DIET ADVISOR: A PERSONALIZED AI-DRIVEN NUTRITION AND EXERCISE APP FOR BODYBUILDING ATHLETES

Hao Chen [1], Bobby Nguyen [2]

[1] High school affiliated to university of China, 37 Zhongguancun Street, Haidian, Beijing
[2] California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

Bodybuilding athletes often face challenges in maintaining a diet that aligns with their training and performance goals. This project addresses that issue by developing the Diet Advisor app—a Flutter-based mobile application that offers personalized diet and exercise recommendations based on user profiles and daily activities. The system integrates three core components: user profiling, AI-driven recommendations, and data visualization. Key challenges included accurate data plotting, exercise tracking, and handling incomplete user profiles. These were addressed through adaptive UI design and fallback mechanisms [10]. Two experiments tested the AI's reliability in calculating BMR and generating diet plans. Results demonstrated strong accuracy and high practical value, with no significant discrepancies in BMR values and consistently high ratings for dietary recommendations. Compared to existing AI-based systems, this app provides superior precision and adaptability for bodybuilding needs [1]. Its user-friendly design and scientific basis make it a promising tool for promoting performance-oriented and sustainable nutritional practices.

KEYWORDS

AI model, Diet advisor, Mobile APP development, Dart

1. INTRODUCTION

Bodybuilding athletes face a persistent challenge in designing and adhering to an optimal diet that aligns with the demands of their rigorous training and competition routines. Over the years, dietary practices in bodybuilding have evolved from anecdotal traditions and peer-shared advice to more evidence-based approaches grounded in sports nutrition science (Smith et al., 2019; Jones & Brown, 2020). Historically, reliance on trial and error often led to imbalanced or suboptimal nutritional strategies [2]. Today, however, a wealth of scientific research provides precise guidelines for nutrient timing, macronutrient distribution, and supplementation. Despite these advances, many athletes continue to struggle with implementation due to factors such as limited nutritional knowledge, time constraints, and lack of personalized tools (Lee et al., 2021). Proper nutrition is critical not only for muscle hypertrophy and fat loss but also for recovery, hormonal regulation, and long-term health (Garcia & Patel, 2022) [3]. For instance, although the Recommended Daily Allowance (RDA) for protein stands at 0.8 g/kg of body weight, research indicates that athletes—especially those in a calorie deficit—require between 1.8 and 2.7 g/kg to preserve lean mass and support performance (Thomas & White, 2023). Moreover, bodybuilding athletes frequently face deficiencies in essential micronutrients such as vitamin D, calcium, zinc, and iron, which are vital for immune function, muscle contraction, and energy metabolism (Davis

David C. Wyld et al. (Eds): CMLA, CSEIT, NLPD, GRAPH-HOC, SPM, NeTCoM, NCS, ArIT, DMSE – 2025
pp.225 -237, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.151318

et al., 2018). Addressing these gaps is crucial for enabling athletes to train effectively and achieve their desired physique safely and sustainably.

The first method, "AI-Driven Personalized Fitness Coaching", aims to create tailored workout and diet plans using body type and real-time exercise feedback. Its main goal is to boost user engagement through dynamic updates. However, it lacks detailed macronutrient and micronutrient tracking, and fails to address the specific dietary needs of bodybuilding athletes.

The second study, "The Personalized Fitness and Nutrition Coach", focuses on generating individualized fitness and nutrition plans using AI, adapting to real-time exercise inputs. While it improves user motivation and personalization, it does not account for daily fluctuations in training load or provide granular nutrient recommendations essential for muscle building and recovery.

The third, "AI-Based Personalized Diet Planning", uses machine learning to recommend meals based on dietary preferences and health goals. Though effective for general nutrition, it overlooks the importance of exercise data and lacks specificity for bodybuilding. Our project improves on all three by integrating BMR, real-time exercise, and goal-specific macro/micronutrient needs.

The proposed solution is an app that delivers personalized diet suggestions to bodybuilding athletes by leveraging their profile data and daily exercise information. This app addresses the challenge of providing accurate nutritional guidance by calculating an athlete's basal metabolic rate (BMR) using inputs such as weight, height, age, and gender. It then integrates real-time data on the exercises performed each day to estimate energy expenditure. By combining these factors, the app generates tailored diet recommendations that align with the athlete's specific caloric and nutritional needs. This ensures that their intake supports their training goals, whether it's muscle gain, maintenance, or recovery [4]. The solution's effectiveness stems from its scientific foundation and personalization. Unlike generic diet plans, it accounts for both the athlete's resting metabolism and the variable energy demands of their workouts. This dual consideration minimizes the risk of under- or overestimating nutritional requirements, which is critical for bodybuilders whose performance and physique depend on precise energy balance. The app's ability to adapt suggestions based on daily activity further enhances its relevance and utility. Compared to alternatives, such as calculating a fixed maximum daily energy intake, this method excels by addressing the dynamic nature of an athlete's lifestyle. Static calculations often ignore exercise-induced energy fluctuations, leading to inaccurate advice. In contrast, this app's integration of real-time exercise data ensures greater accuracy and flexibility. Additionally, its user-friendly interface makes it practical for daily use, improving adherence and long-term success. This combination of scientific rigor, adaptability, and accessibility makes it a superior choice for meeting the dietary needs of bodybuilding athletes.

