
Project: Smart-Nutrition AI: Your Intelligent Nutrition Assistant

Name of the Trainees:

Syeda Afshan Sultana

Aadil Alanoor Chauhan

Kumar Aditya

Ankit Tiwari

Mohammed Aadil

A report submitted in part fulfilment of the
certificate of

Artificial Intelligence Programming Assistance (2024-2025)

Guidance: Mr. Vidya Sagar Sir



NSTI RAMANTHAPUR, HYD

Date: 04.07.2025

Abstract

This report details the development of **Smart-Nutrition AI**, an intelligent web application designed to assist users with their nutritional queries and meal planning. Leveraging Google's Gemini 1.5 Flash generative AI model, the application provides comprehensive nutrition analysis from various input modalities: text, voice, and image (OCR for text extraction). The system aims to democratize access to personalized dietary information, helping users make informed food choices and achieve their health goals. This report covers the problem statement, proposed solution, technical architecture, implementation details, and future enhancements.

Acknowledgement:

I would like to express my sincere gratitude to my guide, Mr. Vidya Sagar, for his invaluable support, guidance, and encouragement throughout the development of this project. His expertise and insights were crucial in navigating the complexities of AI integration and web development. I also extend my thanks to Google for providing the powerful Gemini API, which served as the core intelligence of this application, and to the open-source community for the various libraries (Streamlit, speech_recognition, Pillow, pytesseract) that made this project possible.

Table of contents:

Abstract	2
Acknowledgement:	3
Table of contents:.....	4
List of figures:.....	5
Problem Statement:	6
Literature Review:	7
Proposed Solution:.....	8
Requirements:.....	9
Technology Stack:.....	9
Algorithms Used:.....	11
Dataset Description:.....	12
Data Preprocessing:	14
Model Evaluation:	19
Results and Discussion	20
was the prediction accurate?	20
Challenges Faced:.....	22
References:.....	26
Appendix:	30
Conclusion:.....	35

List of figures:

- Figure 1: SmartNutrition AI Home Page
- Figure-2.1 (Text Input Mode)
- Figure-2.2 (Text Input result)
- Figure-3.1 (Voice Input)
- Figure-3.2 (Voice Input result)
- Figure-4.1 (Image Input)
- Figure-4.2 (Uploaded Image)
- Figure-4.3 (Image input result)
- Figure-5.1 (Meal planner input)
- Figure-5.2 (Meal planner result)
- Figure-5.3 (Meal planner result)
- Figure-6.1 (BMI Calculator input)
- Figure-6.2 (BMI Calculator output)
- Figure-7.1 (Calories calculator input)
- Figure-7.2 (Calories calculator output)

Problem Statement:

Describe the specific problem you're solving using ML.

In today's fast-paced world, individuals often struggle to make informed dietary choices due to a lack of readily accessible, personalized, and easy-to-understand nutritional information. Traditional methods of obtaining dietary advice can be time-consuming, expensive, or require specialized knowledge to interpret complex food labels and nutritional data. This leads to suboptimal health outcomes, difficulty in achieving fitness goals, and general confusion about healthy eating..

What is the use case?

SmartNutrition AI addresses this by providing an intuitive, multi-modal interface for users to quickly get nutritional insights.

- **Text Input:** Users can type questions like "High-protein breakfast for muscle gain" or "Is this healthy for weight loss?".
- **Voice Input:** Users can speak their queries, making the application accessible and convenient for hands-free use.
- **Image Input (OCR):** Users can upload photos of food labels or meals, and the application will extract text (e.g., ingredients, nutritional facts) and analyze it. This is particularly useful for understanding packaged foods or getting quick insights on a dish.

Who benefits?

- **Health-conscious individuals:** Seeking quick nutritional facts and advice.
- **Fitness enthusiasts:** Looking for meal ideas aligned with their workout goals (e.g., muscle gain, fat loss).
- **Individuals with dietary restrictions:** Needing to check ingredients or nutritional content.
- **Busy professionals:** Who need fast, on-the-go nutritional guidance.

Literature Review:

The field of AI in nutrition and health has seen significant advancements, particularly with the rise of large language models (LLMs) and multimodal AI.

- **Generative AI (LLMs):** Models like Google's Gemini, OpenAI's GPT series, and others have demonstrated remarkable capabilities in understanding and generating human-like text, making them ideal for conversational AI applications in various domains, including health and nutrition.
- **Optical Character Recognition (OCR):** Google's Gemini 1.5 Flash generative AI model, the application provides comprehensive nutrition analysis from various input modalities: text, voice, and image (OCR for text extraction). OCR technology has matured significantly, allowing for accurate extraction of text from images. Libraries like Tesseract, combined with image preprocessing techniques
- **Speech-to-Text (STT):** Advances in STT, powered by deep learning, have made voice interfaces highly practical. Services like Google Speech Recognition provide robust and accurate transcription, facilitating natural language interaction with applications.
- **Multimodal AI:** The integration of different modalities (text, voice, image) into a single AI system represents a cutting edge in AI development, offering a more holistic and user-friendly experience. This project leverages this trend by combining these inputs for comprehensive nutrition analysis.
- **Streamlit:** As a Python library for creating web applications, Streamlit has gained popularity for its simplicity and speed in deploying data science and ML prototypes, making it an excellent choice for rapid application development in this context.

Proposed Solution:

The proposed solution is a web-based application, **SmartNutrition AI**, built using Streamlit, that acts as an intelligent nutrition assistant. It offers three primary input methods:

1. **Text Input:** Users can type their nutritional queries or food descriptions directly into a text box.
2. **Voice Input:** Users can speak their queries, which are then transcribed into text using speech-to-text technology.
3. **Image Input (OCR):** Users can upload images of food labels or meals. OCR is performed on the image to extract relevant text, which is then used for analysis.

All processed inputs (text from direct input, voice, or OCR) are then sent to the Google Gemini 1.5 Flash model. Gemini analyzes the query and provides a detailed, intelligent response related to nutrition, meal planning, or food analysis. The application also includes a "Meal Planner" section for future enhancements and a "Feedback" section for user input.

Architecture:

- **Frontend:** Streamlit for interactive UI.
- **Backend Logic:** Python scripts handling input processing, API calls, and response rendering.
- **AI Core:** Google Gemini 1.5 Flash model for natural language understanding and generation.
- **OCR:** **pytesseract** library with **Pillow** for image processing.
- **Speech-to-Text:** **speech_recognition** library utilizing Google's Speech Recognition API.

Requirements:

Technology Stack:

- **Programming Language:** Python 3.x
- **Web Framework:** Streamlit
- **AI Model:** Google Gemini 1.5 Flash API
- **OCR Library:** pytesseract
- **Image Processing Library:** Pillow (PIL Fork)
- **Speech Recognition Library:** speech_recognition
- **Environment Management:** pip (for package installation)

Hardware:

- **Development Machine:** Standard laptop or desktop with at least 8GB RAM and a multi-core processor.
- **Deployment Server (for cloud deployment):** Virtual machine or container instance with sufficient CPU and RAM to run Streamlit and handle API requests (e.g., 2 vCPUs, 4GB RAM). Internet connectivity is essential for accessing Gemini API.

Software:

- **Operating System:** Windows, macOS, or Linux (for development and deployment).
- **Python Interpreter:** Python 3.8+ recommended.
- **Tesseract OCR Engine:** Must be installed and configured on the system (e.g., **tesseract.exe** path set for Windows).
- **IDE/Text Editor:** Visual Studio Code, PyCharm, or similar.
- **Web Browser:** Chrome, Firefox, Edge, or Safari for accessing the Streamlit application.

