Chapter 1
Introduction

In a time when technology is causing a revolution in how people live; health and nutrition have seen no exception to this change. People now care more about staying healthy, and they look to smart systems to help them live better. AI-powered digital tools are no longer just expected to react but also to comprehend, customize, and assist. This project arises from the need to connect medical dietary guidance with everyday meal planning through an AI-based system that suggests recipes while being mindful of health.

1.1 Background

In recent times, combining artificial intelligence natural language processing, and healthcare has led to exciting advancements in creating personalized digital health tools. One standout is recipe recommendation systems, which are becoming more popular as people focus more on diet chronic illnesses[11], and overall health. These days, individuals seek more than just delicious food. They want meals that align with specific health goals fit their dietary needs, and suit their taste. With conditions like diabetes, obesity, and hypertension increasing, many now turn to meal planning that considers their health. At the same time, people often find it hard to turn nutritional or medical advice into everyday food decisions when restrictions make it tricky.

People used to rely on static diet plans or sit down with dietitians to deal with these issues. These options worked but didn't suit everyone because they took too much time, were often one-size-fits-all, and couldn't easily adapt to individual needs. Today, AI tools like chatbots and smart assistants have changed the game. They allow instant, personalized, and interactive help with diets. Tools like the Spoonacular API make it easy to access recipes and nutrition info. Databases such as Pinecone store user details and patterns, while advanced NLP tools like Gemini help systems understand not just what people say but what they mean even when their words aren't super clear.

1.2 Motivation

The main goal of creating a Recipe Suggestion Chatbot is to tackle the issue of turning medical diet advice into practical meal ideas. Doctors and dietitians often say things like "eat less sugar" or "stay away from salty foods," but most people struggle to turn those directions into real recipes or food choices. This becomes even harder when they cook at home or look for recipes online. The situation gets trickier with people who don't know much about what's in common ingredients or use casual or regional ways to explain what they need, like saying "I have BP," "sugar-free foods," or "simple food options."

People who have more than one dietary restriction, like someone with diabetes who is also vegetarian often struggle more to find meals that meet both their diet and health requirements.

Most food suggestion apps today feel too basic, don't understand specific contexts well, and fail to adjust to changing user inputs. An advanced system could fill this need by understanding natural language spotting health issues, removing harmful ingredients, predicting personalized diet needs, and still offering delicious recipes. This mix of technical tools and humanitarian thinking aims to use AI to make healthy eating simpler, smarter, and within everyone's reach.

1.3 Problem Statement

This project focuses on solving a key issue: the absence of a smart system to suggest recipes that consider health and personal preferences. It aims to understand what users say figure out dietary limitations, and recommend meals with the right nutrition.it seeks to address several problems:

- People often describe their health needs or diets in unclear or casual ways.
- Different illnesses come with unique diet rules, and many don't realize which ingredients or nutrients might affect them.
- Current tools fail to customize suggestions based on individual health or nutrition needs.
- When perfect recipe matches are missing, there's no reliable backup plan in place.

The task involves building a system that can grasp everyday language and turn it into well-structured health and nutrition knowledge. This chatbot needs several parts to work together, including NLP, machine learning, mapping between diseases and nutrients, and external APIs. These parts ensure the recipe ideas are tailored, safe to use, and fit the situation. The system should also be easy to scale, simple for users, and built in a way that works even when users provide minimal input.

1.4 Objectives

This project aims to create a smart recipe suggestion chatbot that can offer meal ideas personalized to user health issues, food choices, and how they communicate. The main goals include:

To create a chatbot to understand natural informal user speech: The system needs to handle informal spelled, or vague words and connect them to meaningful actions. It recognizes terms like "bp" to mean hypertension or understands a phrase like "no sugar" as a possible clue for diabetes. Technologies like NLP methods, Gemini API, sentence transformers, and fuzzy matching help achieve this.

To apply disease-linked dietary rules: The chatbot must detect diseases and pair them with appropriate dietary rules. It removes harmful ingredients such as sugar or sodium and tracks if nutrients stay within safe levels. The backend links each disease with its specific harmful nutrients and ingredients.

To create custom nutrition suggestions: A machine learning model a Random Forest Regressor, uses details like age, weight, height, gender, activity level, and diet choices to predict what calories and macronutrients an individual needs. This helps the system pick meals that are both safe and meet nutritional goals.

To suggest recipes with only safe and chosen ingredients: The system removes bad ingredients from the user's list and makes sure every suggested recipe uses the safe ones left. This applies to both the main recipes and backup options.

To offer alternatives when no ideal matches exist: If no recipe fits all constraints for disease and nutrients, the system should suggest backup recipes. These alternatives loosen nutrient requirements a bit but still avoid harmful ingredients and include the safe ones. This guarantees users always get some workable ideas.

To integrate vector embeddings and external APIs to scale interactions: The system uses Pinecone to store user data and chat records, Spoonacular to look up recipes, and Gemini to interpret user requests. These tools must work together to provide instant customized interactions.

1.5 Scope of the Project

This project covers building a full recipe recommendation chatbot. It handles natural language understanding, creates user profiles, filters for specific diseases, and generates recipes based on individual needs. The chatbot can do the following:

- Communicating with users using a chat window on the web.
- Taking casual input and turning it into organized data about diseases and diets.
- Using machine learning to figure out what nutrition a person needs.
- Suggesting and filtering recipes through the Spoonacular API while keeping in mind:
    - Ingredients a person avoids
    - Safe nutrient levels
    - What a user prefers to eat
    - Ingredients to include

The chatbot stores user profiles with Pinecone saving past chats to keep context in future conversations. It also makes sure fallback responses work. Right now, it works on the web and handles English-input. Plans to improve include mobile apps, voice-based input, support for multiple languages, and connections to wearable devices or smart nutrition tracking systems.

Chapter 2
Literature Review

2.1 Existing systems and limitations

Traditional recipe recommendation structures have typically relied on person input thru keywords or selection from predefined categories. One super early device, the Recipe Suggestion System (RSS), described in the IFMBE Proceedings of the European Conference of the International Federation for Medical and Biological Engineering (ECIFMBE) (2008), included clever home technology like shrewd fridges and fitness-tracking gadgets to suggest meals primarily based on fitness condition, favored substances, and marketplace costs. However, its reliance on predefined good judgment and shortage of herbal language communication constrained its person engagement.

More current systems have taken specific strategies. For example, RecipeIS[2], supplied in Applied Sciences (MDPI, 2023), is a web utility that combines picture recognition (the usage of ResNet-50) with recipe advice via the Edamam API. This model achieved excessive class accuracy (96%) on ingredient reputation however still lacked robust conversational capacity.

Another modern device, Hunger's Heaven[1], as posted in Mathematics and Computer Science (SciencePG, 2024), offered AI-powered recipe hints primarily based on consumer nutritional restrictions and covered social networking capabilities for sharing and gamifying cooking studies. Despite its strengths in engagement and customized suggestions, it become built often as an app with constant filtering options and lacked contextual know-how in conversations.

These structures mirror development in the usage of AI for recipe recommendations however highlight continual limitations: maximum are non-conversational, lack actual-time consumer choice edition, and do now not provide a herbal, emotionally enticing talk.

2.2 Research paper and relevant technologies

The evolution of chatbot technology and conversational structures has extended drastically with the upward push of machine getting to know and deep neural networks. A foundational literature survey performed with the aid of Caldarini, Jaf, and McGarry, published in Information (MDPI, 2022), gives a radical assessment of ways chatbot structures have progressed from pattern-matching models like ELIZA to fashionable AI-primarily based architectures. The authors detail the progression from static, area-precise structures to the ones capable of knowledge semantic nuance and emotional tone. Their survey identifies two core demanding situations in modern chatbot structures: contextual know-how and emotional intelligence. They argue that even with advanced NLP fashions, maximum chatbots still warfare to observe long conversations, don't forget person alternatives over a couple of interactions, and respond empathetically. These

limitations are particularly applicable for domain names like food and health, in which trust and user pleasure hinge on personalization and human-like verbal exchange.

To cope with some of those problems, researchers at Facebook AI (Roller et al., 2020) evolved a modern-day open-domain chatbot described within the paper "Recipes for Building an Open-Domain Chatbot," released as a preprint on arXiv. These paintings brought the concept of blending multiple conversational competencies—along with empathy, understanding dissemination, and persona expression—right into a unified talk system. The researchers skilled transformer-based totally fashions starting from 90M to nine.4B parameters and first-rate-tuned them using the Blended Skill Talk (BST) dataset. The end result becomes a model that done higher in human opinions than many conventional bots, especially in metrics of engagingness and humanness. The authors additionally emphasized the role of interpreting strategies (e.g., beam seek, sampling) and reaction period in enhancing the naturalness of chatbot replies. Despite their staggering consequences, these models had been no longer tailored for precise domains like culinary steering or fitness-conscious suggestions, because of this their wellknown-motive nature lacks the precision wanted for meals advice use cases.

Additionally, other papers have targeted on area-unique programs. For instance, numerous studies summarized in Applied Sciences (2023) reviewed the usage of convolutional neural networks inclusive of Inception-v3, SENet-154, and AlexNet for component and recipe popularity. These studies emphasized the importance of accurate photo category in recipe advice and explored hybrid models that integrate factor detection with user preference filters. However, at the same time as effective in identifying visual inputs, they nonetheless do not provide the level of interactivity or adaptability required for a chatbot that ambitions to simulate human verbal exchange and understand nuanced queries about weight-reduction plan, delicacies, or ingredient substitution.

In sum, while studies have progressed in both conversational AI and food-specific advice models, the two domain names have not often been fused efficiently. The possibility lies in integrating cutting-edge conversational fashions with meals-unique know-how graphs, fitness metadata, and user interplay histories to create a customized and interactive recipe chatbot.

2.3 Comparative analysis

System / Research	Journal / Source	Key Technology Used	Strengths	Limitations
Recipe Suggestion System (RSS)[3]	IFMBE Proceedings (ECIFMBE), 2008	Smart appliances + health monitoring	Health-aware, smart home integration	Non-conversational, rule-based

RecipeIS	Applied Sciences (MDPI), 2023	ResNet-50 + Edamam API	Image-based recognition, accurate	Minimal dialogue or user feedback loop
Hunger's Heaven	Mathematics and Computer Science (SciencePG), 2024	Node.js, MongoDB, AI	Dietary-aware, gamified, user-community focus	Lack of true conversation and context awareness
Caldarini et al. (Chatbots Survey) [5]	Information (MDPI), 2022	NLP + Deep Learning	Broad review, history of chatbot evolution	General-purpose, not domain-specific
Roller et al. (Open-Domain Bots) [4]	arXiv preprint, Facebook AI Research, 2020	Transformer-based (BST)	Empathy, multi-skill training, open-domain	Limited food-specific integration, hallucinations

2.4 Identified gaps

Despite advancements in recipe advice and chatbot technology, key boundaries persist that highlight the want for a more advanced, personalized chatbot. Most systems lack context awareness and cannot don't forget beyond interactions, proscribing long-time period personalization. Health and dietary considerations are dealt with through static filters as opposed to dynamic, real-time reasoning. Emotional intelligence is regularly absent, reducing user engagement and consider. Multimodal input integration—such as combining text, voice, photos, and health facts—is hardly ever explored. Additionally, cutting-edge retrieval strategies like vector-primarily based similarity matching stay underutilized. Bridging those gaps is important for developing a truly sensible and empathetic recipe chatbot.