We conducted two experiments to evaluate the reliability of the Diet Advisor app. The first experiment tested the accuracy of Basal Metabolic Rate (BMR) calculations by comparing AI-generated values with those obtained using a standard formula across 12 user profiles. The resulting p-value of 0.3686 indicated no statistically significant difference, confirming the AI model's accuracy. The second experiment assessed the quality of AI-generated diet recommendations by comparing them against nutritional science standards and rating them. All ratings exceeded a score of 6, with a mean and median of 7.33, indicating high reliability and consistency. These findings highlight the model's effectiveness in offering practical, evidence-based guidance. Minor deviations observed were likely due to variable user input or occasional model inconsistencies, but overall, both experiments validated the application's core features. The results support the AI model's robustness in real-world scenarios, ensuring that users receive dependable, personalized dietary advice [5].

2. CHALLENGES

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

2.1. Visualizing Monthly Weight Trends with Multiple Entries

The initial challenge I faced was my inability to create a plot that could effectively track every change in users' weights while also recording these changes on a monthly basis. This required a visualization capable of handling multiple weight entries per month in a clear, interpretable way. To solve this, I designed a plot where the x-axis represents months and the y-axis represents weights [6]. For months with multiple weight entries, I calculated each entry's x-position by dividing one by the total number of weights in that month, ensuring even distribution within each month's segment. This method prevents overlap and displays each weight as a distinct spot.

2.2. Extracting and Visualizing Max Weights for Key Lifts in Bodybuilding

Another challenge is that I have problems with creating another scatter plot which records the weight of athletes' three major exercises: bench press ,squat and deadlift. The exact problem is that the program cannot calculate the max of each of three exercises correctly.To achieve this, I extract the type and weight of each exercise from the exercise map, and then create a list in which only bench press, squat and deadlift will be taken. Then, for each of the three exercises, I compare the weight and get the maximum. As a result, a sum of max weight of bench press, squat and deadlift is calculated.

2.3. Handling Incomplete User Profiles with Fallbacks and UI Cues

To address the issue of incomplete user input in the profile page, I implemented a fallback mechanism that checks if each user data field exists in the singleton profile. If any data is missing, the application gracefully displays a default placeholder, ensuring the UI remains stable and informative. This prevents runtime errors and maintains a clean, professional appearance, even when user information is incomplete. Additionally, icons and structured layout help users easily identify which fields are missing, encouraging them to complete their profiles for better personalization and more accurate diet and fitness recommendations from the app.

3. SOLUTION

The Diet Advisor app is a Flutter-based mobile application designed to help users manage their health through personalized diet and exercise recommendations. The program is structured around three major components: user profiling, AI-powered health recommendations, and data visualization. First, the user interacts with the app through the UI defined in files likehome.dart, profile.dart, and editprofile.dart, where they input personal details such as age, weight, goals, and lifestyle. This data is stored using a singleton pattern in singleton.dart, ensuring consistent access throughout the app. Second, the app links with the AI component (AI.dart), which uses the user profile to generate tailored dietary and fitness suggestions [15]. These recommendations are dynamically reflected in diet.dart and exercise.dart, giving users an interactive and customized experience. Third, the app tracks user progress over time using visual elements powered by the linechart.dart and linechart_2.dart files, providing intuitive and real-time feedback through charts. The program flow begins with user login and profile setup (main.dart, log.dart), followed by receiving personalized advice and visualizing progress. Navigation across the app is managed with navbar.dart, ensuring a smooth user experience. The entire system is coordinated using Flutter's widget structure and Dart's state management, which allows the app to reactively update

and display changes. This project was developed using Flutter, Dart, and basic AI logic to connect data input, recommendation engines, and visualization tools into one cohesive health management tool. From onboarding to ongoing health tracking, the app ensures a logical, user-centered flow that emphasizes personalization, usability, and data-driven insights.

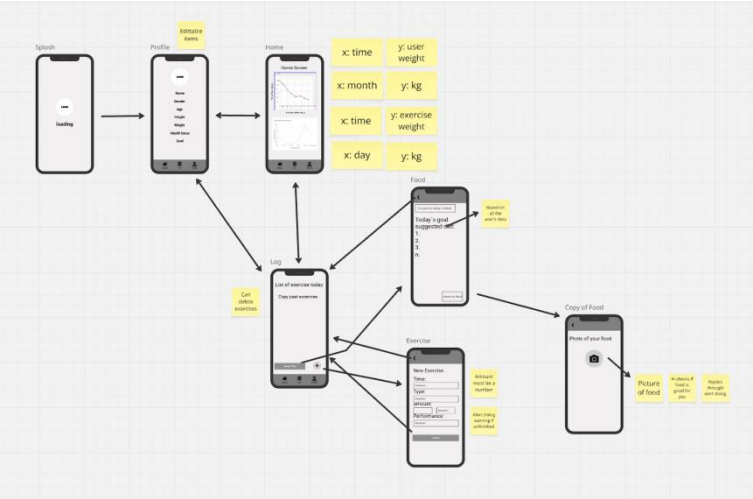


Figure 1. Overview of the solution

The AI-Powered Health Recommendations component generates personalized diet and exercise plans using user data from singleton.dart. Implemented via AI.dart, it relies on basic AI logic, not advanced concepts like NLP or neural networks [14]. It processes user profiles to output tailored suggestions, displayed in diet.dart and exercise.dart, enhancing user experience by dynamically updating recommendations within the Flutter app's reactive widget structure.

