, 2009).

2.2. Ensuring Data Consistency in Flutter with Firebase

During development, we encountered a challenge related to data consistency with flutter pages. In other words, pages that should contain data and new information that the user has input, do not display such data. Flutter builds widgets through a reactive widget tree, meaning changes are not always reflected in real time. To resolve this issue, we integrated Firebase as both a backend database and a state synchronization tool. By using firebase's real time storage and data authentication services, we were able to store information in the cloud and retrieve updates dynamically through providers and stream listeners. Firebase acts as a central storage of information to allow consistent page refreshment and accuracy (Google, 2024).

2.3. Ensuring JSON Output Consistency in AI-Driven Features

An additional challenge lies in the AI's expected output, which was a specified json formatted to match the app's data models. However, the llm's output was inconsistent with this format at times and would cause the feature relying on it to break. This was addressed by directly instructing the ai model to give the exact format by providing it a template, along with specified prompting that repeatedly reiterates a response until the accepted json format is provided. By acknowledging and fixing this issue, the system ensures every meal recommendation and nutrition generation is consistently accurate and will never result in UI/legibility issues for the user (Brown et al., 2020).

3. Solution

The program is a meal-planning application that guides users from the first launch to fully personalized meal generation. The flow begins at the splash screen and moves into the login screen for returning users, while new users are directed to the sign-up screen. New users then complete an onboarding sequence consisting of profile setup, initial ingredients setup, initial meals setup, and health profile setup. These steps collect the user's preferences, available ingredients, and dietary information, ensuring the system can generate accurate and personalized recommendations.

After onboarding or login, the user arrives at the main app navigation hub. From here, they can access key features including the home screen, the weekly meal plan, the health profile, and the ingredients section. The home screen provides personal recommendations and quick access to favorite meals. The weekly meal plan allows the user to view and generate tailored meal schedules, while the health profile stores dietary goals and restrictions. The ingredients section manages the user's pantry inventory to ensure meal generation aligns with what is available. If certain ingredients are not present, the AI model will attempt to provide a healthy meal with the higher number of aligned ingredients to the user's inventory [7].

The application is developed in Dart, Flutter, and Firebase, offering a cross-platform interface, and integrates AI-powered ingredient categorization to streamline data entry and ensure accurate classification [8]. This linear flow creates a smooth user experience, moving logically from initial setup to ongoing personalized meal planning, and is designed to be scalable for future features such as advanced nutrition tracking and recipe discovery (Nguyen &Vi, 2025; Majumder, 2025).

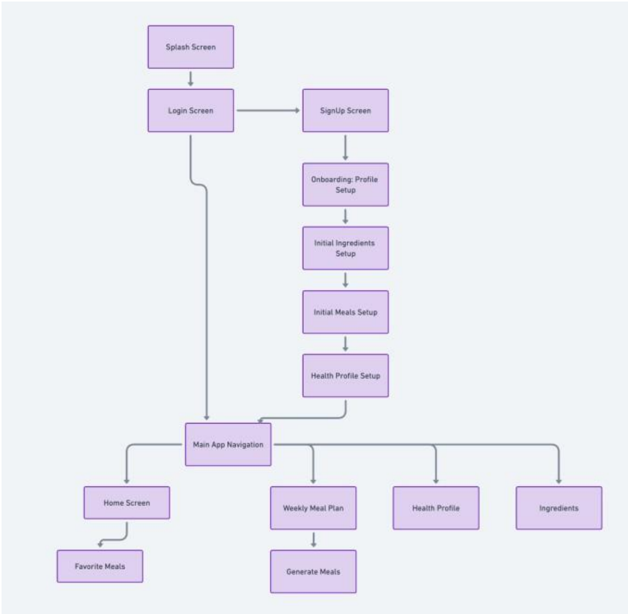

Figure 1. Overview of the solution

The MealRecommendationsScreen lets users select a meal type (breakfast, lunch, dinner, snack) and generate personalized meal ideas using an AI service [9]. It pulls user health profile and ingredient data from Riverpod providers, sends them to the backend for AI-powered recommendations, and displays the results. Users can view meal details, report inappropriate content, or add ingredients if needed. The screen is reactive, updating as user data or preferences change.