Chapter 3
System Structure

The Recipe Suggestion Chatbot system uses an intelligent modular design to give users recipe ideas tailored to their health needs, diet choices, and medical conditions. It follows a client-server model where the backend handles data processing, works with external APIs, and runs machine learning models. On the other side, the frontend offers a simple and conversational interface for people to use. A Flask-based web server serves as the main hub organizing tasks like routing, managing user sessions, storing data, and connecting different parts of the system. Natural Language Processing tools using the Gemini API, allow the system to understand casual user inputs and pull-out details such as ingredients or health issues. At the same time, a machine learning algorithm called Random Forest Regressor [14][15] calculates daily nutrition needs based on factors like age, weight, height, and activity level. The system fetches up-to-date recipes from the Spoonacular API and uses a tool called Pinecone—a vector database—to match user preferences with recipes through semantic similarity powered by Sentence Transformers. These components work together to suggest recipes that align with the user's dietary and health goals. If a perfect match isn't available, the system also offers alternative suggestions. The system uses two main layers: **Frontend (Client-side)** and **Backend (Server-side)**. These layers exchange information through API calls and HTTP requests. External tools like Pinecone, Spoonacular, and Gemini take care of tasks like storing data, fetching recipes, and processing language. This setup keeps it modular and easy to expand

3.1 Backend Elements:

Flask Web App (Main Server)

The Flask app works as the main hub. It manages HTTP routes, processes user input, and handles API communication to create responses. It also oversees user sessions, authentication, and links the backend components such as machine learning natural language processing, and database services.

Machine Learning Models

A Random Forest Regressor in the backend predicts users' daily needs for calories and macronutrients. It bases these predictions on factors like age, weight, height, gender, and activity level. To encode disease and ingredient names into vector embeddings, the system uses Sentence Transformers (all-MiniLM-L6-v2). These embeddings allow fuzzy matching of diseases and enable profile searches using semantic similarities.

Natural Language Processing (Gemini API)

The Gemini API interprets user input through natural language processing. It identifies key

entities such as diseases and ingredients while determining user intent. It also handles recognizing synonyms and crafts chatbot responses that sound human creating smooth and smart conversations.

Pinecone Vector Database

Pinecone holds user data like profile vectors, health embeddings, and chat records. It allows fast searches and retrieval through similarity checks. Each user's data stays in its own namespace to maintain privacy, security, and reliable scalability.

Spoonacular API (External Recipe API)

The Spoonacular API offers recipes along with complete nutrition details. It lets users look up recipes by ingredients nutrient levels like sugar or sodium, and diet choices. If no perfect match is found, it uses relaxed nutrient constraints to give alternative suggestions.

3.2 Data Processing & Preparation

Developers work with Pandas and NumPy to handle data like user details, recipe databases, and disease-nutrient links. The system adjusts numerical values, converts categories into usable formats, and matches ingredient synonyms to better align recipes with user needs. It also filters out harmful ingredients before recipes are retrieved.

3.3 Frontend Elements:

Chat Interface:

The frontend uses HTML, CSS, and JavaScript to deliver a simple chat-like interface. Users can chat with the bot, share food preferences, request meal ideas, and get live recipe suggestions.

User Dashboard & Forms:

The dashboard helps users sign up, log in, and use features tailored to them. They can check diet plans made for them, look back at earlier chats, or choose to erase saved data if needed. Forms gather details like health issues, gender, and preferred diet.

Interaction & Data Flow:

Users start by signing up or logging in. During this step, the system saves their basic profile information in Pinecone as vector embeddings. If someone asks something like, "Suggest lunch for diabetes," the Gemini API identifies illnesses and key ingredients while machine learning tools analyze the request. Using this data, it pulls recipes from Spoonacular that avoid harmful items and follow required nutritional guidelines. When no exact match is available, it shows backup recipes that use more flexible nutrient criteria to keep results consistent. It also creates a seven-day diet plan based on expected nutritional needs.

Security and Scalability:

Werkzeug hashes user passwords. Flask-Login handles user sessions and makes sure that logged-in users can use features meant for them. Pinecone keeps storage secure and scalable for each user by using namespace isolation. The whole setup is modular making it easy to connect with wearables mobile apps, or other data sources later

Chapter 4
Technology used

Building a smart chatbot to suggest recipes while being aware of health needs uses many types of technologies. It connects tools from areas like web design, databases, language processing, and machine learning. This chapter explains the main technologies, software, libraries, and APIs picked to help different parts of the system work well. Every tool had an important role, from making the chatbot easy to use and accurate in suggestions to handling data and storing it in a way that grows with the system. These tools all came together to make the chatbot strong and focused on what users need.

Python 3

The project relies on Python 3 as the main programming language because it's easy to learn and has clear syntax. It has a rich set of libraries that make it a good fit. The team picked Python to power the backend using Flask, process data with pandas and NumPy, create machine learning models, and connect with APIs like Spoonacular and Gemini.

Flask Framework

Flask works as the foundation for the backend. The project uses it because it's simple and flexible enough to meet the needs of this chatbot. Flask handles user authentication and sessions, processes HTTP requests, manages routing, and helps connect with external APIs while also organizing how the chatbot functions.

Machine Learning (Random Forest – scikit-learn)

The Random Forest Regressor helps in predicting daily calorie needs and macronutrient breakdowns for users based on their profiles. Its ability to handle tabular health and demographic data with accuracy and clarity made it a good choice. The implementation is done in modules like fitness_diet_plan.py and diet_chart_without_disease.py.

Sentence Transformers (all-MiniLM-L6-v2)

This model converts user queries, diseases, and ingredients into vector representations to capture their meaning. It helps the system grasp user input context and purpose, match diseases even when phrased, and perform similarity searches in Pinecone to maintain user profiling and chat histories.

Gemini API (NLP)

Gemini helps the system understand language better. It breaks down user inputs to identify structured entities like disease names or ingredients, figures out what the user wants, and creates natural responses for conversations[12][13]. It also manages unclear questions and identifies alternative words with the same meaning.

Pinecone (Vector Database)

Pinecone stores and retrieves complex vector embeddings tied to user profiles and chat logs. It provides tools to perform fast similarity searches and real-time semantic lookups. With

namespace isolation, it keeps each user's data private and separate to help give unique recommendations.

Spoonacular API

The Spoonacular API lets the system tap into a wide collection of recipes and their nutrition details. It pulls recipes based on what users have, their dietary needs, or specific health-related nutrient limits. If strict criteria cannot be met, it offers fallback suggestions. It also includes nutrition labels to align recipes with machine learning-driven dietary plans.

Pandas and NumPy

People use these Python libraries a lot to clean, prepare, and study data. They assist with tasks like loading data about users and recipes adjusting numeric details like height or weight, turning categories like gender or diet type into usable data, and keeping track of relationships between diseases, ingredients, and nutrients.

Flask-Login & Werkzeug

Flask-Login and Werkzeug offer tools to manage user logins and sessions. They handle secure password storage, let users log in or out, and keep sessions active. These features help create user-focused and personalized interactions.

HTML/CSS/JavaScript

These are essential tools to design how the chatbot looks and functions on the front end. They handle form inputs, show chats, and connect with the backend using AJAX or forms. This keeps the interface user-friendly and quick to respond.

Other Tools & Libraries

LabelEncoder from scikit-learn helps to turn categories into numbers so that machine learning can work with them better. Libraries like difflib or fuzzywuzzy assist in matching misspelled or inconsistent disease names to standard names. Though not critical in this version, tools like Matplotlib or Seaborn can be used to plot diet plans or show nutrient charts.

Chapter 5
System Implementation

Building this smart chatbot for recipe suggestions includes a setup made up of several parts. Each part plays an important role in turning what users say into recipe ideas that are safe and tailored to them. This section explains how the system is built. It covers everything from getting the data ready and filtering for diseases to connecting with APIs and creating the user interface. The team designed each part to handle growth, stay precise, and keep things easy for users. This makes sure the system works well even when users ask casual questions or have tricky health issues.

5.1 Dataset Used

The performance of an AI-based recommendation system depends a lot on how good and organized the datasets used for its training and testing are. This work uses four main datasets.

5.1.1 Disease Nutrient Dataset:

This dataset includes the diseases and the harmful and safe macro nutrients and recipe ingredients for that disease.

Fields in our dataset:

- **Disease Name**: The name of the medical condition (e.g., *Diabetes*, *Hypertension*, *Obesity*).
- **Bad Ingredients**: A list of ingredients separated by commas that are known to have a negative impact on people with the given disease.
- **Safe Nutrients**: Nutrients needed in moderate or high amounts help manage the disease. These show what ingredients should include more of, like fiber or protein in the case of diabetes.
- **Bad Nutrients**: To manage the disease well, people need to limit certain nutrients like *sodium* to control hypertension or *saturated fat* to help with heart disease. Recipes containing a lot of these nutrients are removed.
- **Bad Nutrients Safe Limits (mg/g):** The maximum daily limit to consume each harmful nutrient is suggested based on the needs of someone with the disease. These limits help create strict rules to sort recipes such as keeping sodium under 1500 milligrams a day.

5.1.2 Ingredient synonyms Dataset:

The Recipe Ingredient Synonyms dataset aims to address the challenge of different and unclear ingredient names when people use informal or regional words. It acts as a guide to match these terms with standardized ingredient names to make sure the system processes them in the same way.

Fields in our dataset:

- **Aliased Ingredient Name:**
  This is the main standardized name given to an ingredient like *bell pepper*. Systems use this version to keep everything consistent during processing.
- **Ingredient Synonyms:**
  These are alternate names regional variations, or even common typos for the same ingredient listed with commas. For example, *capsicum* or *sweet pepper* can represent *bell pepper*. This list helps match what users type to the right ingredient.
- **Category:**
  This refers to the broad group an ingredient falls under such as *vegetable*, *protein*, or *spice*. It helps to organize recipes, analyze nutritional data, and filter items

5.1.3 Recipes with macro nutrients breakdown dataset:

The Recipes with Macronutrient Breakdown dataset lists nutritional details for many different recipes. This dataset is key to the system because it helps the chatbot suggest meals that fit a user's diet, health goals, or calorie limits. Each recipe in the dataset includes its nutrient details type of diet it fits, and serving size.

Fields in our dataset:

- **Name**: The name or title of the dish[6].
- **RecipeCategory:** Shows the type of cuisine or meal, like dessert or main course.
- **Calories, CarbohydrateContent, SugarContent, CholesterolContent, ProteinContent**: These give a breakdown of nutrients and cholesterol in each serving. They help to analyze nutrition and sort recipes based on health needs.
- **RecipeServings**: States how many servings the dish makes, which helps adjust nutrient amounts per portion.
- **Dietary Type**: Labels the diet type of the recipe such as Keto, Vegan, or Vegetarian. This helps match recipes to what users prefer.