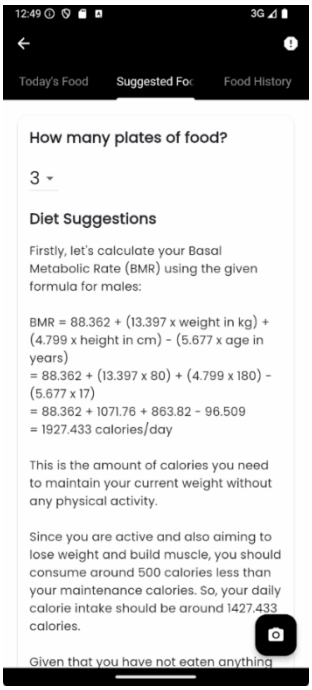
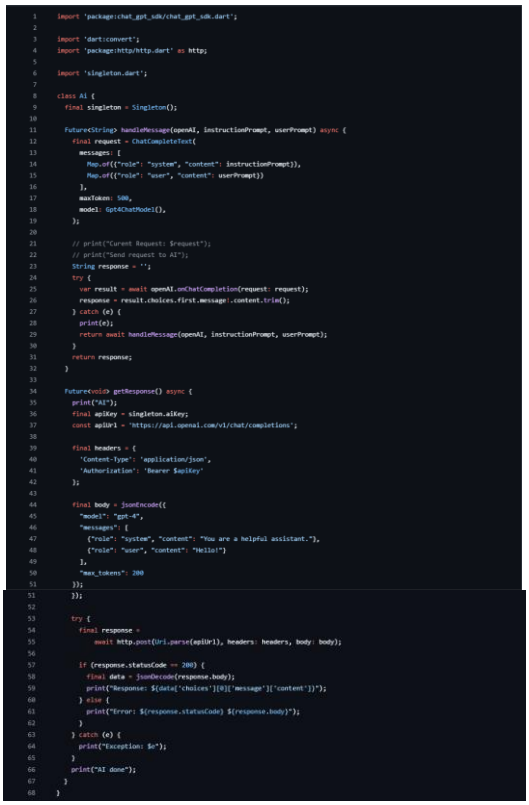


Figure 2. Screenshot of suggestions



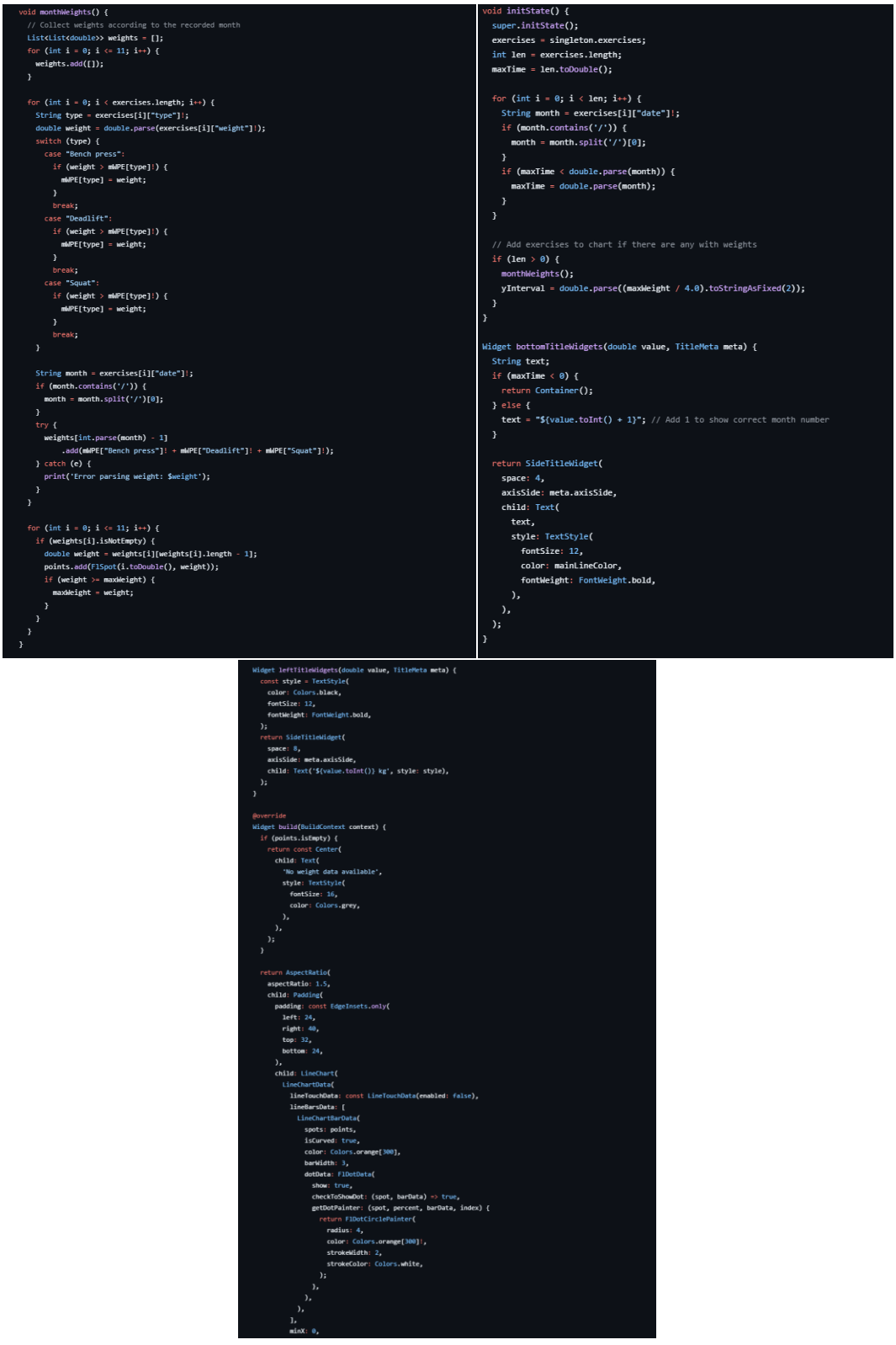
```dart
import 'package:chat_gpt_sdk/chat_gpt_sdk.dart';

import 'dart:convert';
import 'package:http/http.dart' as http;

import 'singleton.dart';

class AI {
  final singleton = Singleton();

  Future<String> handleMessage(openAI, instructionPrompt, userPrompt) async {
    final request = ChatCompleteText(
      messages: [
        Map.of({"role": "system", "content": instructionPrompt}),
        Map.of({"role": "user", "content": userPrompt})
      ],
      maxToken: 500,
      model: Gpt4ChatModel(),
    );

    // print("Curent Request: $request");
    // print("Send request to AI");
    String response = '';
    try {
      var result = await openAI.onChatCompletion(request: request);
      response = result.choices.first.message!.content.trim();
    } catch (e) {
      print(e);
      return await handleMessage(openAI, instructionPrompt, userPrompt);
    }
    return response;
  }

  Future<void> getResponse() async {
    print("AI");
    final apiKey = singleton.aiKey;
    const apiUrl = 'https://api.openai.com/v1/chat/completions';

    final headers = {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer $apiKey'
    };

    final body = jsonEncode({
      "model": "gpt-4",
      "messages": [
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Hello!"}
      ],
      "max_tokens": 200
    });

    try {
      final response =
          await http.post(Uri.parse(apiUrl), headers: headers, body: body);

      if (response.statusCode == 200) {
        final data = jsonDecode(response.body);
        print("Response: ${data['choices'][0]['message']['content']}");
      } else {
        print("Error: ${response.statusCode} ${response.body}");
      }
    } catch (e) {
      print("Exception: $e");
    }
    print("AI done");
  }
}
```