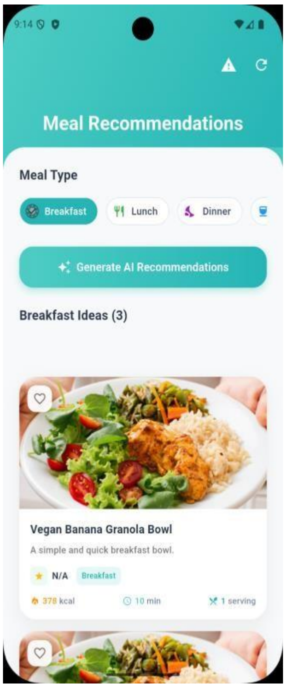

Figure 2. Screenshot of the recommendation

```dart
Future<List<Meal>> generateMealRecommendations(
  List<Ingredient> availableIngredients,
  HealthProfileModel healthProfile,
  MealType mealType,
) async {
  final ingredientsList = availableIngredients
      .map((i) => '${i.name} (${i.quantity}${i.unit})')
      .join(', ');

  AppLogger.debug('AIService: Generating meal recommendations for meal type: ${mealType.name}');
  final prompt = generateMealRecommendationsPrompt(
    ingredientsList,
    healthProfile,
    mealType,
  );

  try {
    final response = await _model.generateContent(
      [Content.text(prompt)],
      generationConfig: GenerationConfig(
        responseMimeType: 'application/json',
        responseSchema: _mealRecommendationSchema,
      ),
    );

    if (response.promptFeedback?.blockReason != null || response.text == null || response.text!.isEmpty) {
      AppLogger.warning(
        'AIService: Meal recommendation request was blocked or had no content.',
      );
      return [];
    }

    final List<Map<String, dynamic>> data = List<Map<String, dynamic>>.from(
      json.decode(response.text!) as List,
    );
    AppLogger.debug('AIService: Successfully generated meal recommendations.');

    return data.map((mealData) => _parseMeal(mealData, availableIngredients, mealType)).toList();
  } catch (e) {
    AppLogger.error('AIService: Error with meal recommendations: $e');
    return [];
  }
}
```

Figure 3. Screenshot of code 1

The generateMealRecommendations function manages the control flow for producing personalized meal suggestions within the application. It accepts three critical input parameters: a list of available ingredients, a user-specific health profile model, and a designated meal type. The function begins with an input validation step to ensure that all required parameters are present; if this condition is not met, the system transitions into an error state using AsyncValue.error. This in turn halts the execution and ensures no further parsing is done. In the case of valid inputs, the function updates the state to AsyncValue.loading, signaling to the user interface that processing is underway. Subsequently, the function initiates an asynchronous backend API request through _aiService, which internally calls the Gemini AI model [10]. Gemini performs intelligent meal generation by parsing the provided ingredients, applying nutritional and dietary constraints derived from the user's health profile, and generating contextually appropriate meal suggestions. If the request is successful, the resulting meal data is encapsulated within AsyncValue.data, allowing the front-end to reactively render the output. Conversely, if the API request fails, the function captures the exception and current stack trace, sets the state to AsyncValue.error, and logs the error for debugging purposes. This architecture supports robust error handling, maintains application stability, and enables seamless, reactive UI updates that are tightly integrated with asynchronous AI-driven logic (Doshi et al., 2023; Deng et al., 2024).

The IngredientsScreen lets users manage their pantry ingredients. It displays a searchable, filterable list of ingredients, allows adding, editing, and deleting items, and provides a nutritional summary (calories, protein, carbs, fat) for all ingredients. Users can filter by category, search by name, and clear all ingredients. The screen uses Riverpod for state management and modern UI elements for a smooth experience [11].