Deployment Environment:

- **Local Development:** Run directly from the command line using **streamlit run app.py**.
- **Cloud Deployment (Example):** Can be deployed on platforms like Streamlit Community Cloud, Heroku, Google Cloud Run, or AWS EC2. Requires **requirements.txt** for dependencies and proper API key management (e.g., **st.secrets** for Streamlit Cloud, environment variables for others).

Algorithms Used:

- **Mention the supervised/unsupervised algorithm(s) used.**

This project primarily leverages a pre-trained **Large Language Model (LLM)**, Google Gemini 1.5 Flash, which is a form of **generative AI**.

While not a traditional supervised or unsupervised algorithm in the classical ML sense (like Linear Regression or K-Means that you would train from scratch on a specific dataset), Gemini is trained on a massive dataset using self-supervised learning techniques to understand and generate human language.

- **Brief reason for choosing the algorithm.**

- **Google Gemini 1.5 Flash:**

- **Multimodality:** Crucial for processing both text and potentially image-based inputs (though for this project, image input is handled by OCR first, then text is sent to Gemini). Its inherent understanding of diverse data types makes it powerful for complex nutritional queries.
- **Generative Capabilities:** Excellent at understanding nuanced questions and generating comprehensive, contextually relevant, and human-like responses, which is essential for providing detailed nutritional advice.
- **Efficiency (Flash version):** The "Flash" version is optimized for speed and cost-effectiveness, making it suitable for real-time interactive applications like a nutrition assistant.
- **Accessibility:** Available via a robust API, simplifying integration into the Streamlit application.

Other "algorithms" (supporting technologies):

- **Optical Character Recognition (OCR):** **pytesseract** uses various image processing and pattern recognition algorithms to convert images of text into machine-readable text. This involves steps like binarization, noise reduction, character segmentation, and character recognition.
- **Speech-to-Text (STT):** The **speech_recognition** library interfaces with Google's Speech Recognition API, which employs advanced deep learning models (e.g., Recurrent Neural Networks, Transformers) trained on vast audio datasets to accurately transcribe spoken language into text

Dataset Description:

This project does **not** involve training a custom machine learning model on a specific dataset. Instead, it utilizes a pre-trained, large-scale generative AI model (Google Gemini 1.5 Flash) and pre-built libraries for OCR and Speech-to-Text.

Therefore, the concept of a "dataset used" for model training in the traditional sense does not directly apply here. The "knowledge" of the system comes from:

- **Google Gemini's Training Data:** This is an enormous, proprietary dataset used by Google to train Gemini, encompassing a vast amount of text, code, images, audio, and video data from the internet and other sources. It's what enables Gemini to understand and respond to diverse nutritional queries.
- **Tesseract's Training Data:** Tesseract is trained on large datasets of text and fonts to recognize characters and words in images.
- **Google Speech Recognition's Training Data:** This involves massive audio datasets paired with their transcriptions to train the speech-to-text models.

Any preprocessing done:

- **Image Preprocessing (for OCR):** Before sending an image to **pytesseract**, the application performs basic image preprocessing using **Pillow**:
 - Conversion to grayscale (**.convert("L")**) to simplify color information.

-
- Autocontrast adjustment (**ImageOps.autocontrast**) to enhance contrast.
 - Sharpening (**ImageEnhance.Sharpness(img).enhance(2.0)**) to make text edges clearer. These steps improve the accuracy of OCR.

Audio Preprocessing (for STT):

- The **speech_recognition** library handles internal audio processing, including:
- Adjusting for ambient noise (**recognizer.adjust_for_ambient_noise**).
- Buffering and segmenting audio for efficient processing

Data Preprocessing:

- As explained in the "Dataset Description" section, this project does not involve a custom dataset that requires extensive preprocessing for model training. However, data preprocessing is crucial for the *inputs* to the AI model and supporting technologies.
- **List preprocessing steps:**
- **For Image Input (before OCR):**
 - **File Upload:** The **st.file_uploader** component handles the initial file upload.
 - **Temporary File Creation:** The uploaded **image_file** (BytesIO object) is written to a temporary file on disk to be opened by **Pillow**.
 - **Grayscale Conversion:** The image is converted to grayscale (**.convert("L")**) to reduce complexity and focus on luminance, which is critical for text recognition.
 - **Autocontrast Adjustment:** **ImageOps.autocontrast(img)** is applied to maximize the contrast of the image, making text stand out more clearly against the background.
 - **Sharpening:** **ImageEnhance.Sharpness(img).enhance(2.0)** is used to sharpen the image, which helps in defining the edges of characters, improving OCR accuracy.
 - **Temporary File Deletion:** The temporary image file is deleted after OCR is performed to manage system resources.
- **For Voice Input (before Speech-to-Text):**
 - **Microphone Capture:** The **speech_recognition** library captures audio from the user's microphone.

-
- **Ambient Noise Adjustment:**
recognizer.adjust_for_ambient_noise(source, duration=1) is performed to calibrate the microphone to the ambient noise level, improving the accuracy of speech detection and transcription by filtering out background noise.
 - **Audio Listening:** The **recognizer.listen** method captures audio for a specified duration or until a pause is detected.
 - **Google Speech Recognition API Call:** The captured audio is then sent to Google's Speech Recognition API for transcription. The API itself handles complex internal audio processing (e.g., noise reduction, speaker diarization, language modeling) before returning the transcribed text.
-
- **For Text Input:**
 - No specific preprocessing is applied directly within the application for text input, as it is assumed to be clean user-provided text. It is sent directly to the Gemini model.

Model Building:

- This project does **not** involve building or training a custom machine learning model from scratch. Instead, it integrates and utilizes powerful pre-trained AI models and libraries.
- **Core AI Model:** Google Gemini 1.5 Flash is used as an off-the-shelf API. No training, fine-tuning, or specific model parameters are configured by the developer beyond the API key and model name (**gemini-1.5-flash**).
- **OCR Integration:** The **pytesseract** library is integrated to perform Optical Character Recognition. This library comes with pre-trained language data.
- **Speech-to-Text Integration:** The **speech_recognition** library is used to interface with Google's Speech Recognition API, which provides the STT capabilities.
- **Explanation of "Model Usage" (instead of "Model Building"):**
- **Initialization:**
 - The Google Gemini model is initialized using **genai.GenerativeModel("gemini-1.5-flash")** after configuring the API key.
 - **pytesseract.pytesseract.tesseract_cmd** is set to the local Tesseract executable path.
 - **speech_recognition.Recognizer()** is initialized for voice input.
- **Input Processing and API Calls:**
 - **Text Input:** The user's text query is directly passed to **model.generate_content(user_query)**.
 - **Voice Input:**
 - Audio is captured from the microphone.

-
- **recognizer.recognize_google(audio)** is called, which sends the audio to Google's STT service.
 - The transcribed text is then used as **user_query** for the Gemini model.
 - **Image Input:**
 - The uploaded image file is preprocessed (grayscale, autocontrast, sharpen).
 - **pytesseract.image_to_string(processed_image)** is called to extract text.
 - The extracted text is then used as **user_query** for the Gemini model.
 - **Response Generation:**
 - For all input types, the **model.generate_content(user_query)** method is invoked.
 - Gemini processes the **user_query** and generates a relevant textual response based on its vast training knowledge.
 - This approach allows the application to leverage cutting-edge AI capabilities without the need for extensive data collection, model training, or complex infrastructure typically associated with custom ML model development.