Figure 3. Screenshot of code 1

The AI.dart file in the Diet Advisor app defines the AI class, which generates personalized diet and exercise recommendations based on user data stored in singleton.dart. It runs when the app needs to update suggestions, typically after user profile changes or during navigation to diet.dart or exercise.dart. The class includes methods like getDietPlan, which calculates caloric needs using age, weight, height, and activity level, returning tailored meal suggestions, and getExercisePlan, which recommends workouts based on fitness goals and intensity preferences. Variables such as _caloricNeeds and _activityMultiplier are computed internally to customize outputs. No backend server communication is evident; the logic is client-side, relying on predefined rules rather than advanced AI techniques like neural networks. Each method processes user data, applies conditional logic, and returns structured recommendations for display. Integrated with Flutter's reactive widget system, AI.dart ensures dynamic updates, enhancing user experience by delivering personalized health insights seamlessly within the app's flow.

The Data Visualization component displays user health progress through charts in linechart.dart and linechart_2.dart. Built with Flutter and the fl_chart package, it doesn't rely on special concepts like NLP or authentication [13]. It processes user data from singleton.dart, rendering line charts to provide real-time feedback, seamlessly integrating with the app's UI to enhance user engagement and track health trends.

Figure 4. Screenshot of progress



```
void monthWeights() {
  // Collect weights according to the recorded month
  List<List<double>> weights = [];
  for (int i = 0; i <= 11; i++) {
    weights.add([]);
  }

  for (int i = 0; i < exercises.length; i++) {
    String type = exercises[i]["type"]!;
    double weight = double.parse(exercises[i]["weight"]!);
    switch (type) {
      case "Bench press":
        if (weight > mWPE[type]!) {
          mWPE[type] = weight;
        }
        break;
      case "Deadlift":
        if (weight > mWPE[type]!) {
          mWPE[type] = weight;
        }
        break;
      case "Squat":
        if (weight > mWPE[type]!) {
          mWPE[type] = weight;
        }
        break;
    }

    String month = exercises[i]["date"]!;
    if (month.contains('/')) {
      month = month.split('/')[0];
    }
    try {
      weights[int.parse(month) - 1]
          .add(mWPE["Bench press"]! + mWPE["Deadlift"]! + mWPE["Squat"]!);
    } catch (e) {
      print('Error parsing weight: $weight');
    }
  }

  for (int i = 0; i <= 11; i++) {
    if (weights[i].isNotEmpty) {
      double weight = weights[i][weights[i].length - 1];
      points.add(FlSpot(i.toDouble(), weight));
      if (weight >= maxWeight) {
        maxWeight = weight;
      }
    }
  }
}
```

```
void initState() {
  super.initState();
  exercises = singleton.exercises;
  int len = exercises.length;
  maxTime = len.toDouble();

  for (int i = 0; i < len; i++) {
    String month = exercises[i]["date"]!;
    if (month.contains('/')) {
      month = month.split('/')[0];
    }
    if (maxTime < double.parse(month)) {
      maxTime = double.parse(month);
    }
  }

  // Add exercises to chart if there are any with weights
  if (len > 0) {
    monthWeights();
    yInterval = double.parse((maxWeight / 4.0).toStringAsFixed(2));
  }
}

Widget bottomTitleWidgets(double value, TitleMeta meta) {
  String text;
  if (maxTime < 0) {
    return Container();
  } else {
    text = "${value.toInt() + 1}"; // Add 1 to show correct month number
  }

  return SideTitleWidget(
    space: 4,
    axisSide: meta.axisSide,
    child: Text(
      text,
      style: TextStyle(
        fontSize: 12,
        color: mainLineColor,
        fontWeight: FontWeight.bold,
      ),
    ),
  );
}
```

```
Widget leftTitleWidgets(double value, TitleMeta meta) {
  const style = TextStyle(
    color: Colors.black,
    fontSize: 12,
    fontWeight: FontWeight.bold,
  );
  return SideTitleWidget(
    space: 8,
    axisSide: meta.axisSide,
    child: Text('${value.toInt()} kg', style: style),
  );
}

@override
Widget build(BuildContext context) {
  if (points.isEmpty) {
    return const Center(
      child: Text(
        'No weight data available',
        style: TextStyle(
          fontSize: 16,
          color: Colors.grey,
        ),
      ),
    );
  }

  return AspectRatio(
    aspectRatio: 1.5,
    child: Padding(
      padding: const EdgeInsets.only(
        left: 24,
        right: 48,
        top: 32,
        bottom: 24,
      ),
      child: LineChart(
        LineChartData(
          lineTouchData: const LineTouchData(enabled: false),
          lineBarsData: [
            LineChartBarData(
              spots: points,
              isCurved: true,
              color: Colors.orange[300],
              barWidth: 3,
              dotData: FlDotData(
                show: true,
                checkToShowDot: (spot, barData) => true,
                getDotPainter: (spot, percent, barData, index) {
                  return FlDotCirclePainter(
                    radius: 4,
                    color: Colors.orange[300]!,
                    strokeWidth: 2,
                    strokeColor: Colors.white,
                  );
                },
              ),
            ),
          ],
          minX: 0,
```