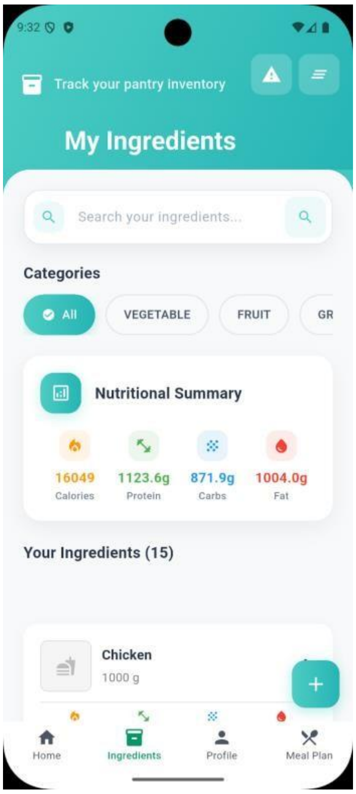

Figure 4. Screenshot of ingredients

```
Future<IngredientCategory?> generateIngredientCategory(
  String ingredientName,
) async {
  AppLogger.debug( 'AIService: Generating ingredient category for: $ingredientName',);
  final prompt = generateIngredientCategoryPrompt(ingredientName);

  try {
    final response = await _model.generateContent(
      [Content.text(prompt)],
      generationConfig: GenerationConfig(
        responseMimeType: 'application/json',
        responseSchema: Schema.object(
          properties: {
            'category': Schema.enumString(
              enumValues: IngredientCategory.values
                  .map((e) => e.name)
                  .toList(),
            ),
          },
        ),
      ),
    );

    final data = response.text;
    if (data != null) {
      final jsonMap = jsonDecode(data) as Map<String, dynamic>;
      final categoryString = jsonMap['category'] as String?;
      if (categoryString != null) {
        try {
          AppLogger.debug('AIService: Category generated for $ingredientName: $categoryString',);
          return IngredientCategory.values.firstWhere(
            (e) => e.name.toLowerCase() == categoryString.toLowerCase(),
          );
        } catch (e) {
          AppLogger.error(
            'AIService: Error parsing category string: $categoryString. Falling back to Other. Error: $e',
          );
          return IngredientCategory.OTHER;
        }
      }
    }
  } catch (e) {
    AppLogger.error('AIService: Error generating ingredient category for $ingredientName: $e',);
  }
  return IngredientCategory.OTHER;
}
```

Figure 5. Screenshot of code 2

The generate ingredient category function is a part of the AI service functionality in the application and works by first utilizing a hand crafted system prompt that is then fed into the model in order to get a correct output [12]. The prompt works by giving the ai model all of the previously set ingredient categories that are available in the application. In doing so, the function generates a category based on the ingredient name in the set Json format. The code then decodes the json category, and matches it to aenum list of values. This method balances adaptability with reliability. It is integrated in the addIngredient function to generate said category, and the categories are then used to filter ingredients in the ingredient screen. This category is also used to inform better predictive conclusions when generating meals.

The searchProducts function serves to power ingredient and product search within the system. It handles API communication, data parsing, error handling, and model conversion. This ensures that the rest of the app can work with reliable structured nutritional data. Its design supports both userdriven search and automated features like meal planning and recommendations (Achananuparp et al., 2018; Kuo et al., 2020).