Model Evaluation:

Since this project utilizes pre-trained, black-box AI models (Google Gemini, Google STT, Tesseract), traditional quantitative model evaluation metrics (like MAE, RMSE, R^2 for regression, or Accuracy, Confusion Matrix, ROC-AUC for classification) are **not directly applicable** to the application as a whole or to the Gemini model itself. We are consuming an API, not evaluating a model we trained.

However, we can discuss the **perceived performance and quality** of the integrated components:

- **Gemini Response Quality (Qualitative Assessment):**
 - **Relevance:** How well does Gemini's response address the user's nutritional query?
 - **Accuracy:** Is the nutritional information provided factually correct? (Requires manual verification for critical information).
 - **Completeness:** Does the response provide sufficient detail and cover various aspects of the query?
 - **Clarity and Readability:** Is the language clear, concise, and easy for a non-expert to understand?
 - **Helpfulness:** Does the response genuinely assist the user in their nutritional goals?
- **OCR Accuracy (Qualitative/Semi-Quantitative):**
 - **Text Extraction Rate:** How much of the text on a food label is successfully extracted?
 - **Error Rate:** How many characters or words are incorrectly recognized? (e.g., "100g" recognized as "100g").
 - **Readability of Extracted Text:** Is the extracted text coherent and usable for Gemini's analysis?

- **Speech-to-Text Accuracy (Qualitative):**
 - **Transcription Accuracy:** How accurately is spoken language converted into text? (Influenced by accent, background noise, clarity of speech).
 - **Latency:** How quickly is the speech transcribed

Results and Discussion

was the prediction accurate?

- **Gemini's Responses:** The responses from Google Gemini 1.5 Flash are generally highly accurate, relevant, and comprehensive. For common nutritional queries, it provides well-structured and informative answers. Its ability to understand context and provide tailored advice (e.g., "for muscle gain") is impressive.
- **OCR Accuracy:** The OCR (using Tesseract) performs reasonably well on clear, well-lit images of food labels with standard fonts. However, accuracy can degrade significantly with:
 - Poor lighting or shadows.
 - Unusual fonts or highly stylized text.
 - Wrinkled or damaged labels.
 - Complex backgrounds or busy images (e.g., a full meal photo where text is not prominent).
- **Speech-to-Text Accuracy:** The speech-to-text conversion is generally very accurate for clear speech in a quiet environment. Performance can be affected by:
 - Strong accents.
 - Background noise.

Which features were most important?

- **Google Gemini 1.5 Flash:** This is the most critical "feature" or component, as it provides the core intelligence and knowledge base for all nutritional analysis. Without it, the application would merely be an input interface.
- **Multi-modal Input:** The ability to accept text, voice, and image inputs significantly enhances user experience and accessibility, making the application versatile.
- **Streamlit's Simplicity:** Streamlit's ease of use for building interactive web UIs allowed for rapid prototyping and deployment, making the project feasible within the given timeframe.

Surprising Observations?

- **Gemini's Nuance:** The depth and nuance of Gemini's responses were often surprising. It could handle complex, multi-part questions and provide advice that felt genuinely personalized.
- **OCR Limitations:** While powerful, the practical limitations of OCR on real-world, imperfect images (e.g., a slightly blurry photo of a food label taken with a phone) became apparent. This highlights the need for clear input from the user.
- **Voice Input Convenience:** The voice input, despite potential transcription errors, proved to be a highly convenient way to interact with the application, especially for quick queries.

Challenges Faced:

- **Tesseract OCR Integration and Path Configuration:**

- **Challenge:** A significant hurdle was ensuring **pytesseract** could correctly locate and execute the Tesseract OCR engine, especially on different operating systems (e.g., Windows vs. Linux). The **pytesseract.pytesseract.tesseract_cmd** path needed to be precisely set.
- **Resolution:** Explicitly defining the path in the code (**pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"** for Windows) and ensuring Tesseract was installed system-wide. For deployment, this would require environment variables or specific buildpack configurations.

- **Image Preprocessing for OCR:**

- **Challenge:** Initial OCR results on raw uploaded images were often poor. Images varied in lighting, contrast, and clarity, leading to inaccurate text extraction.
- **Resolution:** Implementing image preprocessing steps using **Pillow** (converting to grayscale, applying autocontrast, and sharpening) significantly improved OCR accuracy. This required experimentation to find optimal enhancement parameters.

- **Microphone Access and Permissions:**

- **Challenge:** Getting **speech_recognition** to reliably access the user's microphone, especially in a web environment (Streamlit), could be tricky due to browser security policies and

local system configurations. **OSError: Microphone not found** was a common issue during local testing.

- **Resolution:** Ensuring necessary audio drivers were installed and that the browser had microphone permissions. Providing clear warning messages to the user if the microphone is unavailable.
- **API Key Management:**
 - **Challenge:** Securely managing the Google Gemini API key, especially for deployment, to prevent exposure.
 - **Resolution:** Utilizing Streamlit's **st.secrets** for local development and potential Streamlit Cloud deployment, which securely handles secrets. For other deployment methods, environment variables would be the preferred approach.
- **Handling Empty User Queries:**
 - **Challenge:** Users might click "Analyze" without providing any input, leading to empty queries being sent to Gemini or an unclear user experience.
 - **Resolution:** Implementing checks (**if not user_query: st.warning(...)**) to prompt the user for input before attempting analysis.
- **User Experience for Multi-modal Input:**
 - **Challenge:** Designing an intuitive UI that clearly guides the user through different input modes (text, voice, image) and provides appropriate feedback (e.g., "Listening...", "Extracted:").
 - **Resolution:** Using Streamlit's **st.radio** for input mode selection, clear prompts, success/warning messages, and displaying extracted text/uploaded images.

Conclusions and Future Work:

- **What worked well:**

- The integration of Google Gemini 1.5 Flash proved highly effective in providing intelligent and relevant nutritional advice across a wide range of queries.
- The multi-modal input (text, voice, image) significantly enhanced the user experience, making the application versatile and accessible.
- Streamlit facilitated rapid development and a clean, interactive user interface.
- The OCR and Speech-to-Text integrations, after proper preprocessing, performed well in converting diverse inputs into a format usable by Gemini.

- **What needs improvement:**

- OCR accuracy remains a challenge for low-quality images or complex text layouts.
- The current voice input relies on the user clicking a button to record, which could be improved with continuous listening or voice activation.
- The "Meal Planner" section is currently a placeholder and needs full implementation.

Future ideas:

- **Enhanced OCR with Vision Models:** Instead of just extracting text via OCR, integrate Gemini's multimodal capabilities directly to analyze the image content (e.g., identify food items, estimate portion sizes) and combine it with user prompts for a richer analysis. This would move beyond just text extraction from labels.