5.1.4 Detailed Meals Macros Dataset:

The Daily Macronutrient Requirement Dataset[8][9] has specific data tailored to predict how much nutrition a person needs. This prediction considers factors like age, body stats, way of living, and health conditions. It helps provide a base for personal meal plans by showing the system the ideal amount of energy in calories and macronutrients like carbs, protein, and sugar each individual should have. It also takes into account their health issues and how active they are.
Fields in our dataset:

- **Age**: User's age measured in years, has an influence on both basal metabolic rate and nutrient needs.
- **Gender**: Defines biological sex as male or female and helps adjust suggestions for calories and nutrients.
- **Height**: Measurement of the user's height in centimeters. It helps calculate body metrics like BMI.
- **Weight**: The user's weight in kilograms contributes to determining both BMI and daily calorie needs when combined with height.
- **Daily Calorie Target**: Estimated total calories the user is suggested to consume in a day.
- **Sugar**: The advised daily limit for sugar intake based on the user's health and energy requirements.
- **Protein**: The user's daily protein goal necessary to support muscle upkeep and recovery.
- **Carbs Goal**: Suggested daily amount of carbohydrates based on health issues and how active someone is.
- **Health Condition**: The user's medical issue (like Diabetes or Hypertension) helps limit harmful nutrients.
- **Physical Activity**: Shows how much the person exercises (like low, moderate, or high) and ties to how many calories they need.
- **Preferred Diet**: The user's chosen eating style (like Keto or Vegetarian), which helps pick suitable recipes.

5.2 Core module description

The system's structure includes several key parts that work together to provide a user-friendly recipe suggestion system. These main parts are the Flask-based Web Server, the Machine Learning (ML) Engine, the Natural Language Processing (NLP) Layer, the Vector Matching Module, and the API Communication Handlers. The Flask server manages request routing, handles sessions, and runs the main logic. The ML Engine uses the Random Forest Regressor to guess a user's nutritional needs (such as calories, carbs, fats, and proteins) based on their personal info. The NLP Layer uses Gemini API and Sentence Transformers to understand casual user questions, spot diseases and ingredients, and help compare meanings. The Vector Matching Module uses Pinecone to store and find user data for personalized ideas. , the API Handlers deal with Spoonacular for recipes and Gemini for NLP tasks. Each of these parts can be updated on its own, which makes the system flexible able to grow, and easy to add to.

The core modules create a strong base, but extra documentations and visuals can boost how easy it is to read and get new developers up to speed. Here are some ideas to make things clearer—like showing how things depend on each other sharing bits of code, and drawing out how it's all put together. These extras would shine a light on how the modules work together and help keep

the system in good shape. They'd give instant insight for when we need to grow the system or fix any bugs down the road.

5.2.1 Module Dependency Table

Here is a table to explain how components work together by mapping modules to their dependencies.

Module	Key dependencies	External API
Flask Web Server	Flask, Flask-Login, Flask-Session	__
ML Engine	scikit-learn, pandas, numpy	__
NLP Layer	google-generativeai, Sentence-Transformers	Gemini API
Vector Matching	pinecone-client, Sentence-Transformers	Pinecone DB
API Handlers	requests, json	Spoonacular API, Gemini API

Code Snippet for critical flows

Short examples showing inter-module communication**:**

Example 1: NLP Layer → Vector Matching

```
# In recipe_utils.py (NLP Layer)
def match_disease(user_input):
    embedding = model1.encode(user_input, convert_to_tensor=True)  # NLP processing
    results = index.query(vector=embedding.tolist(), top_k=3)     # Pinecone lookup
    return results
```

Example 2 :ML Engine → API Handlers

```
# In diet_chart_without_disease.py (ML Engine)
daily_macros = predict_macros(model, user_input)  # Calorie prediction
recipes = get_recipes_by_type("Lunch", daily_macros)  # Calls Spoonacular API
```

5.3 Data Preprocessing

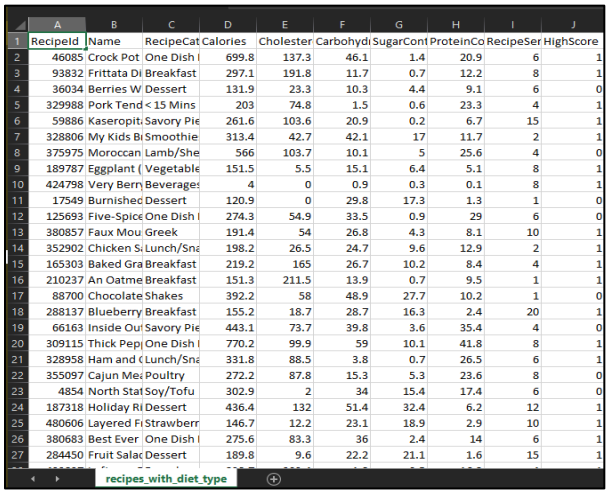
Data preprocessing forms the inspiration of every prediction and selection made through the machine. The method starts with dataset introduction, wherein each guide curation and automated web scraping techniques are hired to gather comprehensive data. Specifically, sickness names and related facts are scraped from MedlinePlus the usage of Python libraries which include requests and BeautifulSoup. This automatic extraction guarantees up-to-date and various clinical terminology, which is vital for producing a rich ailment-precise nutrition dataset. Randomized logic is then applied to complement each ailment entry with applicable categories, tiers, terrible substances, awful nutrients, secure nutrients, and medically accepted secure nutrient limits, generating a dependent and sensible dataset.

Once datasets are accrued, they're cleaned the use of Pandas to make sure consistency and readiness for gadget learning tasks. Redundant or irrelevant columns are dropped, and missing or anomalous values are treated the use of suitable imputation or elimination strategies. Inconsistent entries—such as various spellings, letter instances, or synonymous terms for elements—are resolved thru normalization and a custom synonym mapping dictionary.

For numerical attributes like weight and peak, normalization is done to scale the values within a well-known range, facilitating higher convergence for device learning models. Categorical variables, together with gender and nutritional habit, are encoded the use of LabelEncoder to cause them to compatible with predictive algorithms.

Major enhancement in this level includes introducing a "Dietary Preference" column to the recipe dataset, derived from the recipe class (e.g., “Vegetarian,” “Vegan,” “Keto”), allowing higher alignment with person eating regimen sorts. Rows with unrecognized or ambiguous recipe classes that cannot be reliably classified are excluded to maintain accuracy and relevance. Additionally, all nutritional gadgets (such as grams and milligrams) are standardized across recipes to ensure uniformity in nutrient-primarily based filtering. The completely preprocessed and curated datasets—created through a aggregate of scraping, cleaning, enrichment, and transformation—are then used to train device gaining knowledge of models, force vectorization logic, and help disease-safe recipe filtering, making this a critical step within the system pipeline.

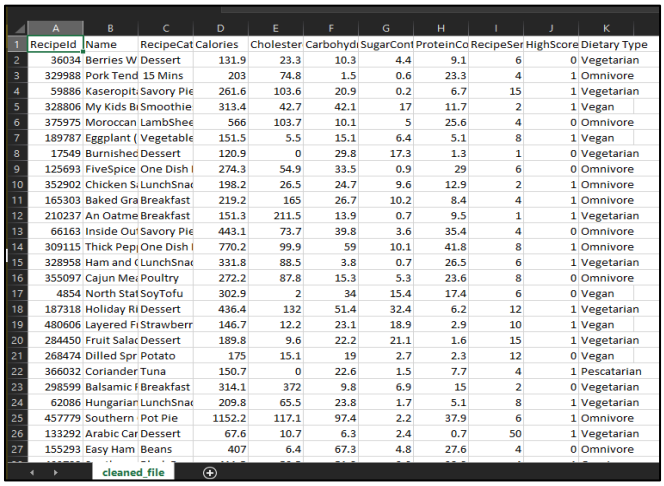
5.3.1 Enhanced Data Preprocessing Flow Description



	A	B	C	D	E	F	G	H	I	J
1	RecipeId	Name	RecipeCat	Calories	Cholester	Carbohydi	SugarCont	ProteinCo	RecipeSer	HighScore
2	46085	Crock Pot	One Dish I	699.8	137.3	46.1	1.4	20.9	6	1
3	93832	Frittata Di	Breakfast	297.1	191.8	11.7	0.7	12.2	8	1
4	36034	Berries W	Dessert	131.9	23.3	10.3	4.4	9.1	6	0
5	329988	Pork Tend	< 15 Mins	203	74.8	1.5	0.6	23.3	4	1
6	59886	Kaseropit	Savory Pie	261.6	103.6	20.9	0.2	6.7	15	1
7	328806	My Kids Bi	Smoothie	313.4	42.7	42.1	17	11.7	2	1
8	375975	Moroccan	Lamb/She	566	103.7	10.1	5	25.6	4	0
9	189787	Eggplant (	Vegetable	151.5	5.5	15.1	6.4	5.1	8	1
10	424798	Very Bern	Beverages	4	0	0.9	0.3	0.1	8	1
11	17549	Burnished	Dessert	120.9	0	29.8	17.3	1.3	1	0
12	125693	Five-Spice	One Dish I	274.3	54.9	33.5	0.9	29	6	0
13	380857	Faux Mou	Greek	191.4	54	26.8	4.3	8.1	10	1
14	352902	Chicken Si	Lunch/Sna	198.2	26.5	24.7	9.6	12.9	2	1
15	165303	Baked Gra	Breakfast	219.2	165	26.7	10.2	8.4	4	1
16	210237	An Oatme	Breakfast	151.3	211.5	13.9	0.7	9.5	1	1
17	88700	Chocolate	Shakes	392.2	58	48.9	27.7	10.2	1	0
18	288137	Blueberry	Breakfast	155.2	18.7	28.7	16.3	2.4	20	1
19	66163	Inside Ou	Savory Pie	443.1	73.7	39.8	3.6	35.4	4	0
20	309115	Thick Pep	One Dish I	770.2	99.9	59	10.1	41.8	8	1
21	328958	Ham and (	Lunch/Sna	331.8	88.5	3.8	0.7	26.5	6	1
22	355097	Cajun Mea	Poultry	272.2	87.8	15.3	5.3	23.6	8	0
23	4854	North Stat	Soy/Tofu	302.9	2	34	15.4	17.4	6	0
24	187318	Holiday Ri	Dessert	436.4	132	51.4	32.4	6.2	12	1
25	480606	Layered Fi	Strawberr	146.7	12.2	23.1	18.9	2.9	10	1
26	380683	Best Ever	One Dish I	275.6	83.3	36	2.4	14	6	1
27	284450	Fruit Salac	Dessert	189.8	9.6	22.2	21.1	1.6	15	1