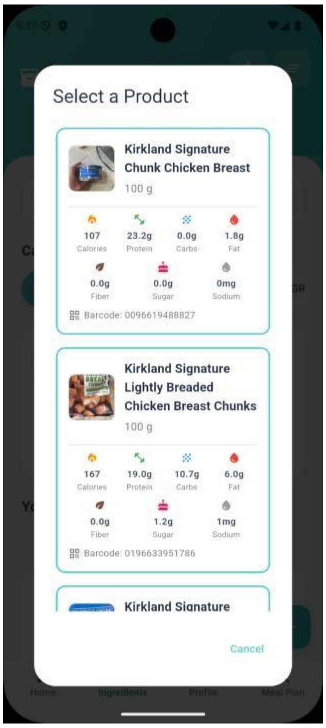

Figure 6. Screenshot of products

```
Future<List<Ingredient>> searchProducts(String query) async {
  AppLogger.debug('OpenFoodFactsService: Searching for query: $query');
  try {
    final uri = Uri.parse(
      '${AppConfig.openFoodFactsSearchUrl}?search_terms=$query&search_simple=1&action=process&json=1&page_size=${AppConfig.defaultSearchResults}&lc=en',
    );
    final response = await http.get(uri);

    AppLogger.debug('OpenFoodFactsService: Search response status: ${response.statusCode}');

    if (response.statusCode != 200) {
      return [];
    }

    final data = json.decode(response.body);
    final products = data['products'] as List? ?? [];

    final productsToUse = products.where((product) {
      final hasEnglishName = _hasEnglishName(product);
      if (hasEnglishName) return true;

      AppLogger.debug(
        'OpenFoodFactsService: Skipping product without English name: ${product['product_name']}',
      );
      return (product['product_name']?.toString().isNotEmpty ?? false) ||
            (product['generic_name']?.toString().isNotEmpty ?? false);
    }).toList();

    AppLogger.debug('OpenFoodFactsService: Found ${productsToUse.length} relevant products for query: $query');

    return productsToUse.map((product) {
      return Ingredient(
        name: _getBestProductName(product),
        barcode: product['code'],
        calories: _getNutrient(product, ['energy-kcal_100g', 'energy_100g', 'energy-kcal']),
        protein: _getNutrient(product, ['proteins_100g', 'protein_100g', 'protein']),
        carbs: _getNutrient(product, ['carbohydrates_100g', 'carbohydrate_100g', 'carbs_100g']),
        fat: _getNutrient(product, ['fat_100g', 'total-fat_100g', 'fats_100g']),
        fiber: _getNutrient(product, ['fiber_100g', 'fiber']),
        sugar: _getNutrient(product, ['sugars_100g', 'sugar_100g']),
        sodium: _getNutrient(product, ['sodium_100g', 'salt_100g']),
        imageUrl: product['image_front_url'],
        brand: product['brands'],
        saturatedFat: _parseNutriments(product, 'saturated-fat_100g'),
        cholesterol: _parseEstimatedNutriments(product, 'cholesterol_100g'),
        vitaminA: _parseEstimatedNutriments(product, 'vitamin-a_100g'),
        vitaminB12: _parseEstimatedNutriments(product, 'vitamin-b12_100g'),
        vitaminC: _parseEstimatedNutriments(product, 'vitamin-c_100g'),
        vitaminD: _parseEstimatedNutriments(product, 'vitamin-d_100g'),
        calcium: _parseEstimatedNutriments(product, 'calcium_100g'),
        iron: _parseEstimatedNutriments(product, 'iron_100g'),
        zinc: _parseEstimatedNutriments(product, 'zinc_100g'),
        magnesium: _parseEstimatedNutriments(product, 'magnesium_100g'),
        potassium: _parseEstimatedNutriments(product, 'potassium_100g'),
        quantity: 100,
        unit: 'g',
      );
    }).toList();
  } catch (e) {
    AppLogger.error('OpenFoodFactsService: Error searching for $query: $e');
    return [];
  }
}
```

Figure 7. Screenshot of code 3

The provided code defines the searchProducts service. When invoked, the method logs the search query and sends an HTTP GET request to the OpenFoodFacts search endpoint, including parameters for the search-term, result limit, and a preference for English-language results [13].

Upon receiving a successful response, it decodes the JSON and extracts the list of products [14]. The method then filters these products, first selecting those with English names using the _hasEnglishName helper. If none are found, it falls back to products with any available name. For each relevant product, it constructs an Ingredient object, using _getBestProductName to determine the most appropriate name. It also extracts values such as calories, carbohydrates, fat, fiber, sugar, and sodium. For each relevant product, it constructs an ingredient object, using _getBestProductName to determine the most appropriate name.

4. EXPERIMENT

4.1. Experiment 1

A key blind spot in my program is the reliable interpretation and enforcement of dietary restrictions when generating meal recommendations. While the app allows users to input preferences such as vegetarian, gluten-free, or nut-free, there’s a risk that ingredient parsing may not catch all relevant violations.

To test how accurately the app applies dietary restrictions, I will create 11 user profiles, each with a different restriction (e.g. vegetarian, gluten-free, dairy-free, nut-free, and low-sodium). With this, I will generate meals using our system, and I will assess the accuracy of the meal generation by three factors: if the meal follows their health profile, if it contains ingredients that the user has within their inventory, and if it follows the correct meal type (Breakfast, lunch, dinner). The experiment is set up in this format to gain an efficient understanding of how safe the app is for the average user, and if it will output meals that suits an individual's needs on a precise basis.