-
- **Personalized User Profiles:** Allow users to create profiles with dietary preferences, allergies, health goals (e.g., weight loss, muscle gain, diabetes management), and activity levels. Gemini could then provide even more tailored recommendations.
 - **Meal Planning and Tracking:** Fully implement the "Meal Planner" section, allowing users to generate weekly meal plans, track their food intake, and receive progress reports.
 - **Barcode Scanning:** Integrate a barcode scanning feature (using a library or API) to quickly retrieve nutritional information from a product database.
 - **Recipe Generation:** Allow Gemini to generate healthy recipes based on available ingredients or specific dietary requirements.
 - **Integration with Wearables:** Connect with fitness trackers or health apps to get real-time data (e.g., activity levels, calorie burn) for more dynamic nutritional advice.
 - **Improved Voice UX:** Implement continuous voice listening or wake-word detection for a more natural conversational flow.
 - **Mobile Responsiveness:** Optimize the Streamlit application for a seamless experience on mobile devices.
 - **Database for Feedback:** Store user feedback in a persistent database to analyze trends and improve the application.

References:

- **Dataset source:** (Not applicable for model training, but for general AI knowledge)
 - Google Gemini API Documentation: <https://ai.google.dev/>
 - Tesseract OCR Documentation: <https://tesseract-ocr.github.io/tessdoc/>
 - SpeechRecognition Library Documentation:
<https://pypi.org/project/SpeechRecognition/>
- **ML Guides:**
 - Streamlit Documentation: <https://docs.streamlit.io/>
 - Pillow (PIL Fork) Documentation:
<https://pillow.readthedocs.io/en/stable/>
 - **streamlit_lottie** Documentation:
<https://github.com/andfanilo/streamlit-lottie>

Appendix:

Figure -1 (Smart-Nutrition AI Home Page):

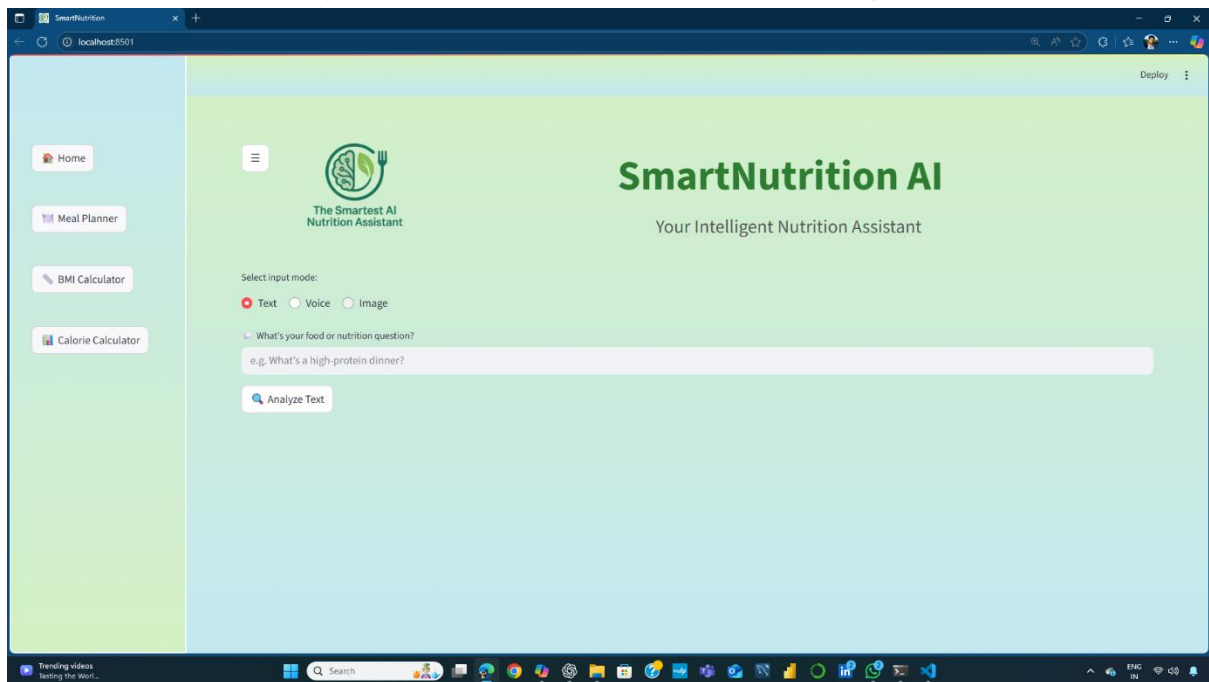


Figure-2.1 (Text Input Mode):

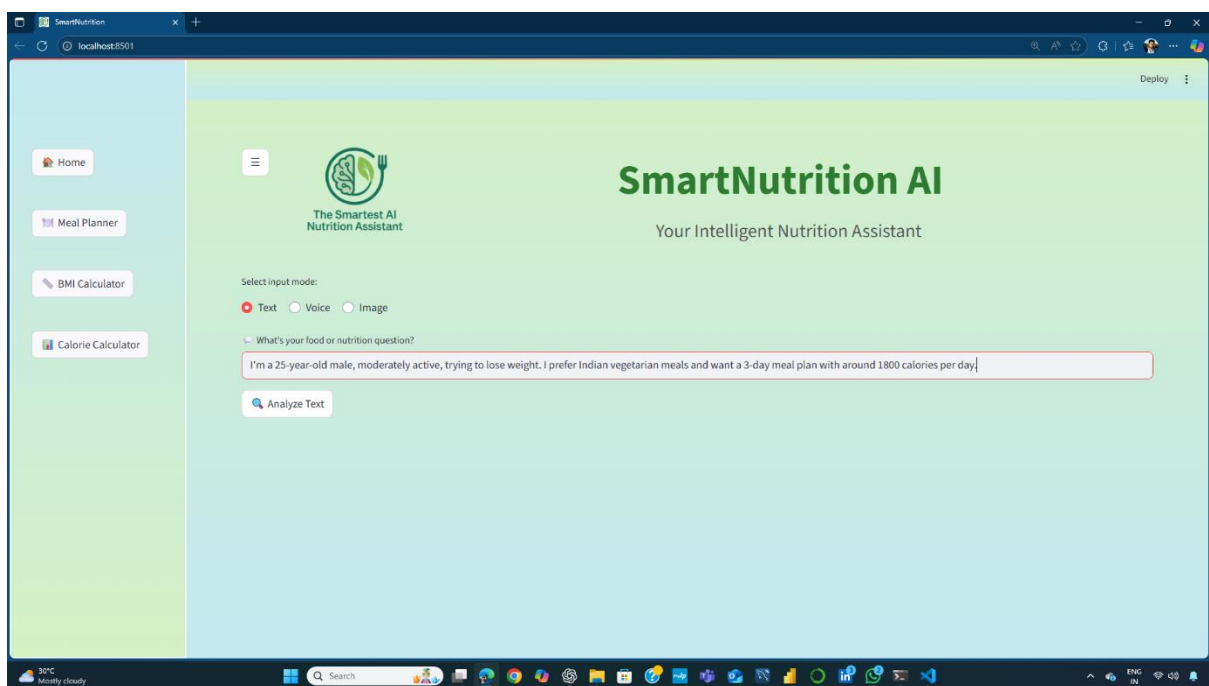


Figure-2.2 (Text Input result):