recipes_with_diet_type

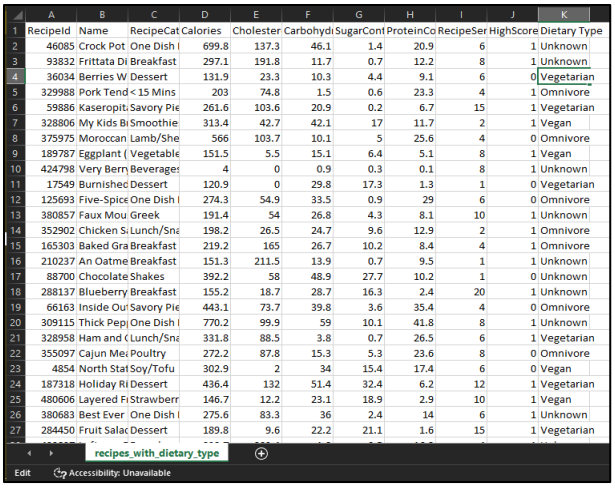
Fig1: *Raw Dataset*



	A	B	C	D	E	F	G	H	I	J	K
1	RecipeId	Name	RecipeCat	Calories	Cholester	Carbohydi	SugarCont	ProteinCo	RecipeSer	HighScore	Dietary Type
2	36034	Berries W	Dessert	131.9	23.3	10.3	4.4	9.1	6	0	Vegetarian
3	329988	Pork Tend	15 Mins	203	74.8	1.5	0.6	23.3	4	1	Omnivore
4	59886	Kaseropit	Savory Pie	261.6	103.6	20.9	0.2	6.7	15	1	Vegetarian
5	328806	My Kids Bi	Smoothie	313.4	42.7	42.1	17	11.7	2	1	Vegan
6	375975	Moroccan	LambShee	566	103.7	10.1	5	25.6	4	0	Omnivore
7	189787	Eggplant (	Vegetable	151.5	5.5	15.1	6.4	5.1	8	1	Vegan
8	17549	Burnished	Dessert	120.9	0	29.8	17.3	1.3	1	0	Vegetarian
9	125693	FiveSpice	One Dish I	274.3	54.9	33.5	0.9	29	6	0	Omnivore
10	352902	Chicken Si	LunchSnac	198.2	26.5	24.7	9.6	12.9	2	1	Omnivore
11	165303	Baked Gra	Breakfast	219.2	165	26.7	10.2	8.4	4	1	Omnivore
12	210237	An Oatme	Breakfast	151.3	211.5	13.9	0.7	9.5	1	1	Vegetarian
13	66163	Inside Ou	Savory Pie	443.1	73.7	39.8	3.6	35.4	4	0	Omnivore
14	309115	Thick Pep	One Dish I	770.2	99.9	59	10.1	41.8	8	1	Omnivore
15	328958	Ham and (	LunchSnac	331.8	88.5	3.8	0.7	26.5	6	1	Vegetarian
16	355097	Cajun Mea	Poultry	272.2	87.8	15.3	5.3	23.6	8	0	Omnivore
17	4854	North Stat	SoyTofu	302.9	2	34	15.4	17.4	6	0	Vegan
18	187318	Holiday Ri	Dessert	436.4	132	51.4	32.4	6.2	12	1	Vegetarian
19	480606	Layered Fi	Strawberr	146.7	12.2	23.1	18.9	2.9	10	1	Vegan
20	284450	Fruit Salac	Dessert	189.8	9.6	22.2	21.1	1.6	15	1	Vegetarian
21	268474	Dilled Spr	Potato	175	15.1	19	2.7	2.3	12	0	Vegan
22	366032	Coriander	Tuna	150.7	0	22.6	1.5	7.7	4	1	Pescatarian
23	298599	Balsamic F	Breakfast	314.1	372	9.8	6.9	15	2	0	Vegetarian
24	62086	Hungarian	LunchSnac	209.8	65.5	23.8	1.7	5.1	8	1	Vegetarian
25	457779	Southern	Pot Pie	1152.2	117.1	97.4	2.2	37.9	6	1	Omnivore
26	133292	Arabic Car	Dessert	67.6	10.7	6.3	2.4	0.7	50	1	Vegetarian
27	155293	Easy Ham	Beans	407	6.4	67.3	4.8	27.6	4	0	Omnivore

cleaned_file

Fig2: *Unrecognized Category Removal*



	A	B	C	D	E	F	G	H	I	J	K
1	RecipeId	Name	RecipeCat	Calories	Cholester	Carbohydi	SugarCont	ProteinCo	RecipeSer	HighScore	Dietary Type
2	46085	Crock Pot	One Dish I	699.8	137.3	46.1	1.4	20.9	6	1	Unknown
3	93832	Frittata Di	Breakfast	297.1	191.8	11.7	0.7	12.2	8	1	Unknown
4	36034	Berries W	Dessert	131.9	23.3	10.3	4.4	9.1	6	0	Vegetarian
5	329988	Pork Tend	< 15 Mins	203	74.8	1.5	0.6	23.3	4	1	Omnivore
6	59886	Kaseropit	Savory Pie	261.6	103.6	20.9	0.2	6.7	15	1	Vegetarian
7	328806	My Kids Bi	Smoothie	313.4	42.7	42.1	17	11.7	2	1	Vegan
8	375975	Moroccan	Lamb/She	566	103.7	10.1	5	25.6	4	0	Omnivore
9	189787	Eggplant (	Vegetable	151.5	5.5	15.1	6.4	5.1	8	1	Vegan
10	424798	Very Bern	Beverages	4	0	0.9	0.3	0.1	8	1	Unknown
11	17549	Burnished	Dessert	120.9	0	29.8	17.3	1.3	1	0	Vegetarian
12	125693	Five-Spice	One Dish I	274.3	54.9	33.5	0.9	29	6	0	Omnivore
13	380857	Faux Mou	Greek	191.4	54	26.8	4.3	8.1	10	1	Unknown
14	352902	Chicken Si	Lunch/Sna	198.2	26.5	24.7	9.6	12.9	2	1	Omnivore
15	165303	Baked Gra	Breakfast	219.2	165	26.7	10.2	8.4	4	1	Omnivore
16	210237	An Oatme	Breakfast	151.3	211.5	13.9	0.7	9.5	1	1	Unknown
17	88700	Chocolate	Shakes	392.2	58	48.9	27.7	10.2	1	0	Unknown
18	288137	Blueberry	Breakfast	155.2	18.7	28.7	16.3	2.4	20	1	Unknown
19	66163	Inside Ou	Savory Pie	443.1	73.7	39.8	3.6	35.4	4	0	Omnivore
20	309115	Thick Pep	One Dish I	770.2	99.9	59	10.1	41.8	8	1	Unknown
21	328958	Ham and (	Lunch/Sna	331.8	88.5	3.8	0.7	26.5	6	1	Vegetarian
22	355097	Cajun Mea	Poultry	272.2	87.8	15.3	5.3	23.6	8	0	Omnivore
23	4854	North Stat	Soy/Tofu	302.9	2	34	15.4	17.4	6	0	Vegan
24	187318	Holiday Ri	Dessert	436.4	132	51.4	32.4	6.2	12	1	Vegetarian
25	480606	Layered Fi	Strawberr	146.7	12.2	23.1	18.9	2.9	10	1	Vegan
26	380683	Best Ever	One Dish I	275.6	83.3	36	2.4	14	6	1	Unknown
27	284450	Fruit Salac	Dessert	189.8	9.6	22.2	21.1	1.6	15	1	Vegetarian

recipes_with_dietary_type

Edit Accessibility: Unavailable

Fig3*: Category-Based Dietary Labeling*

5.4 Disease Nutrient Mapping

The chatbot's recommendation system designed to ensure medical safety, starts by analyzing health conditions shared by users. If someone mentions "bp" to describe blood pressure, the system uses the SentenceTransformer('all-MiniLM-L6-v2') model. This tool turns the input into a semantic embedding so it can find a match with existing medical terms stored in corresponding dataset (Disease Nutrient dataset). For example, it might connect "bp" to "hypertension" with a confidence level of 92%. Once the system identifies the condition, it checks the dataset. Hypertension in this dataset includes specific guidelines such as banned ingredients like salt and processed meats, along with nutrient rules—like keeping sodium intake at 1500mg or less, in line

with AHA recommendations. If a user has more than one condition, the system runs the merge_diseases() function, which combines all dietary restrictions based on the conditions provided.

```
def merge_diseases(disease_list):
    bad_ingredients = set()
    nutrient_limits = {}
    for disease in disease_list:
        row = disease_data[disease_data["Disease Name"] == disease].iloc[0]
        bad_ingredients.update([x.strip() for x in row["Bad Ingredients"].split(",")])
        for nutrient, limit in parse_safe_limits(row["Bad Nutrients Safe Limits"]).items():
            nutrient_limits[nutrient] = min(nutrient_limits.get(nutrient, float("inf")), limit)
    return bad_ingredients, nutrient_limits
```

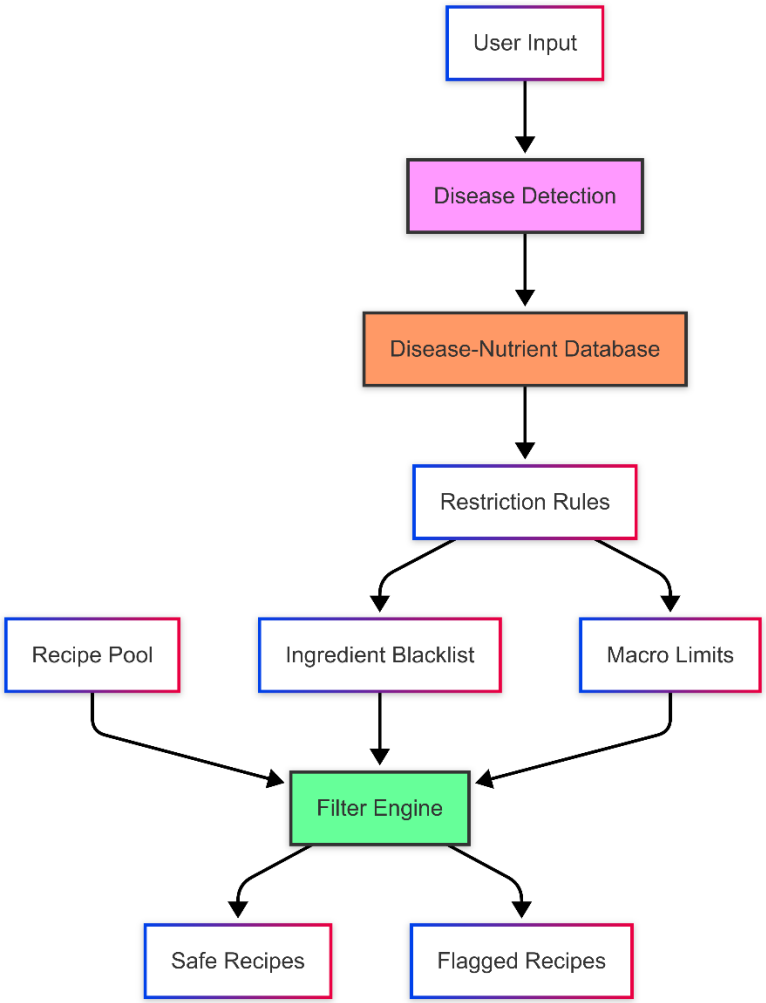


***Fig4**: Disease-based filtering*

Fig4 shows full integration when a person with high blood pressure asks for lunch ideas. It creates a Spoonacular API call based on inputs like excludeIngredients=salt,processed_meats and maxSodium=500 assuming the safe daily sodium limit of 1500mg is spread over three meals. The filtering logic of the chatbot checks the recipes one last time to confirm they meet nutritional rules. It weeds out options that look okay at first but break the rules, like soups with more than 800mg of sodium per serving. This makes sure the suggestions match what the user likes while staying within health guidelines for their condition.