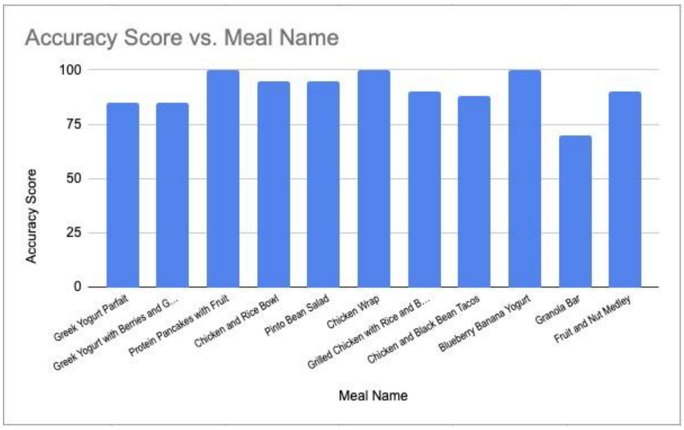

Figure 8. Figure of experiment 1

The evaluation of dietary restriction enforcement revealed strong overall performance. However, there is notable variation across meal types. Across the eleven meals tested, the mean accuracy score was approximately 90.7, and the median was 90.0, indicating dependable parsing capability for most items. Simple meals, such as Protein Pancakes with Fruit, Chicken Wrap, and Blueberry Banana Yogurt, achieved perfect accuracy (100.0), while the lowest score, 70.0, was recordedfor the Granola Bar, warranting closer investigation.

This discrepancy is consistent with existing findings: AI systems display diminished accuracy when parsing compound or inclusive ingredient lists. External research on LLMs confirms that decomposing compound ingredients remains a key challenge, as even leading models struggle with ambiguous or nested components, such as seasoning blends or sauce bases (Kopitar et al., 2024). Moreover, attention-based ingredient parsers, while achieving strong performance on structured recipe phrases, still encounter difficulties with highly variable, unlabeled input formats common in real-world data (~0.93 F1-score) (Shi et al., 2022).

Our findings reflect these patterns: meals featuring layered or composite ingredients (e.g., tacos, mixed bowls) yielded modestly lower accuracy, suggesting extended parsing complexity. The Granola Bar’ s low score likely stems from undetected allergens or multi-part elements. These insights underscore that ingredient complexity is the dominant factor limiting system accuracy and supports the pursuit of enhanced parsing approaches, particularly for multi-component foods.

4.2. Experiment 2

A potential blindspot within our system is the generated nutrition data of certain ingredients within our AI model, since when the user doesn't use the food facts api, we use ai to decide the nutrient composition for said ingredient.

To test this potential issue, we design an experiment where we test 14 different ingredients’ nutritional information generations via ai. We will score accuracy via a 1-5 scale, with a 5 reflecting a completely accurate generation of nutrients. These different ingredients cover the most significant categories of food, including meat/protein, snack, fruit, grain, and vegetable. Using these core labels, we can accurately assess the reliability of our system for every type of ingredient.