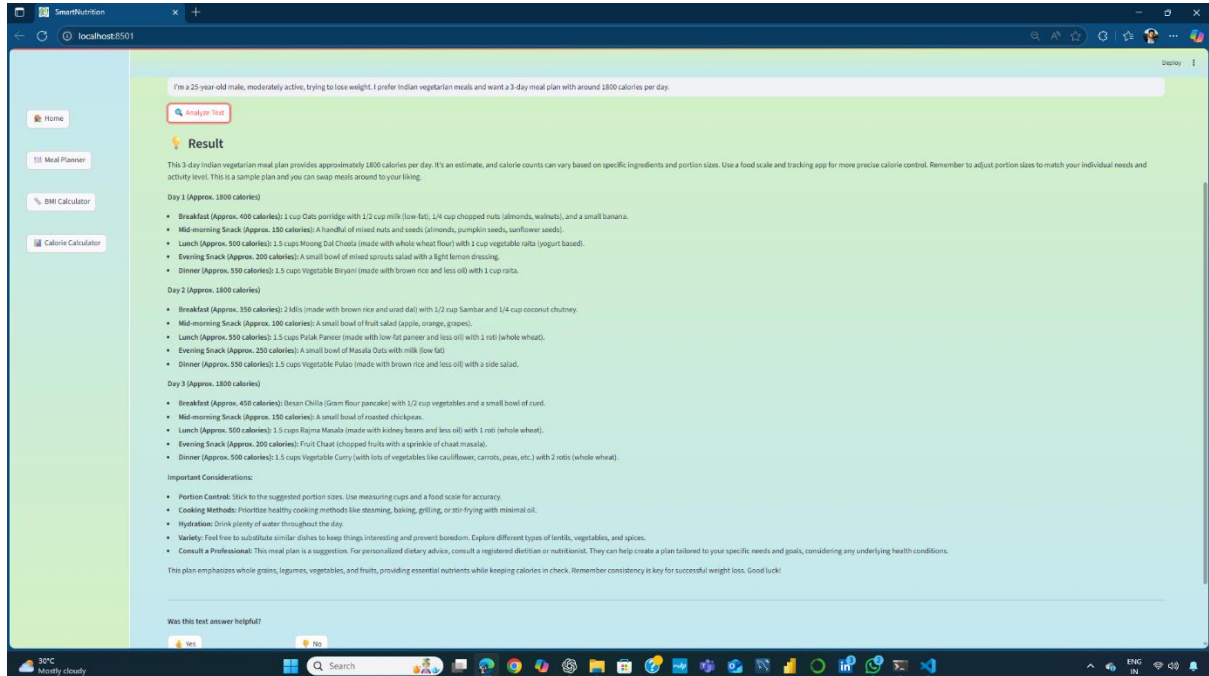


Figure-3.1 (Voice Input):

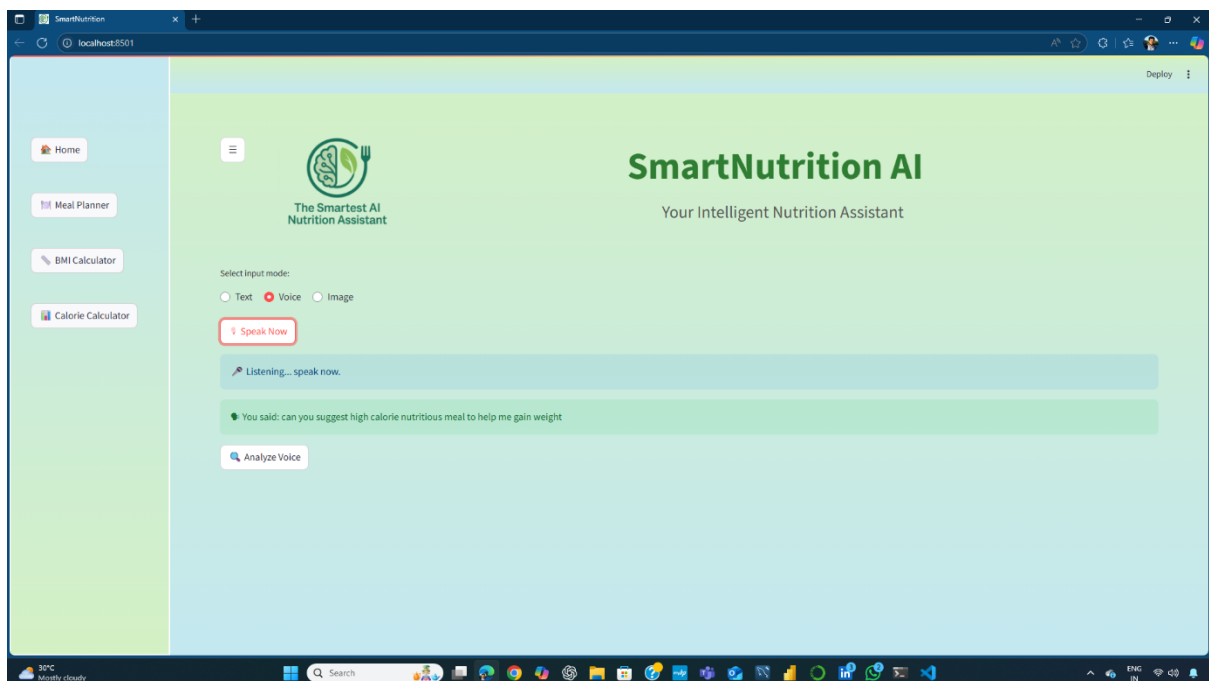


Figure-3.2(Voice Input result):

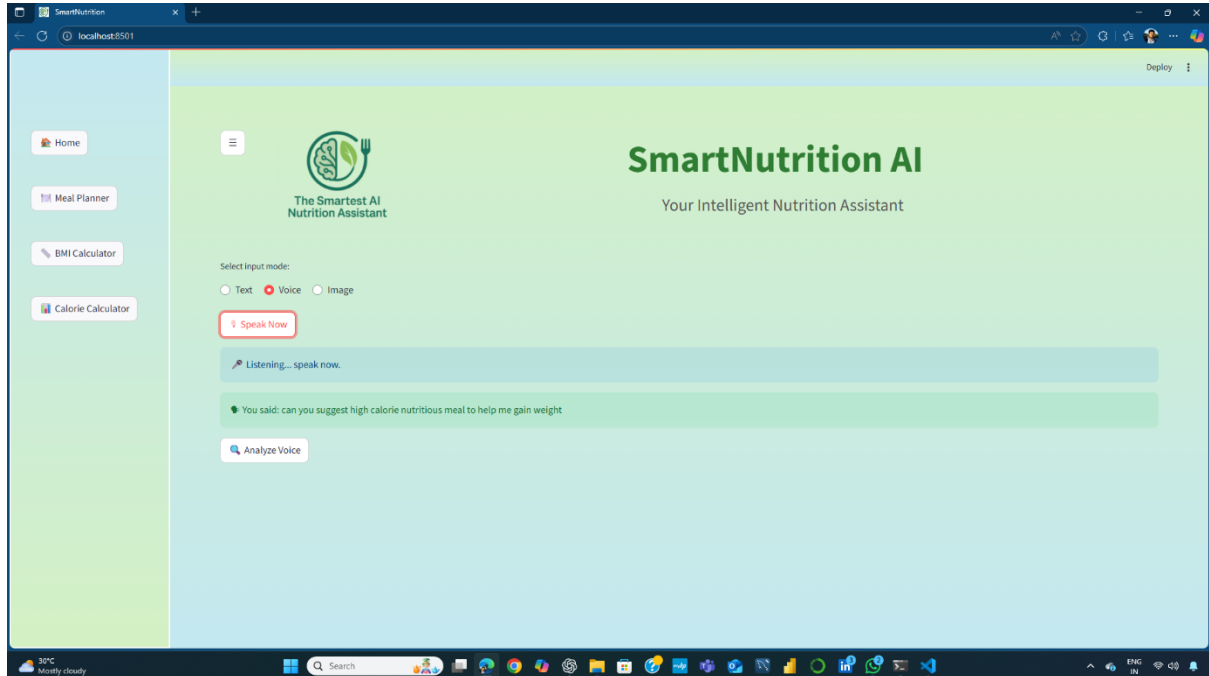


Figure-4.1(Image Input):