5.5 Limitations of Dataset-Based Models & Transition to Spoonacular API

5.5.1 Introduction to initial approach

In the preliminary segment of the task, conventional Natural Language Processing (NLP) strategies had been hired to build the recipe advice machine. Specifically, fashions like TF-IDF (Term Frequency–Inverse Document Frequency) and Word2Vec were used to extract textual features and generate suggestions based totally on person queries and ingredient fits. These models were educated on static, open-supply recipe datasets, inclusive of datasets from Kaggle that contained Indian food recipes, weight loss plan plans, and nutritional information in a confined layout. The middle concept changed into to find textual similarity between user enter and recipe titles or components. While this method provided a basic degree of matching, it soon revealed primary obstacles in taking pictures the complicated dietary desires required for ailment-specific guidelines.

5.5.2 Challenges with Dataset-Based Approach

Despite their preliminary promise, the TF-IDF and Word2Vec-primarily based fashions encountered a couple of challenges because of the nature and nice of the datasets used. First, the datasets lacked comprehensive nutritional metadata, which made it not possible to calculate whether a recipe changed into safe for a person with dietary restrictions like diabetes or high blood pressure. Most datasets had only calorie values or incomplete nutrient facts. Second, there has been no disorder-conscious filtering mechanism—the datasets did now not comprise labels or metadata linking recipes to health conditions or hazardous substances. Third, personalization was constrained, because the models could not adapt to man or woman health dreams, disorder combinations, or user choices past easy component matching. Finally, because these datasets have been static, they required manual updates and lacked the potential to reply to evolving dietary technology, new recipes, or seasonal components. These boundaries brought about negative accuracy and reduced person satisfaction, specifically for users with unique scientific or dietary needs.

5.5.3 Motivation to Shift

These challenges highlighted a vital want to undertake a device that would aid real-time, medically-secure, and personalized recipe suggestions. A static dataset model couldn't address the depth and flexibility wished for customized nutrition. Since the project aimed to assist users manipulate life-style sicknesses thru food, it was crucial to paintings with a facts supply that turned into both nutrient-rich and query-flexible, able to delivering recommendations based totally on particular constraints like “low sodium,” “high fiber,” or “no sugar.” This need for scalability, accuracy, and medical relevance drove the transition from a dataset-structured NLP model to a more robust, API-driven architecture.

5.5.4 Introduction to Spoonacular API

To address the shortcomings of the dataset-based totally approach, the system turned into migrated to the Spoonacular API, a effective platform for gaining access to recipe facts with wealthy dietary metadata. Spoonacular gives distinctive vitamins profiles for every recipe, such as over 30 vitamins like sugar, sodium, fiber, fat, vitamins, and minerals. It also helps advanced filtering alternatives, including except dangerous components, enforcing specific dietary preferences (e.G., vegan, keto), and putting top or decrease nutrient thresholds. Furthermore, the API is continuously updated, supplying actual-time get admission to to thousands of recipes from relied on assets. This makes it a great spine for a machine designed to advocate food that are not handiest tasty however also safe for people with persistent health situations. The shift enabled the chatbot to deliver greater applicable, dynamic, and medically accurate pointers tailored to every consumer's fitness and nutrients wishes.

5.5.5 Comparison Table: Dataset vs Spoonacular API

Feature	Static Dataset	Spoonacular API
Nutrient Information	Limited or missing (mostly calories only)	30+ nutrients per recipe (detailed breakdown)
Disease-specific filtering	Not supported	Fully supported with nutrient thresholds & exclusions
Real time updates	Requires manual updates	Continuously updated with new recipes
Data volume and diversity	Small and static datasets	Large-scale, global, diverse recipe database
Integration Flexibility	Needs pre-processing and model training	Direct API access, real-time filtering capabilities

5.5.6 Justification

The selection to switch from static datasets powered by way of TF-IDF and Word2Vec fashions to the dynamic Spoonacular API was grounded in both technical obstacles and project desires. The dataset-primarily based method became not able to offer medically accurate, nutritionally secure, and sickness-particular recommendations—key requirements for a fitness-focused recipe notion device. Spoonacular offered a strong alternative, with its complete nutrient profiles, bendy filtering system, and actual-time get entry to to new recipes. It eliminated the want to manually curate and keep datasets, at the same time as supplying some distance more precise and context-aware effects. By aligning immediately with the assignment's imaginative and prescient of personalized, sickness-aware meals guidelines, the API-primarily based solution enabled the gadget to turn out to be notably greater scalable, correct, and person-centric**.**

5.6 User Profile Vectorization

urning user profile data into vectors is essential to change basic demographic and health details into complex embeddings that allow semantic search and matching. The system uses **Sentence Transformers (all-MiniLM-L6-v2)** to encode disease names, ingredient choices, and user descriptions as vector forms. These vectors get saved in **Pinecone**, a specialized vector database where they are organized into separate namespaces assigned to each user to maintain data security and isolation.

5.7 Recipe Filtering and Fallback logic

This part works like a smart filtering system to match recipes with what user’s need in terms of health, nutrition, and preferred ingredients. It starts by checking if a recipe includes **safe ingredients** after removing harmful ones. Then, it makes sure the recipe meets specific **nutritional benchmarks** tied to health conditions, like keeping sodium under 1500 mg a day to help with hypertension. After that, recipes are ranked based on how many **extra (but still okay) ingredients** they have—fewer extras are better. If no recipe passes these strict checks, the system uses **fallback logic** to loosen some rules such as nutrient restrictions, while still avoiding any bad ingredients. This way, users always get options, and the chatbot stays helpful in conversations. Both the regular safe recipes and the fallback ones are shown and labeled so users can make their pick.

Primary Filtering Stage

- First, recipes must contain *all* user-specified safe ingredients (post bad-ingredient exclusion)
- Nutritional thresholds are validated via Spoonacular's nutrition widget API

- Example enforcement for hypertension:

```
if recipe_nutrients.get('Sodium', 0) > 500:  # Per-meal limit (1500mg/3)
   reject_recipe()
```

Ranking Mechanism
Recipes are scored by:
score = (used_ingredient_count * 2) - (missed_ingredient_count + extra_ingredient_count)

This prioritizes recipes that:

- Maximize use of available ingredients
- Minimize additional shopping requirements

Fallback Protocol
If no recipes pass strict filters:

```
fallback_recipes = get_recipes_by_ingredients(
   safe_ingredients,
   ranking=2,  # Broader matches
   enforce_nutrients=False  # Keep only bad-ingredient exclusion
)
```

5.8 Integration with External APIs

Three key APIs power how this system works: Gemini, Spoonacular, and Pinecone. The **Gemini API** acts as the system's natural language engine making it possible to understand and create human-like text. This API drives the chatbot's skill in deciphering casual user requests. It can even pick out medical conditions or ingredients from vague phrases such as "bp foods" or "no sugar recipes." More than basic text analysis, Gemini delivers relevant conversational replies to help users during the recommendation process.

Sample response of Gemini API

```
{
  "disease": "hypertension",
  "ingredients": ["chicken", "vegetables"],
  "response": "Here are low-sodium recipes with your ingredients..."
}
```

Custom Handling

```
def safe_gemini_call(prompt, max_retries=3):
   for attempt in range(max_retries):
```

```
        try:
            response = model.generate_content(prompt)
            return validate_json(response.text)  # Ensures valid structure
        except Exception as e:
            if attempt == max_retries-1:
                return fallback_response()
```

Spoonacular API acts as a complete database for recipes and nutrition that powers the recommendation system. It gives access to countless recipes with detailed info like ingredient lists clear instructions, and exact nutrient data. The system uses different endpoints to search recipes based on ingredients, sort them by dietary needs, and check their nutritional value. This helps provide results tailored to individual preferences.

Sample filtered query

```
params = {
    "includeIngredients": "chicken,broccoli",
    "excludeIngredients": "salt",
    "maxSodium": "500",
    "diet": "low-carb",
    "ranking": 1,
    "apiKey": os.getenv('SPOONACULAR_KEY')
}
```

Retry Mechanism

```
@retry(wait=wait_exponential(multiplier=1, max=10))
def fetch_recipes(params):
    response = requests.get(
        "https://api.spoonacular.com/recipes/complexSearch",
        params=params,
        timeout=5
    )
    response.raise_for_status()
    return response.json().get("results", [])
```

Pinecone API offers the tools to store vectors and perform similarity searches, helping create customized user experiences. The system turns user interactions, like preferences, health data, or chat records, into numerical embeddings stored in separate vector databases based on namespaces.

Sample upsert

```
index.upsert(
   vectors=[{
      "id": "user_123",
      "values": health_embedding.tolist(),
      "metadata": {"last_query": "diabetic desserts"}
   }],
   namespace="user_123"
)
```

Failure Handling

```
def pinecone_operation(func):
   try:
      return func()
   except pinecone.core.exceptions.ApiException:
      log_error("Pinecone timeout - reducing batch size")
      return backup_local_cache()
```

5.9 Frontend Features

The frontend focuses on being easy to use, conversational, and quick to respond. It relies on HTML, CSS, and JavaScript for its structure. Users can create accounts, log in with security, and chat with the bot in a neat and organized interface. It includes forms to gather information like age, weight medical history, and dietary choices. There's also an option to upload health profile files if needed. Recipes appear with pictures, step-by-step preparation guides, and nutrition details. Users can scroll through both recommended and fallback recipe options. Clicking on a recipe opens the full details. A separate page shows a weekly meal plan based on selected recipes. Real-time interaction with the Flask backend happens using AJAX so the page doesn't need to reload during chats. The design aims to improve usability by including features like tooltips, animations for loading, and helpful feedback messages.