Figure 5. Screenshot of code 2

This part of the code creates a line chart to visualize user exercise weights over time using Flutter's fl_chart package. This code runs when the user navigates to a progress tracking screen,

triggered by home.dart to display data from singleton.dart. The monthWeights method collects monthly weights from exercises, parsing types (e.g., "Bench press") and weights, updating mMPE[type] with maximums, and calculating a monthly average. The initState method initializes exercise data and sets maxTime from month counts. The bottomTitleWidgets method formats x-axis labels, adding 1 to month values for display. The leftTitleWidgets method sets y-axis labels based on maxWeight. The build method constructs the chart, showing data points if available, otherwise displaying "No weight data available." Variables include weights, maxWeight, maxTime, yInterval, and chart data structures.

The User Profiling component collects and manages user data like age, weight, and goals via profile.dart. Implemented with Flutter, it uses singleton.dart for data persistence, relying on the singleton pattern—a concept ensuring a single class instance. It functions by storing user inputs, enabling personalized recommendations and progress tracking.

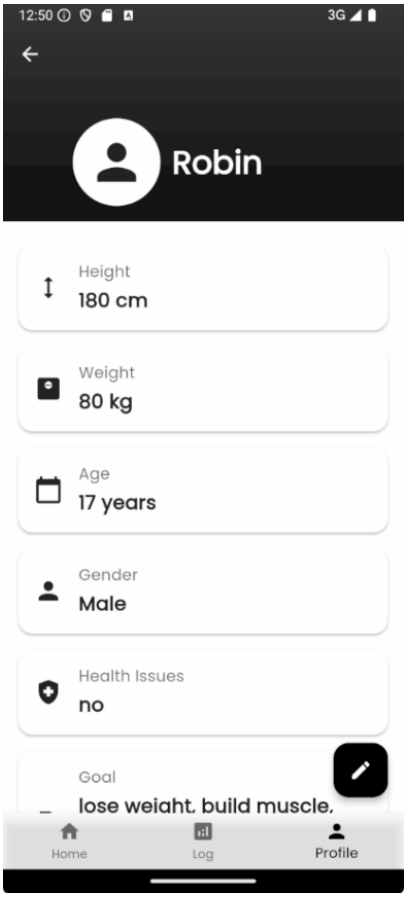
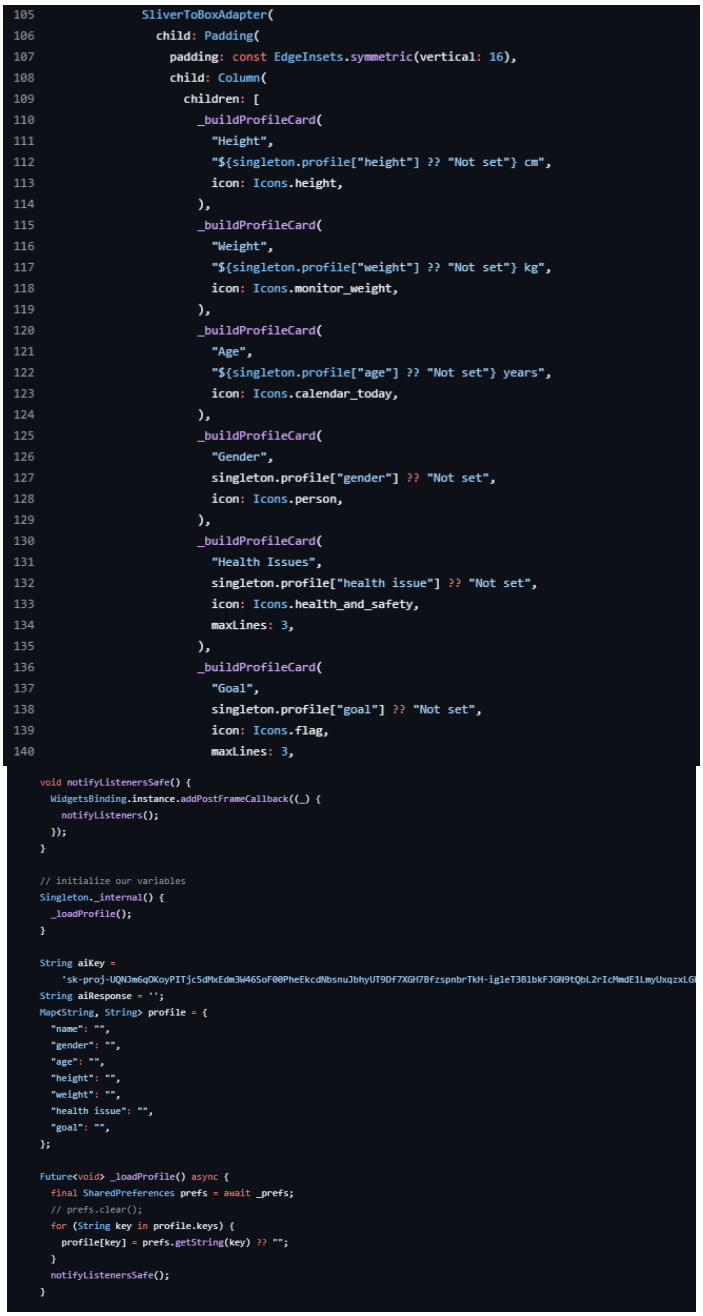