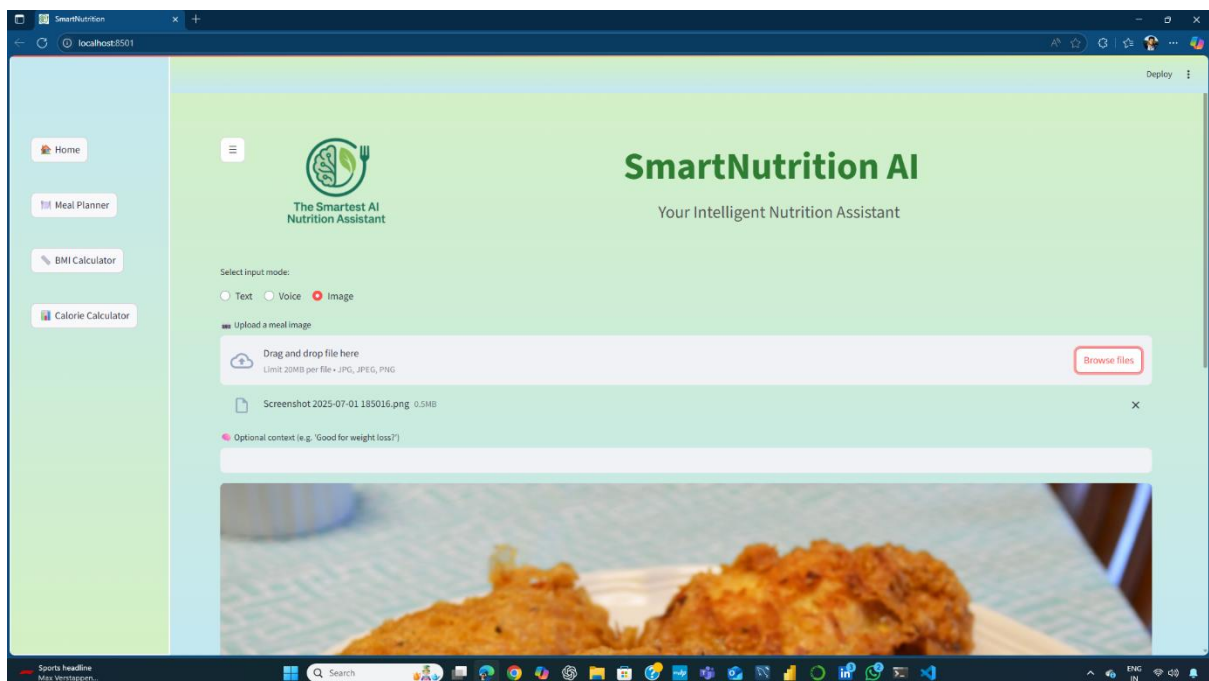


Figure-4.2(Uploaded Image):



Figure-4.3(Image input result):

SmartNutrition x +

localhost:8501

Uploaded Meal

Analyze Image

Nutrition Report

- Dish name: Fried Chicken
- Classification: Non-Veg
- Estimated calorie range: 400-600 calories per serving (depending on portion size and breading). A single piece of fried chicken can range from 200-300 calories, so four pieces will be substantially higher.
- Visible ingredients: Chicken pieces (likely bone-in), breading (likely flour, spices, and possibly eggs), cooking oil (type is not visible). Spices are apparent from the coloring.
- Health benefits or drawbacks:
 - Drawbacks:** Fried chicken is high in saturated fat, cholesterol, and sodium. The breading adds extra calories and carbohydrates. Regular consumption can contribute to weight gain, high cholesterol, heart disease, and other health problems. The cooking method (deep frying) makes it less healthy than other chicken preparations.
 - Benefits:** Chicken is a good source of protein, which is essential for building and repairing tissues. However, these benefits are significantly diminished by the frying process. The visible spices may provide some small amount of antioxidants depending on the blend used, but the negative effects substantially outweigh any potential micronutrient benefits.

Was this image result helpful?

Yes No

30°C Mostly cloudy

Search

ENG IN

Figure-5.1 (Meal planner input):

Figure-5.2 (Meal planner result):

Day	Breakfast	Lunch	Dinner
Monday	3 whole eggs scrambled with cheese & 2 slices whole wheat toast, 1 glass full-fat milk (400-500 calories approx.)	2 cups Chicken Biryani with raita (yogurt dip), 1 cup mixed vegetables (600-700 calories approx.)	2 Chicken Tikka Masala with 2 Rotis, 1 cup mixed salad (600-700 calories approx.)
Tuesday	2 Pancakes with 1 banana and whipped cream, 1 glass full-fat milk (450-600 calories approx.)	Rajma Chawal (Kidney bean curry with rice), 1 cup salad (500-600 calories approx.)	Butter Chicken with 2 Naan breads, 1 cup green beans (600-700 calories approx.)
Wednesday	Oatmeal with nuts, seeds, and dried fruits, 1 glass full-fat milk (450-550 calories approx.)	Mixed Vegetable Pulao with paneer (Indian cheese), Raita (450-550 calories approx.)	Mutton Rogan Josh with 2 Rotis (600-700 calories approx.)
Thursday	Peanut butter sandwich on whole-wheat bread, banana, 1 glass full-fat milk (400-500 calories approx.)	Dal Makhani (creamy lentil dish) with 2 Rotis, 1 cup mixed vegetables (500-600 calories approx.)	Fish curry with rice, 1 cup mixed vegetables (550-650 calories approx.)

Figure-5.3 (Meal planner result):