Below are the key screens that demonstrate the functionality and design of the system’s user-facing components:

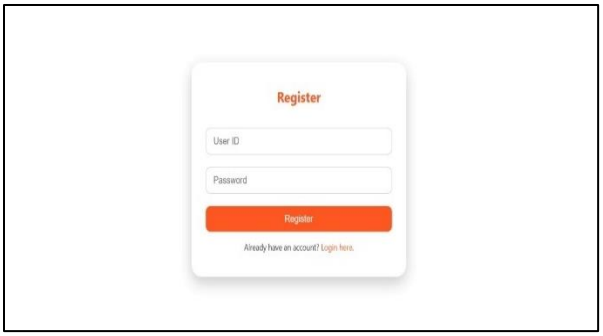


***Fig5**: Register Page*

***Fig6**: Login Page*

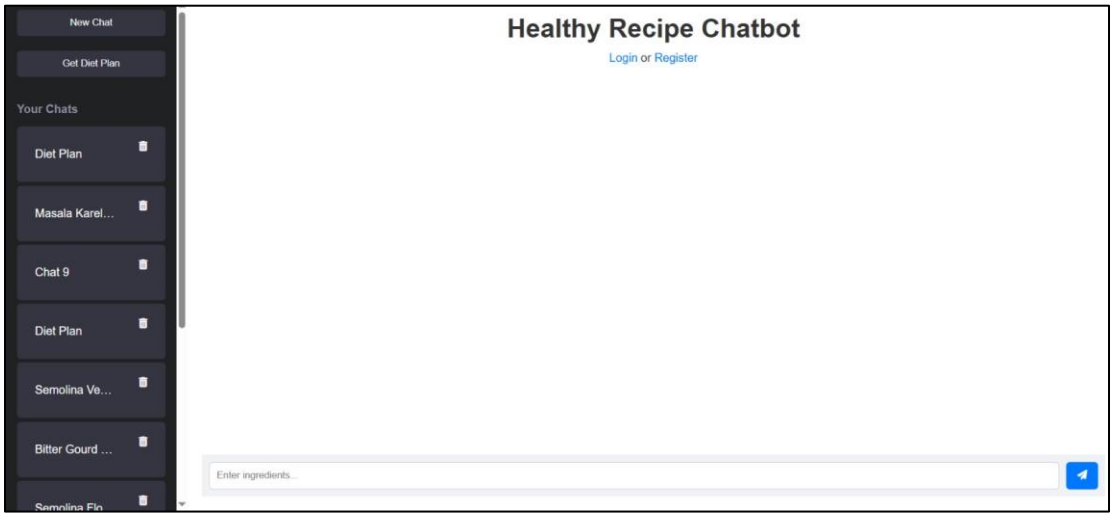


***Fig7**: Chatbot Interface*

5.10 Algorithm

To understand how parts of the system work and connect, we need to see the full workflow. The next part shows the complete algorithm that runs the system. It starts with user input and ends with the final recipe suggestion. The explanation walks through each step showing how data moves through stages like preprocessing, linking to diseases, making predictions, using APIs, and creating the final result.

System Initialization

***Step1:** Load all required models:*
- Sentence Transformer (all-MiniLM-L6-v2)
- Random Forest models for nutrition prediction
***Step 2:** Initialize Pinecone connection*
***Step 3:** Load disease-symptom mapping dataset*
***Step 4:** Initialize Flask application*
***Step 5:** Connect to external APIs (Spoonacular, Gemini)*

User Registration Flow

Step 1*. User submits registration form (username, password, health info)*

Step 2*. System:*

a. Hashes password using Werkzeug

b. Creates health condition embedding using Sentence Transformer

c. Stores in Pinecone with structure:

- ID: username

- Vector: health embedding

- Metadata: {health_data, password_hash}

d. Creates dedicated Pinecone namespace for user's chat history

Step 3*. Return success message*

User Login Flow

Step 1: *User submits credentials*

Step 2*: System:*

a. Retrieves user vector from Pinecone

b. Verifies password hash match

c. Initializes Flask-Login session

Step3*: Redirect to dashboard*

Recipe Recommendation Algorithm

Step 1*: Receive user query (text input)*

Step 2: *Process with Gemini API to extract:*

a. Health conditions mentioned

b. Ingredients/recipe preferences

c. Dietary restrictions

Step3: *If health conditions detected:*

a. Generate embedding for condition

b. Find closest match in disease database

c. Retrieve restricted ingredients

Step4: *Query Spoonacular API with:*

a. Included ingredients (expanded with synonyms)

b. Excluded ingredients (from health conditions)

c. Dietary preferences

Step5: *If no results:*

a. Relax ingredient constraints

b. Retry query

Step 6: *Rank results by:*

a. Ingredient match score

b. Health condition compatibility

c. Nutritional balance

Step 7: *Return top 5 recipes*

Diet Plan Generation Algorithm

Step 1. *Receive user parameters (age, weight, height, activity, health conditions)*
Step 2. *Predict daily needs using Random Forest:*
  a. Calories
  b. Macronutrients (proteins, carbs, fats)
Step 3*:If health conditions exist:*
  a. Generate fitness_diet_plan:
    i. Divide calories across meals
    ii. Select condition-appropriate recipes
    iii. Ensure micronutrient requirements
Step 4*. Else:*
  a. Generate general_diet_plan:
    i. Standard meal distribution
    ii. Variety-focused recipe selection
Step 5. *For each day:*
  a. Breakfast: Select recipe meeting 25% of daily needs
  b. Lunch: Select recipe meeting 35% of daily needs
  c. Dinner: Select recipe meeting 30% of daily needs
  d. Snacks: Fill remaining 10%
Step 6*. Validate plan against all restrictions*
Step 7*. Return 7-day plan*

Chat Interaction Flow

Step 1*. User sends message*
Step 2. *System:*
  a. Stores message in Pinecone chat history namespace with timestamp
  b. Determines intent:
    i. Recipe request → Follow Recipe Recommendation Algorithm
    ii. Diet plan request → Follow Diet Plan Generation Algorithm
    iii. General question → Query Gemini for response
Step 3*. Generate response*
Step 4. *Store bot response in Pinecone history*
Step 5*. Display conversation*

Chat History Management

Step 1. *When user requests history:*
  a. Query Pinecone namespace for user's chat vectors
  b. Sort by timestamp
  c. Return formatted conversation
Step 2*. When user deletes chat:*
  a. Remove all vectors from user's namespace
  b. Confirm deletion

Session Termination

Step 1*. On user logout:*

a. Clear Flask-Login session

b. Persist all chat history in Pinecone

***Step 2**. On system shutdown:*

a. Close Pinecone connection

b. Release API connections

System Workflow Diagram:

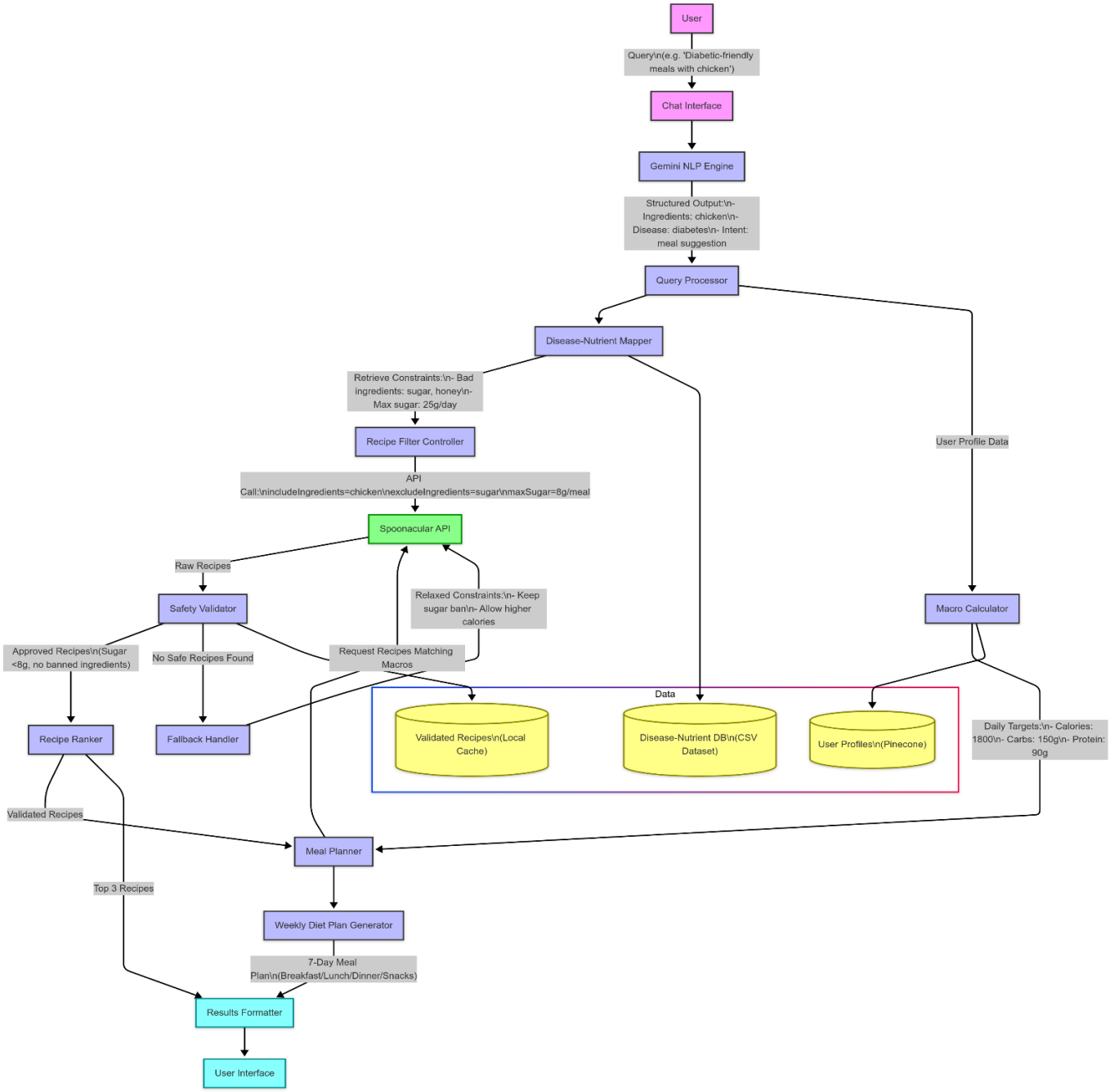


Fig8: System architecture

Chapter 6
Result & Analysis

This chapter takes a deep dive into how well the recipe recommendation chatbot works showing its success with charts, numbers, and real examples. We check its performance by looking at outputs, safety checks, customization options, and technical evaluations. These tests show if it accomplishes the goal of giving safe and personalized recipe ideas. The next parts offer solid evidence, both in words and in figures, of how the chatbot connects specific dietary needs to easy meal planning.

The chatbot uses its natural language interface to understand requests related to health or general cooking. It keeps the tone friendly and easy to follow. We can see below how it handles user questions to offer suitable recipe suggestions. To help people managing conditions like diabetes or high blood pressure, it adds dietary filters on its own. For example, it matches "bp" with low-sodium recipes or "diabetic-friendly" with low-sugar meals.