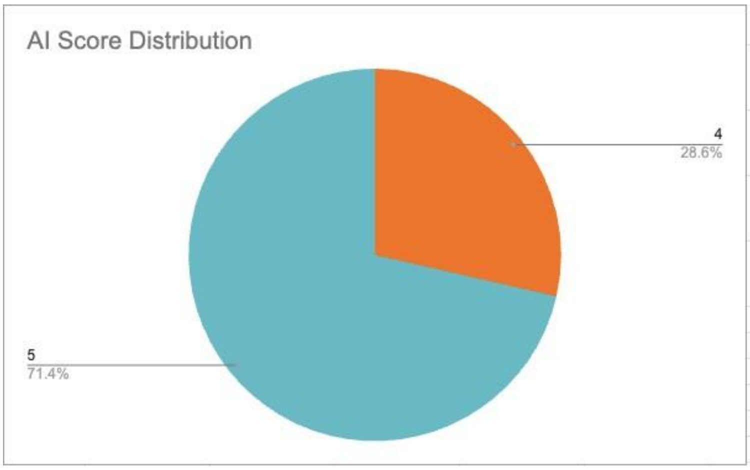

Figure 9. Figure of experiment 2

Via this graph, we see that meat is consistently 100% accurate. For other ingredient types, the model can sometimes miss certain micronutritional information. This is likely due to an inherent variability and heterogeneity of plant-based and composite foods. In essence, This means that the nutrient content for these foods are less uniform, and depend regularly on the composition of the food, the preparation method, and regional labeling standards. For example, micronutrients such as iron or vitamin C can fluctuate widely in vegetables depending on soil composition and storage conditions, which complicates reliable parsing. Prior work in food informatics has shown that NLP systems are more effective at identifying macronutrient categories than micronutrient details, especially when ingredients are represented in unstructured formats or include implicit subcomponents (Lee et al., 2020; Trattner&Elsweiler, 2019). These findings suggest that the model's blind spot stems not from lack of recognition for major food groups, but from the difficulty of standardizing micronutritional data in diverse ingredient contexts.

5. RELATED WORK

NutrifyAI (Han et al., 2024) combines computer vision (YOLOv8), nutritional analysis via Edamam APIs, and personalized meal recommendations [15]. It enables real-time food detection and provides immediate nutritional insights with approximately 80% accuracy. By contrast, the system proposed in this work surpasses NutrifyAI in several dimensions. First, it supports textbased and flexible ingredient input, rather than relying solely on image recognition, making it more adaptable to diverse contexts. Second, instead of offering general health insights, it enforces userspecific dietary constraints, such as allergies or medical needs, using real-time API filtering. Finally, while NutrifyAI delivers static recommendations, our system incorporates interactive user feedback loops, tracks nutritional compliance and satisfaction, and is designed for deployment across web and mobile platforms, with extensibility to wearable and electronic health systems.

To assess the practical applications and utility of our application, we compare it against MealRec (Zhou et al, 2022), a recent benchmark dataset bundled meal recommendation framework [16]. MealRec excels in the field of bundled recommendation by introducing a dataset of over 3,800 three-course meals, it remains limited in applicability to real world dietary recommendation tasks. MealRec's dataset was constructed using reviews from AllRecipies.com and identifying positively rated appetizer-main-desert trios under the assumption that co-rated items represent coherent bundles. Although this formulation supports the development of graph-based collaborative filtering models such as the proposed Category-Constrained Meal

Recommendation (CCMR) algorithm, the reliance on retrospective co-ratings poses limitations in modeling true dietary intent. Notably, the dataset does not confirm whether the meals were consumed together or within the same temporal context, nor does it accommodate non-Western or non-course-based meal structures. It also doesn't account for the potential fake reviews left to boost certain meal curations. Moreover, the classification of recipes into fixed categories is a reductive assumption, ignoring the flexibility and variability of real-world meal compositions.

Our system addresses these constraints through a fundamentally different approach. Rather than using pre-identified models, our system utilizes explicit user inputs, and assembles meals in real time. Users may specify ingredients, their sleep routine, height, weight, allergies, and dietary goals, and the system dynamically filters the ranked recipes using a combination of API-source meta data, and natural language processing.

MealRec+ (Li et al., 2024) introduces a dataset for evaluating meal recommendations with courselevel affiliation and healthiness scores, which are based on established nutritional guidelines [17]. However, there are some clear issues. It remains a static, simulation-based resource, lacking personalization, real-time input, or deployment capability.

In contrast, the system presented in this work is designed for dynamic, user-specific dietary guidance. It accepts explicit input on allergies, restrictions, and preferences, and uses nutritional APIs to enforce individualized health constraints during real-time meal assembly. Unlike MealRec+, which applies uniform health scores, our system tailors recommendations based on user-specific needs, such as sodium limits or glycemic control.

Furthermore, MealRec+ is confined to offline evaluation using ranking metrics like HR@10, whereas our system supports interactive feedback and measures adherence, satisfaction, and safety [18]. Its modular architecture enables deployment via web or mobile platforms, supporting integration with EHRs and wearable devices. Overall, this work advances beyond MealRec+ in personalization, adaptability, and real-world utility.