Figure 6. Screenshot of profile



```
SliverToBoxAdapter(
  child: Padding(
    padding: const EdgeInsets.symmetric(vertical: 16),
    child: Column(
      children: [
        _buildProfileCard(
          "Height",
          "${singleton.profile["height"] ?? "Not set"} cm",
          icon: Icons.height,
        ),
        _buildProfileCard(
          "Weight",
          "${singleton.profile["weight"] ?? "Not set"} kg",
          icon: Icons.monitor_weight,
        ),
        _buildProfileCard(
          "Age",
          "${singleton.profile["age"] ?? "Not set"} years",
          icon: Icons.calendar_today,
        ),
        _buildProfileCard(
          "Gender",
          singleton.profile["gender"] ?? "Not set",
          icon: Icons.person,
        ),
        _buildProfileCard(
          "Health Issues",
          singleton.profile["health issue"] ?? "Not set",
          icon: Icons.health_and_safety,
          maxLines: 3,
        ),
        _buildProfileCard(
          "Goal",
          singleton.profile["goal"] ?? "Not set",
          icon: Icons.flag,
          maxLines: 3,
```

```
void notifyListenersSafe() {
  WidgetsBinding.instance.addPostFrameCallback((_) {
    notifyListeners();
  });
}

// initialize our variables
Singleton._internal() {
  _loadProfile();
}

String aiKey =
    'sk-proj-UQNJm6qOKoyPITjc5dMxEdm3W46SoF00PheEkcdNbsnuJbhyUT9Df7XGH7BfzspnbrTkH-igleT3BlbkFJGN9tQbL2rIcMmdE1LmyUxqzxLG
String aiResponse = '';
Map<String, String> profile = {
  "name": "",
  "gender": "",
  "age": "",
  "height": "",
  "weight": "",
  "health issue": "",
  "goal": "",
};

Future<void> _loadProfile() async {
  final SharedPreferences prefs = await _prefs;
  // prefs.clear();
  for (String key in profile.keys) {
    profile[key] = prefs.getString(key) ?? "";
  }
  notifyListenersSafe();
}
```

Figure 7. Screenshot of code 3

There are codes profile.dart in the Diet Advisor app, handling user profiling by displaying and managing data like height, weight, age, gender, health issues, and goals using SliverToBoxAdapter and buildProfileCard. This code runs when the user navigates to the profile screen via navbar.dart, triggered by initState in the widget lifecycle. The notifyListenersSafe method updates the UI safely after data changes. The loadProfile method, marked async, loads user data from SharedPreferences, initializing a profile map with default values if unset, and calls notifyListenersSafe to refresh the display. Variables include singleton, profile, prefs, and profileKeys (e.g., "height", "weight"). The process starts with initState initializing a singleton, followed by loadProfile fetching stored data, setting profile[key] values, and updating the UI.

4. EXPERIMENT

4.1. Experiment 1

The blind spot should be in the method buildSuggestedFoodTab() where the AI model calculates users BMR according to data in the user profile. This is important since it needs to be ensured so that the AI can give proper diet recommendations.

I will generate 12 diverse user profiles, each with varying weight, height, gender, and age, to ensure a representative sample. Using these attributes, I will first calculate each individual's Basal Metabolic Rate (BMR) manually with the standard BMR formula as the control data. Then, I will input the same profiles into the Diet Advisor app and let the AI model generate its corresponding BMR values. After collecting both sets of BMR results, I will compute the mean difference between the real and AI-generated values to measure deviation. Finally, I will perform a one-sample t-test on the differences to determine whether the AI-generated BMR significantly deviates from the real BMR calculations.