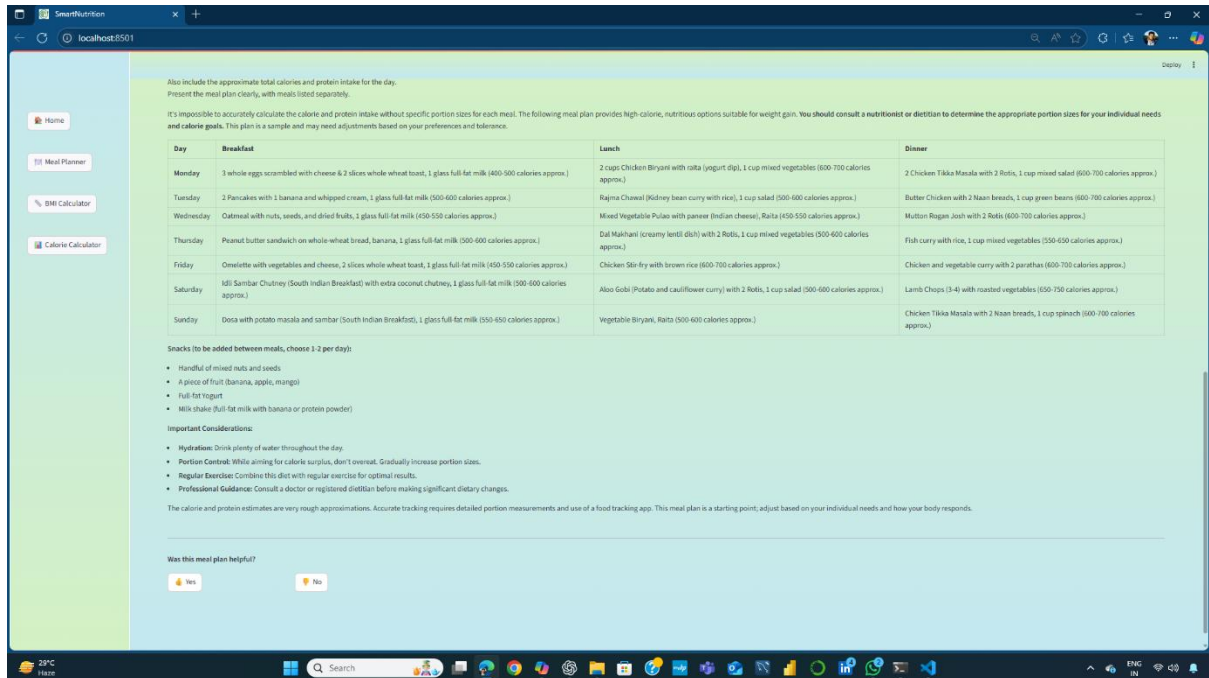


Figure-6.1 (BMI Calculator input):

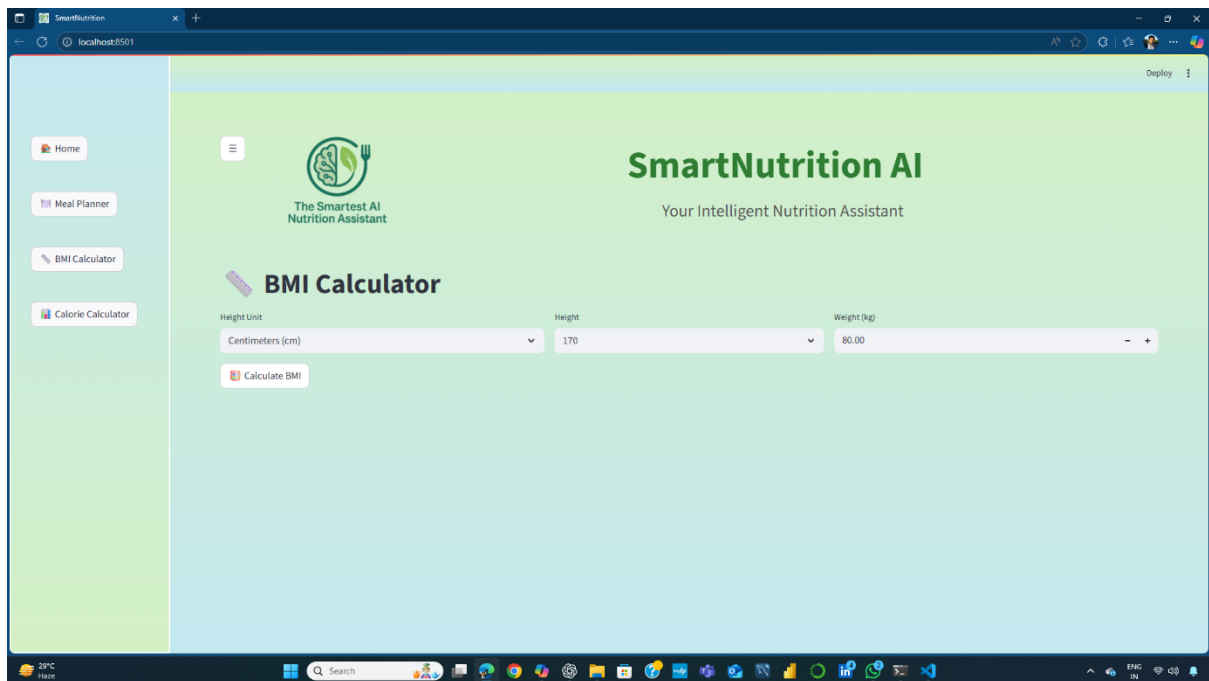
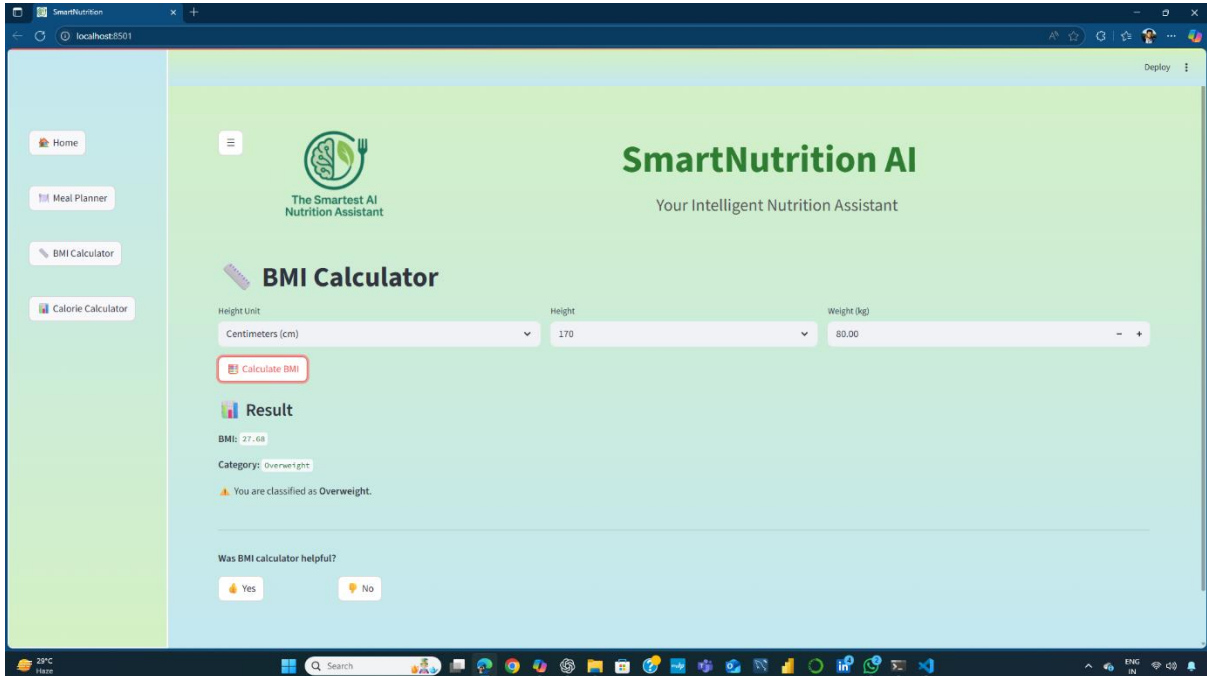
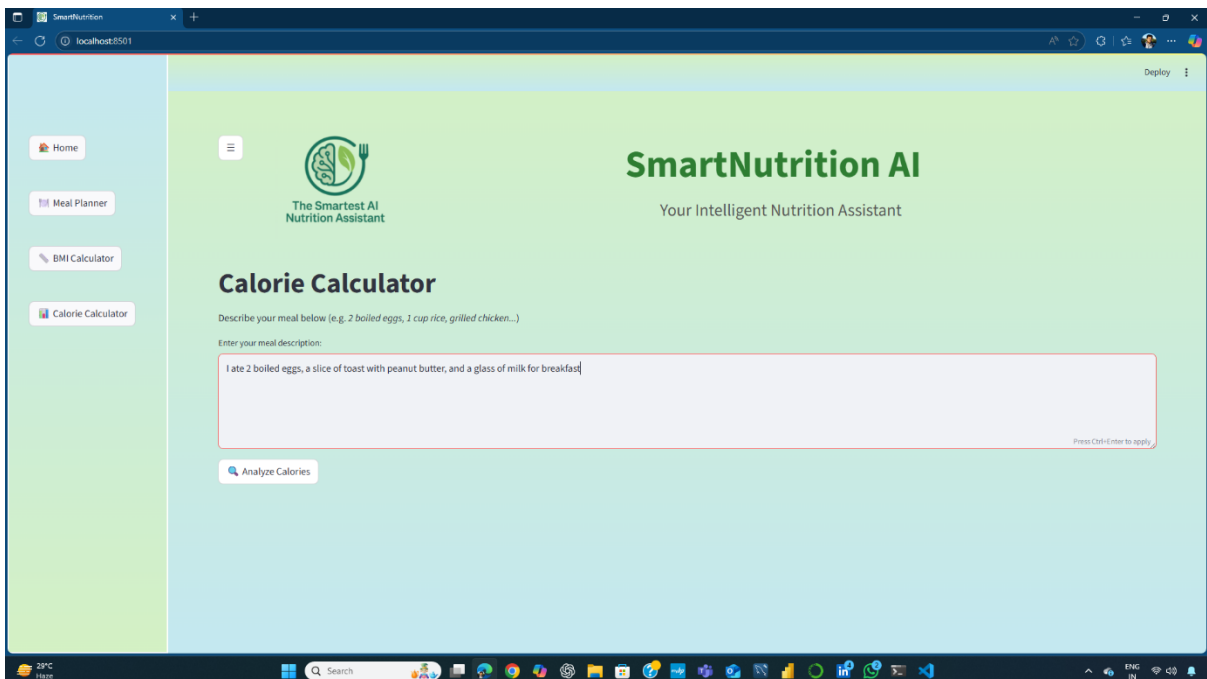