6. CONCLUSIONS

One major limitation of this project is the app’s dependency on external databases like OpenFoodFacts, which may lack comprehensive or up-to-date ingredient information for some products. This can lead to incomplete filtering for dietary restrictions. Additionally, the ingredient parsing algorithm still struggles with complex or ambiguous labeling, such as compound ingredients or regional food terms [19]. While the app uses Firebase to manage real-time updates and cross-page consistency, the backend does not yet support user-submitted corrections or custom ingredient tagging. Another limitation is the sample size testing; while initial testing with five dietary profiles revealed useful patterns, a broader test pool is needed to assess real-world accuracy and edge cases. With more time, I would implement machine learning classifiers trained specifically on food labels, add a community verification feature, and expand testing to include feedback from nutrition professionals and users with medical dietary needs.

This project demonstrates the potential of AI-driven meal recommendation systems to enhance health through personalization and reliable nutritional guidance [20]. While limitations remain in parsing complex foods and micronutrients, the experiments underscore a strong foundation. The system addresses multiple key issues other generative platforms fail to recognize. This system can serve as a scalable tool for everyday dietary decision-making and is destined to change lives for the better.

REFERENCES

[1] Monteiro, Carlos A., et al. "Ultra-processed foods: what they are and how to identify them." Public health nutrition 22.5 (2019): 936-941.

[2] Strong, Kathleen, et al. "Preventing chronic diseases: how many lives can we save?." The Lancet 366.9496 (2005): 1578-1582.

[3] Yabancı, Nurcan, İbrahim Kısaç, and Suzan ŞerenKarakuş. "The effects of mother's nutritional knowledge on attitudes and behaviors of children about nutrition." Procedia-Social and Behavioral Sciences 116 (2014): 44774481.

[4] Pennington, Jean AT, and Rachel A. Fisher. "Classification of fruits and vegetables." Journal of Food Composition and Analysis 22 (2009): S23-S31.

[5] Johnson, D. Gale. "Facts, Trends, and Issues of Open Markets and Food Security." Learning Workshop on Economic Analysis of Food Security Policy, conference of the International Association of Agricultural Economists, Berlin. 2000.

[6] Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.

[7] Team, Gemini, et al. "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context." arXiv preprint arXiv:2403.05530 (2024).

[8] Salord, Tristan, Marie-BenoîtMagrini, and Guillaume Cabanac. "Packaged foods with pulse ingredients in Europe: A dataset of text-mined product formulations." Data in Brief 42 (2022): 108173.

[9] Tran, N. R., et al. "Predicting food consumption using contextual factors: An application of machine learning models." European Journal of Public Health 34.Supplement_3 (2024): ckae144-296.

[10] Orue-Saiz, Iñigo, Miguel Kazarez, and Amaia Mendez-Zorrilla. "Systematic review of nutritional recommendation systems." Applied Sciences 11.24 (2021): 12069.

[11] Trattner, Christoph, and David Elsweiler. "Food recommender systems: important contributions, challenges and future research directions." arXiv preprint arXiv:1711.02760 (2017).

[12] Li, Ming, et al. "Mealrec+: A meal recommendation dataset with meal-course affiliation for personalization and healthiness." Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2024.

[13] Li, Ming, et al. "Mealrec: A meal recommendation dataset." arXiv preprint arXiv:2205.12133 (2022).

[14] Geng, Saibo, et al. "Grammar-constrained decoding for structured NLP tasks without finetuning." arXiv preprint arXiv:2305.13971 (2023).

[15] Shorten, Connor, et al. "Structuredrag: Json response formatting with large language models." arXiv preprint arXiv:2408.11061 (2024).

[16] Koo, Terry, Frederick Liu, and Luheng He. "Automata-based constraints for language model decoding." arXiv preprint arXiv:2407.08103 (2024).

[17] Geng, Saibo, et al. "Jsonschemabench: A rigorous benchmark of structured outputs for language models." arXiv preprint arXiv:2501.10868 (2025).

[18] Aung, SoeThandar, et al. "A study of learning environment for initiating Flutter app development using Docker." Information 15.4 (2024): 191.

[19] Younis, Manal Fadhil, and Zainab Salim Alwan. "Monitoring the performance of cloud real-time databases: A firebase case study." 2023 Al-Sadiq International Conference on Communication and Information Technology (AICCIT). IEEE, 2023.

[20] Hall, Kevin D., et al. "Ultra-processed diets cause excess calorie intake and weight gain: an inpatient randomized controlled trial of ad libitum food intake." Cell metabolism 30.1 (2019): 67-77.

©2025 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.