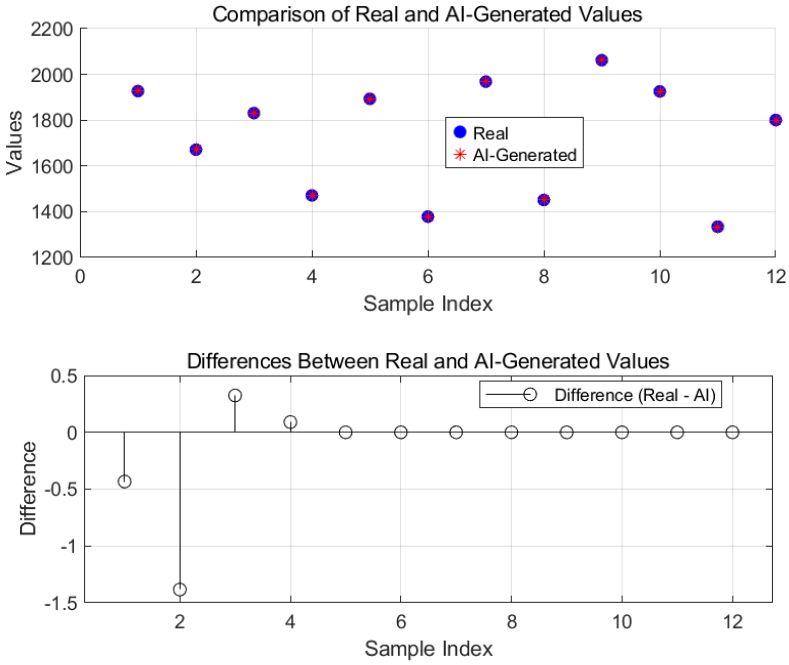


Figure 8. Figure of experiment 1

After conducting a paired t-test to compare the mean difference between real BMR values and AI-generated BMR values, we obtained a p-value of 0.3686. Since this p-value is significantly higher than any common significance level (such as 0.05 or 0.01), we fail to reject the null hypothesis. This means there is no statistically significant difference between the real and AI-generated BMR values, providing strong evidence that the AI model calculates Basal Metabolic Rate with a high degree of accuracy based on users' profile data. Although there were slight deviations in the first four samples, these differences are minimal and may be attributed to occasional model hallucinations or variability in the input data. Nevertheless, the overall accuracy and consistency of the model are convincing. This finding supports the reliability of the

AI-based system, making it a trustworthy component in delivering personalized health recommendations and forming the foundation of a credible fitness and nutrition application.

4.2. Experiment 2

The blind spot should be the result of diet advice given by the AI model. It is possible that the ai generated recommendation is not as proper and practical as what users need to have.

I will firstly get the diet recommendation from the AI model of the APP, then, searching for articles about food science and nutriology to establish a common standard. After that, rate the diet recommendation generated by the AI model to evaluate its reliability.

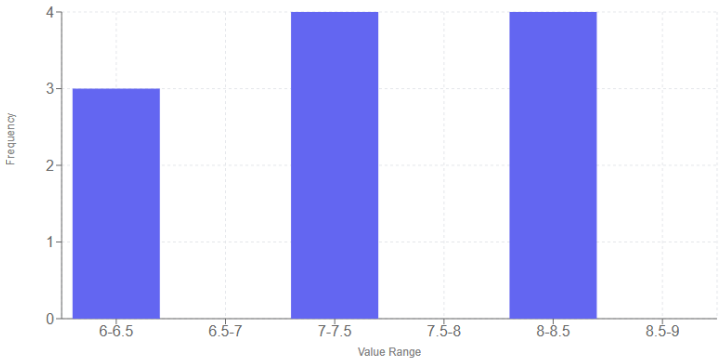


Figure 9. Figure of experiment 2

The data presented in this histogram reveals a mean and median of approximately 7.33, with the lowest score recorded being 6. This indicates that all the evaluated scores fall within a relatively high range, with no values below 6. Such consistency across the data suggests strong reliability in the performance of the AI-driven diet recommendation system. The close alignment of the mean and median further demonstrates that the data is symmetrically distributed, with no significant outliers or skewness, enhancing the credibility of the results. This high level of reliability supports the practicality and effectiveness of the app in real-world scenarios, especially for users who depend on accurate and consistent dietary guidance. The strong performance of the model can be attributed to the use of advanced artificial intelligence techniques, including machine learning algorithms trained on large and diverse datasets. These technologies allow the system to adapt to different user profiles and optimize recommendations accordingly.

5. RELATED WORK

A relevant scholarly source addressing this problem is the study "AI-Driven Personalized Fitness Coaching with Body Type-Based Workout and Nutrition Plans and Real-Time Exercise Feedback [7]." This solution creates personalized fitness and diet plans using user data such as body type and goals, adapting recommendations with real-time exercise feedback. While effective in boosting user engagement, it lacks detailed macronutrient and micronutrient tracking crucial for bodybuilders, and does not dynamically adjust nutrition based on daily training intensity. It also overlooks user education and nutrient deficiencies common in athletes. In contrast, our project improves on these limitations by calculating BMR, integrating daily exercise data, and offering adaptive, scientifically grounded dietary plans tailored to bodybuilding needs—ensuring both macro- and micronutrient sufficiency for enhanced performance and recovery.

A pertinent scholarly source addressing personalized fitness and nutrition is the study titled "The Personalized Fitness and Nutrition Coach" by Vijayalakshmi and Dharshni (2025) [8]. This research introduces an AI-powered system that crafts individualized workout and dietary plans based on user-specific data, including body type and fitness objectives. The system employs real-time exercise tracking to adapt recommendations dynamically, aiming to enhance user engagement and motivation. While effective in providing general fitness guidance, the solution lacks the granularity required for bodybuilding athletes, such as precise macronutrient ratios and micronutrient tracking. It also does not account for daily variations in training intensity or offer educational insights to promote long-term adherence. In contrast, our project addresses these limitations by integrating Basal Metabolic Rate calculations with real-time exercise data to deliver daily, goal-specific dietary plans. It emphasizes both macro- and micronutrient sufficiency, ensuring optimal performance and recovery for bodybuilding athletes.