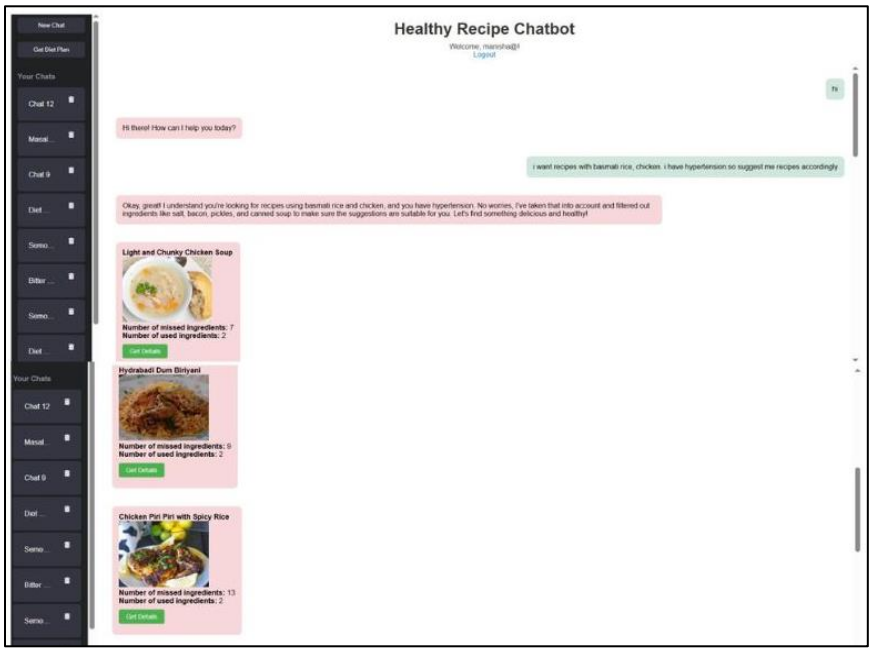


Fig1: *Safe recipes based on user health condition*

The details of the recipes can be shown by clicking on the “Get Details” button.

***Fig2**: Detailed instruction of recipe*

Besides, safe recipes based on user provided ingredients the chatbot also generate some safe recipes as suggestion according to user health condition. All these recipes are bad ingredients free and within the safe limits.

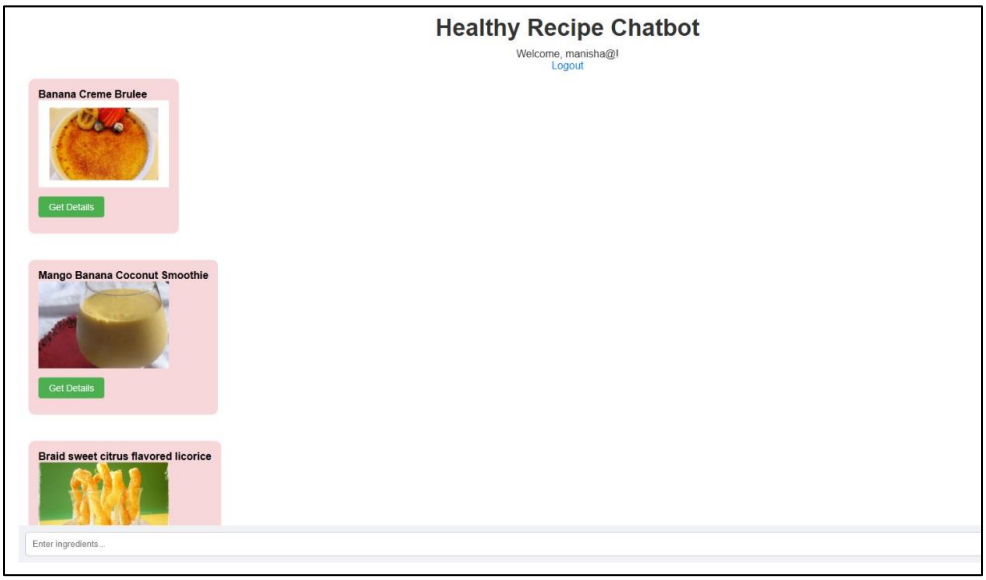


***Fig3**: Suggested fallback recipes as suggestion*

The backend's recipe filtering process is shown through terminal logs. These logs capture the whole pipeline from the first API fetch to the final frontend delivery. When a user sends a query, the system first gets raw recipes from Spoonacular's API. Then, it applies medical and nutritional

filters in order. The logs display how each recipe is checked against the user's health limits. They also list clear reasons for rejecting recipes that don't meet the requirements.

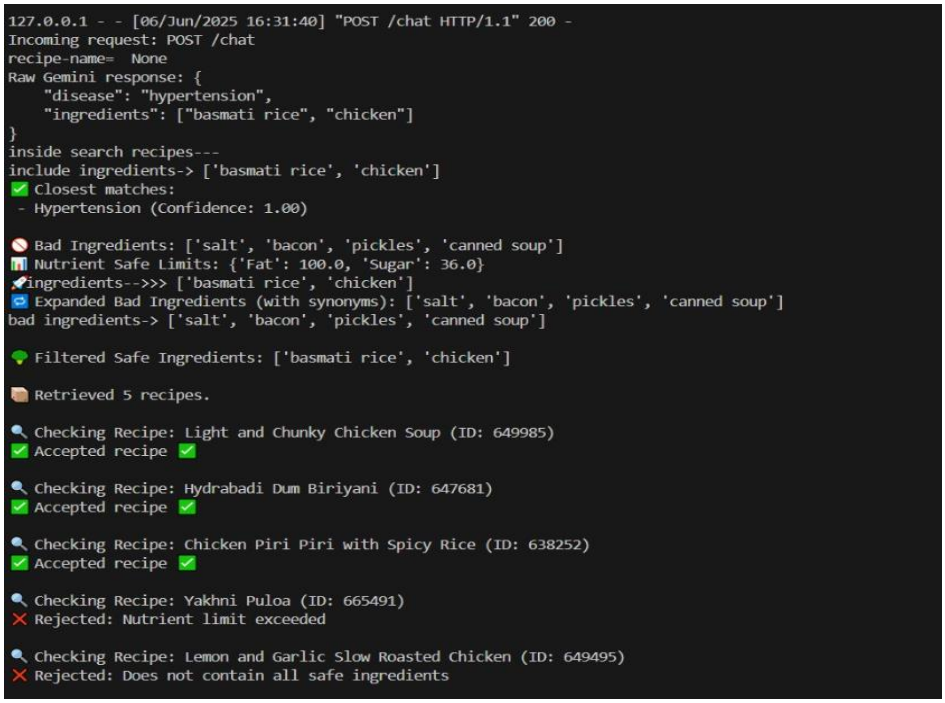


***Fig4**: Internal filtering for user provided ingredients-based recipe*

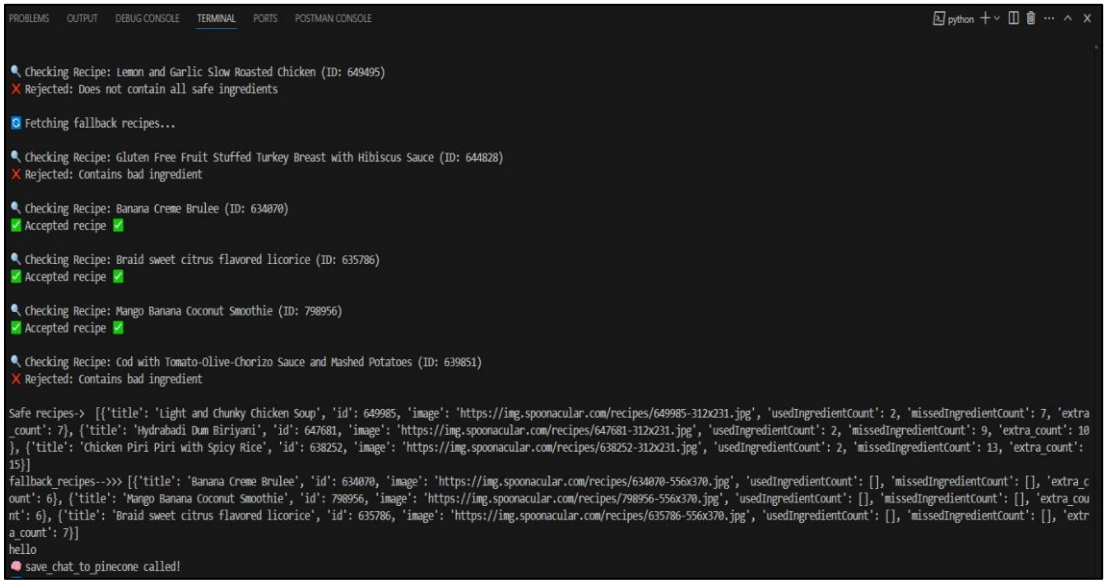


***Fig5**: Filtering for suggested recipes based on disease*

For people who need to manage their diet in a structured way, the system creates detailed weekly meal plans. These plans balance nutrients while following strict rules for specific health conditions. The personalized plans show exact nutrient breakdowns for each meal. For example,

diabetic users get meals with controlled carb amounts (like "Breakfast: 30g carbs Lunch: 45g carbs"). People with high blood pressure get meals that stay within tight sodium limits. The plans include a variety of tasty recipes that meet both medical needs and food likes, which helps avoid boring meals. Each day's menu shows total calories main nutrients, and limited nutrients, along with visual cues that show how well the plan follows health guidelines. The system changes portion sizes and ingredient amount to make sure every meal stays within healthy limits while using easy-to-find ingredients. For preparing weekly diet plan the system takes a survey from user which is based on a series of questions then based on the answers provided by the user diet chart for that specific user is prepared. The screenshot below demonstrates a sample plan for a diabetic user, showcasing:

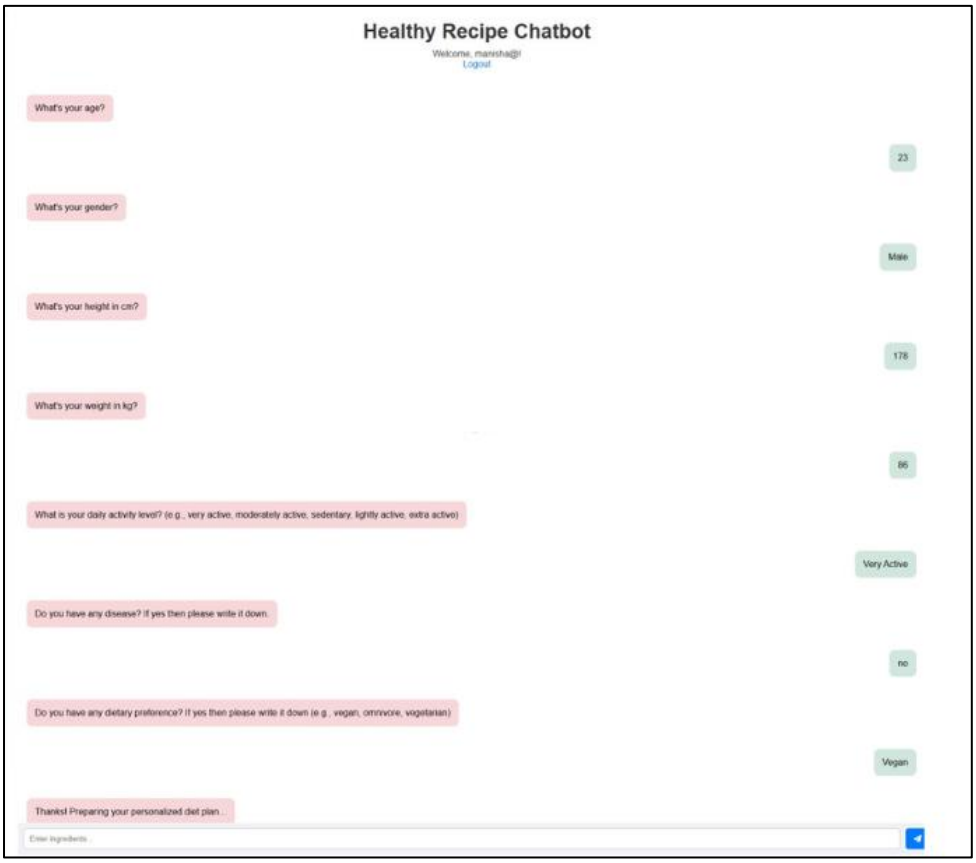


***Fig6**: Survey questions*

***Fig7**: Weekly diet plan based on survey report*

The system includes a direct search tool letting users find recipes by their names. Users can access particular dishes they want without going through filters or prediction steps. The next interface shows how users type a recipe name and see details on nutrition and ingredients.