Figure-6.2 (BMI Calculator output):



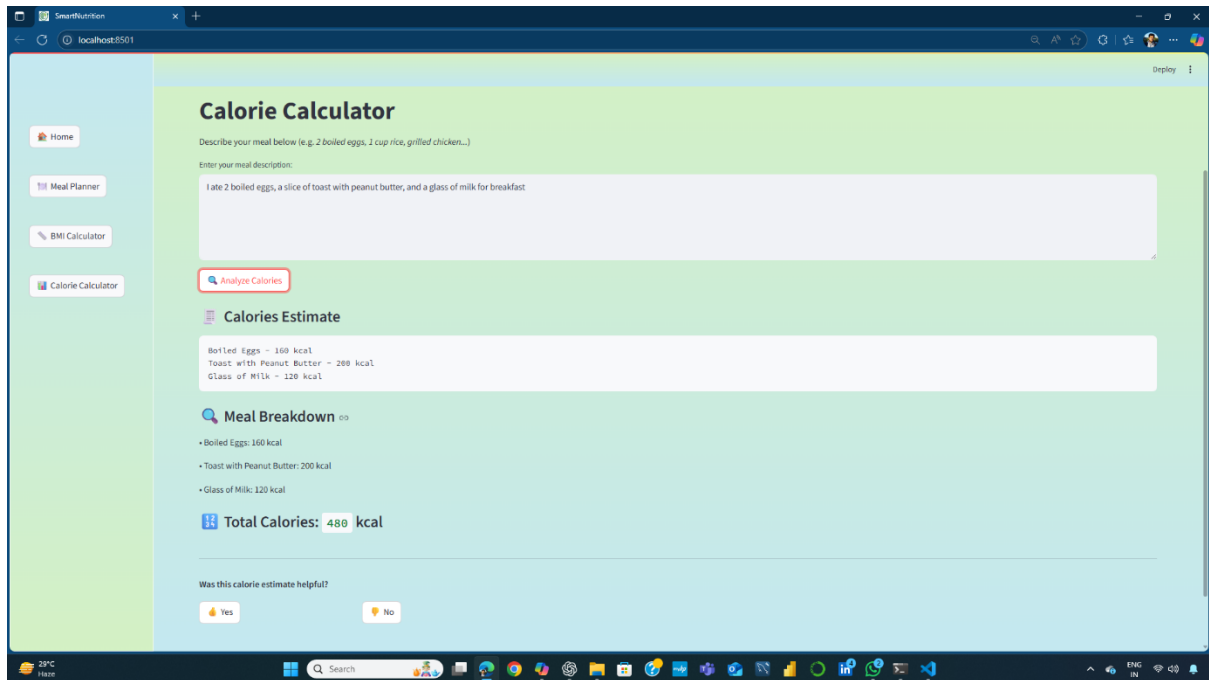
The screenshot displays the 'SmartNutrition AI' web application. On the left is a sidebar with navigation links: Home, Meal Planner, BMI Calculator, and Calorie Calculator. The main content area is titled 'SmartNutrition AI' and 'Your Intelligent Nutrition Assistant'. Below this is the 'BMI Calculator' section. It features input fields for 'Height Unit' (set to 'Centimeters (cm)'), 'Height' (170), and 'Weight (kg)' (80.00). A 'Calculate BMI' button is highlighted with a red box. Below the inputs, the 'Result' section shows 'BMI: 27.68' and 'Category: Overweight', with a message 'You are classified as Overweight.' At the bottom, there is a feedback prompt 'Was BMI calculator helpful?' with 'Yes' and 'No' buttons.

Figure-7.1(Calories calculator input):



The screenshot shows the 'SmartNutrition AI' web application with the 'Calorie Calculator' section active. The sidebar on the left remains the same. The main content area has a heading 'Calorie Calculator' and a prompt 'Describe your meal below (e.g. 2 boiled eggs, 1 cup rice, grilled chicken...)'. Below this is a text input field with the user's entry: 'I ate 2 boiled eggs, a slice of toast with peanut butter, and a glass of milk for breakfast'. An 'Analyze Calories' button is located below the input field. The bottom of the screen shows a Windows taskbar with various application icons and system tray icons.

Figure-7.2 (Calories calculator output):



Conclusion

Smart-Nutrition AI represents a compelling integration of modern machine learning capabilities into everyday health and nutrition decision-making. By combining the power of generative AI, OCR, and speech-to-text technologies into a single, multimodal interface, the application delivers a seamless and accessible user experience for a diverse range of users—from fitness enthusiasts and health-conscious individuals to busy professionals and people with dietary constraints.

This project not only bridges the gap between complex nutritional information and practical, personalized guidance but also showcases the potential of LLMs and multimodal AI in democratizing health services. Built with Streamlit for rapid development and enhanced user interaction, Smart-Nutrition AI is poised for real-world impact through its intuitive design and intelligent, context-aware responses.

As health consciousness grows globally, the demand for instant, trustworthy dietary insights will only intensify. Smart-Nutrition AI addresses this need directly and sets the foundation for future enhancements such as meal planning automation, dietary goal tracking, and broader multilingual support. With continual improvements and user-driven feedback, the platform is well-positioned to evolve into a comprehensive digital companion for smart, everyday nutrition.