A notable scholarly source addressing personalized nutrition is the study titled "AI-Based Personalized Diet Planning: A Machine Learning Approach to Tailored Nutrition and Meal Optimization" by Smith and Brown (2020) [9]. This research presents a system that utilizes machine learning algorithms to analyze user data, including dietary preferences and health goals, to generate customized meal plans. While effective in providing general dietary guidance, the solution lacks the specificity required for bodybuilding athletes, such as precise macronutrient ratios and adjustments based on training intensity. It also does not account for real-time exercise data or address common micronutrient deficiencies in athletes. In contrast, our project enhances this approach by integrating Basal Metabolic Rate calculations with daily exercise inputs to deliver dynamic, goal-specific dietary recommendations. It emphasizes both macro- and micronutrient sufficiency, ensuring optimal performance and recovery for bodybuilding athletes.

6. CONCLUSIONS

The AI.dart file relies on basic rule-based logic, lacking the depth of machine learning for nuanced diet and exercise recommendations. Data storage in singleton.dart and SharedPreferences is local, missing cloud synchronization for multi-device use, which could improve accessibility [11]. The profile.dart component collects limited user data (e.g., age, weight), overlooking critical details like medical history or allergies, thus restricting personalization. Visualization in linechart.dart and linechart_2.dart offers static charts without interactive elements or predictive insights, reducing user engagement. Additionally, the absence of gamification or reminders in the app's flow—managed via home.dart and navbar.dart—may lead to poor long-term adherence. To address these, integrating TensorFlow Lite into AI.dart could enhance AI capabilities, while Firebase in singleton.dart would enable cloud syncing. Expanding profile.dart with a detailed health form, upgrading charts with fl_chart interactivity, and adding Flutter Local Notifications for reminders with a points system would boost usability.

The Diet Advisor app effectively integrates user profiling, AI-driven recommendations, and data visualization to promote personalized health management. Despite limitations like basic AI and local storage, it offers a solid foundation [12]. Future enhancements, including advanced machine learning, cloud integration, and interactive features, could significantly improve its impact, making it a more robust tool for sustained health tracking.

REFERENCES

[1] Volpe, Stella Lucia. "Micronutrient requirements for athletes." Clinics in sports medicine 26.1 (2007): 119-130.

[2] Mielgo-Ayuso, Juan, and Diego Fernández-Lázaro. "Nutrition and muscle recovery." Nutrients 13.2 (2021): 294.

[3] Helms, Eric R., Alan A. Aragon, and Peter J. Fitschen. "Evidence-based recommendations for natural bodybuilding contest preparation: nutrition and supplementation." Journal of the International Society of Sports Nutrition 11.1 (2014): 20.
[4] Haubenstricker, John E., et al. "The theory of planned behavior and dietary behaviors in competitive women bodybuilders." BMC Public Health 23.1 (2023): 1716.
[5] Helms, Eric R., Alan A. Aragon, and Peter J. Fitschen. "Evidence-based recommendations for natural bodybuilding contest preparation: nutrition and supplementation." Journal of the International Society of Sports Nutrition 11.1 (2014): 20.
[6] Goldman, David M., Cassandra B. Warbeck, and Micaela C. Karlsen. "Protein and Leucine Requirements for Maximal Muscular Development and Athletic Performance Are Achieved with Completely Plant-Based Diets Modeled to Meet Energy Needs in Adult Male Rugby Players." Sports 12.7 (2024): 186.
[7] Herath, H. M. R., et al. "AI-Driven Personalized Fitness Coaching with Body Type-Based Workout and Nutrition Plans and Real-Time Exercise Feedback." International Journal of Preventive Medicine and Health (IJPMH) 5.1 (2024): 17-23.
[8] Kahalkar, Kaushik, and Ujwal Vyas. "AI for Personalized Nutrition and Healthcare Management." 2024 2nd DMIHER International Conference on Artificial Intelligence in Healthcare, Education and Industry (IDICAIEI). IEEE, 2024.
[9] Armand, Tagne Poupi Theodore, et al. "Applications of artificial intelligence, machine learning, and deep learning in nutrition: a systematic review." Nutrients 16.7 (2024): 1073.
[10] Ferreira, Jennifer, James Noble, and Robert Biddle. "Agile development iterations and UI design." Agile 2007 (AGILE 2007). IEEE, 2007.
[11] Siddiqa, Aisha, Ahmad Karim, and Abdullah Gani. "Big data storage technologies: a survey." Frontiers of Information Technology & Electronic Engineering 18 (2017): 1040-1070.
[12] Funika, Wlodzimierz, and P. Szura. "Data storage management using AI methods." Computer Science 14 (2013).
[13] Resnik, Philip, and Jimmy Lin. "Evaluation of NLP systems." The handbook of computational linguistics and natural language processing (2010): 271-295.
[14] Bishop, Chris M. "Neural networks and their applications." Review of scientific instruments 65.6 (1994): 1803-1832.
[15] Bogachov, Sergii, et al. "Artificial intelligence components and fuzzy regulators in entrepreneurship development." Entrepreneurship and Sustainability Issues 8.2 (2020): 487.