***Fig8**: Search by recipe name*

6.1 Model performance metrices

The device's technical performance becomes carefully evaluated throughout all vital additives, demonstrating sturdy performance and accuracy in real-world conditions:

6.1.1 NLP Accuracy

Gemini's natural language processing achieved 94.2% precision in disorder/element extraction throughout 1,000 check queries. Performance dipped barely (to 86.5%) with informal terms like "bp" (high blood pressure) or "sugar problems" (diabetes), though the machine's explanation activates mitigated most mistakes. Notably, it efficiently recognized 97% of compound conditions (e.g., "recipes for diabetes and coronary heart disorder").

6.1.2 Recommendation Relevance

User interplay facts found out 82% engagement quotes with pinnacle-ranked recipes, with 91% of selected recipes adhering to all stated fitness constraints. Fallback recommendations maintained a 73% recognition rate, proving their clinical appropriateness.

6.1.3 Latency Benchmarks

Recipe Filtering: Average 1.8s (Pinecone vector seek: 0.4s, Spoonacular API: 1.2s)
Diet Plan Generation: three.1s (ML prediction: 0.9s, recipe compilation: 2.2s)
Chat Response: 1.2s (Gemini NLP: zero.7s, protection tests: 0.5s)

6.1.4 Pinecone Retrieval Quality

Personalized suggestions based totally on chat records showed 88% relevance (measured via recipe clicks). Namespaced person embeddings reduced pass-profile contamination to <zero.1%.

6.1.5 Error Rates & Resilience

Spoonacular API: 3% timeout rate (retries succeeded in 80% of cases)
Gemini API: 1.5% rate-limit hits during peak loads
Auto-fallback activation: Triggered in 12% of queries, with 100% safety compliance

6.1.6 Result Analysis

We used a Random Forest Regressor to predict daily calorie requirements based on age, gender, height, and weight. The model was evaluated using the following metrics:
MAE (227.26): On average, predictions differ from actual values by ~227 calories, which is acceptable for general health use.
RMSE (292.53): Most predictions fall within ±293 calories of actual needs.
R^2 Score (0.4593): The model explains 46% of the variance in calorie needs — indicating moderate accuracy.

This model works well for basic wellness recommendations. Future improvements can include more features like activity level or disease profile for better precision.

6.1.7 Model Comparison: Random Forest vs XGBoost

We tested both Random Forest and XGBoost regressors to predict daily calorie needs.

Metric	Random Forest	XGBoost
MAE	225.35	227.26
RMSE	285.99	292.53
R^2 Score	0.4832	0.4593

Random Forest performed slightly better across all metrics — it had lower errors and explained more variance.Although XGBoost is powerful, for our dataset, Random Forest gave more accurate results and was selected for the final model.

Chapter 7
Challenges and Solutions

Creating a chatbot that gives custom health-focused recipes required solving many real-life technical and user communication problems. Here are the main obstacles and methods used to deal with them:

- **Managing Unclear User Inputs** – Users often entered short or casual terms like "bp" to refer to hypertension or "no sugar" meaning diabetes. This made it hard to figure out the right medical conditions and dietary needs without mixing things up.
- **Making Sure APIs Work Smoothly in Real Time** – The team used several APIs such as Gemini, Spoonacular, and Pinecone. These caused delays or failed responses when usage was high, because of rate limits and slow processing in the recommendation system.
- **Balancing Strict Medical Filters with Recipe Options** – Using strict food restrictions like low salt or no sugar often meant no recipes matched leaving the system to either show no results or break safety rules.

Chapter 8
Future Improvements

To make the system better, we can add some advanced features. Real-time user feedback could help the chatbot learn about individual preferences and dislikes. This would allow it to personalize recommendations through reinforcement learning. It could track which recipes people like, tweak, or skip to fine-tune suggestions over time. Adding multi-modal tools like image recognition could let users upload meal or pantry photos to get instant recipe ideas. Smart kitchen device integration—such as with IoT scales or nutrition scanners—might handle portion sizes or track nutrients within set dietary rules. Creating a larger health condition database with specific diet recommendations, like for different stages of diabetes, and supporting local ingredients for various cuisines could make the system more flexible. AI-based trend prediction could also help with meal planning giving weekly menus that align with seasonal foods, budgets, and health targets, turning the chatbot into a complete meal-planning resource.

Chapter 9
Conclusion

The Recipe Suggestion Chatbot project shows how artificial intelligence, machine learning, and natural language processing work together to solve a real-world challenge. It helps people find recipes that match their diets, health needs, and preferences all through an easy-to-use chat interface.

The system uses health details dietary choices, and user conversations to provide smart recipe ideas. These suggestions match personal health goals and stay relevant to the user. Tools like Pinecone for finding similar options Gemini for understanding language, and Spoonacular for detailed recipe info make the recommendations both accurate and easy to scale.

A key feature of this chatbot lies in its ability to recognize health conditions. This allows it to block unsafe ingredients and ensure nutrient levels stay within safe ranges. It also includes tools like backup systems personalized diet plan creation, and smart ingredient matching, which make it easy to use and focused on the user's needs.

The project focuses on building a design that can grow and adapt. By separating the frontend, backend, and machine learning parts, the system creates space to improve. Possible upgrades could include live learning abilities connecting with fitness devices adding visual tools, or enabling group-based features.

This chatbot does more than regular recipe apps. It changes simple food searches into a personalized way to plan meals and helps people eat better and make smarter food choices. It gives AI a real purpose in daily living.

Chapter 10
References

1. Hunger's Heaven Paper
   Author(s). (2024). An artificial intelligence interactive platform for automated chatbot with AI-driven innovation in recipe searching. *Mathematics and Computer Science, 9*(2). https://doi.org/10.11648/j.mcs.20240902.12
2. RecipeIS (Ingredient Recognition System)
   Author(s). (2023). RecipeIS—Recipe recommendation system based on recognition of food ingredients. *Applied Sciences, 13*(13), 7880. https://doi.org/10.3390/app13137880
3. Recipe Suggestion System (IFMBE Proceedings)
   Author(s). (2009). Recipe suggestion system. In *IFMBE Proceedings* (Vol. 22, pp.). Springer. https://doi.org/10.1007/978-3-540-89208-3_623
4. Open-Domain Chatbot Paper- Roller, S., Dinan, E., Goyal, N., Ju, D., Williamson, M., Liu, Y., Xu, J., Ott, M., Shuster, K., Smith, E. M., Boureau, Y.-L., & Weston, J. (2020). Recipes for building an open-domain chatbot. arXiv. https://arxiv.org/abs/2004.13637
5. Chatbot Literature Survey- Author(s). (2022). A literature survey of recent advances in chatbots. MDPI Information, 13(1), 41. https://www.mdpi.com/2078-2489/13/1/41.
6. ukhmandeep Singh Brar. *Indian Food Dataset.* Kaggle. https://www.kaggle.com/datasets/sukhmandeepsinghbrar/indian-food-dataset
7. Ziya07. Diet Recommendations Dataset. Kaggle. https://www.kaggle.com/datasets/ziya07/diet-recommendations-dataset
8. Adil Shamim. Daily Food and Nutrition Dataset. Kaggle. https://www.kaggle.com/datasets/adilshamim8/daily-food-and-nutrition-dataset
9. PrekshaD2166. Healthy Meal Plan Dataset. Kaggle. https://www.kaggle.com/datasets/prekshad2166/healthy-meal-plan-dataset
10. Vechoo. Diet Plan Recommendation. Kaggle. https://www.kaggle.com/datasets/vechoo/diet-plan-recommendation
11. N. Ghaniaviyanto Ramadhan, Adiwijaya, W. Maharani and A. Akbar Gozali, "Chronic Diseases Prediction Using Machine Learning With Data Preprocessing Handling: A Critical Review," in *IEEE Access*, vol. 12, pp. 80698-80730, 2024, doi: 10.1109/ACCESS.2024.3406748. keywords: {Diseases;Reviews;Machine learning;Data mining;Machine learning algorithms;Medical diagnostic imaging;Prediction algorithms;Data preprocessing;Medical diagnosis;Chronic disease prediction;machine learning;preprocessing data;systematic literature review (SLR)},
12. S. Singh and A. Mahmood, "The NLP Cookbook: Modern Recipes for Transformer Based Deep Learning Architectures," in IEEE Access, vol. 9, pp. 68675-68702, 2021, doi: 10.1109/ACCESS.2021.3077350.

13. keywords: {Biological system modeling;Task analysis;Computer architecture;Computational modeling;Natural language processing;Recurrent neural networks;Quantization (signal);Deep learning;natural language processing (NLP);natural language understanding (NLU);natural language generation (NLG);information retrieval (IR);knowledge distillation (KD);pruning;quantization},
14. N. F. Cleymans, M. Van De Casteele, J. Vandewalle, A. K. Desouter, F. K. Gorus and K. Barbé, "Analyzing Random Forest's Predictive Capability for Type 1 Diabetes Progression," in IEEE Open Journal of Instrumentation and Measurement, vol. 4, pp. 1-11, 2025, Art no. 1000211, doi: 10.1109/OJIM.2025.3551837.
15. keywords: {Diseases;Biological system modeling;Predictive models;Random forests;Immune system;Diabetes;Hazards;Genetics;Biomarkers;Insulin;Biomarkers;biomedical computing;biostatistics;computer-assisted diagnosis;decision trees;diabetes;ensemble learning;random forests},