MAR IVANIOS COLLEGE (AUTONOMOUS)

Mar Ivanios Vidya Nagar, Nalanchira

Thiruvananthapuram – 695015

B.Sc. Computer Science Major Project Report

GENERATIVE AI-BASED RECIPE GENERATOR AND COOK ASSISTANT

A report submitted in partial fulfilment of the requirement for the award of

B.Sc. Computer Science

SUBMITTED BY

Savio Sebastian	**2220826**
Adith R.S	**2220811**
Keerthana. S	**2220805**

Under the guidance of

Dr. M R Vinodh

DEPARTMENT OF COMPUTER SCIENCE

B.Sc. Computer Science

2025

MAR IVANIOS COLLEGE (AUTONOMOUS)

Mar Ivanios Vidya Nagar, Nalanchira

Thiruvananthapuram – 695015

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project entitled "**GENERATIVE AI-BASED RECIPE GENERATOR AND COOK ASSISTANT**" is a bonafide record of the work done by **Savio Sebastian** (2220826), **Adith.R.S** (2220811), **Keerthana.S** (2220805), in partial fulfilment of the requirements for the award of the Degree of Bachelor of Science in Computer Science by the University of Kerala.

INTERNAL GUIDE HEAD OF THE DEPARTMENT

EXTERNAL EXAMINERS

1.

2.

ACKNOWLEDGEMENT

With the profound gratitude to the **ALMIGHTY** I take this chance to thank people who helped us to complete this project.

We sincerely thank our parents for their unwavering support, encouragement, and belief in my abilities, which have been my greatest source of strength. This project stands as a testament to their endless love and inspiration.

We are thankful to Principal **Dr. Meera George,** who gave us the platform to establish ourselves to reach greater heights.

We earnestly thank **Dr. K. OOMMACHAN**, Director, who always encourages us to do novel things and providing all facilities.

We express our sincere gratitude to **Ms. Tinu C. Philp**, Head, Department of Computer Science, for her valuable guidance and support to execute all incline in learning.

It is our delight to thank our project guide **Dr. M.R.Vinodh**, Assistant Professor, Department of Computer Science for his help, support, encouragement, suggestions and guidance throughout the development phases of the project.

We convey our gratitude to all the faculty members of the department who extended their support through valuable comments and suggestions during the reviews.

A great note of gratitude to friends and people who are known and unknown for us and helped in carrying out this project work a successful one.

Savio Sebastian

Adith R.S

Keerthana. S

ABSTRACT

The Generative AI-Based Recipe Generator and Cook Assistant is an intelligent web application designed to simplify the cooking experience by providing personalized recipe suggestions. Users can input ingredients via text or images, and the system leverages Convolutional Neural Networks (CNNs) for ingredient recognition. A pre-trained generative AI model then creates customized recipes based on the identified ingredients. The model achieves an accuracy of 97%, with precision of 97.8% and recall of 95.5%, ensuring high reliability in ingredient classification and recipe generation. This approach helps users discover new recipes, minimize food waste, and accommodate dietary preferences. The application enhances user experience by offering detailed cooking instructions, smart meal planning, and real-time assistance. Future improvements will focus on smart kitchen integration, multilingual support, and AI-driven customization, making the platform a versatile and interactive cooking companion.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

HTML	Hypertext Markup Language
SSD	Solid State Drive
RAM	Random Access Memory
CPU	Central Processing Unit
GUI	Graphical User Interface
CNN	Convolutional Neural Network
AI	Artificial Intelligence
VQGAN	Vector Quantized Generative Adversarial Networks
CLIP	Contrastive Language-Image Pretraining
VIT	Vision Transformer
NLP	Natural Language Processing
RNN	Recurrent Neural network
AWS	Amazon Web Service
AJAX	Asynchronous JavaScript and XML
JSON	Java Script Object Notation
XML	Extensible Markup Language
OpenCV	Open-Source Computer Vision Library
ML	Machine Learning
CSS	Cascading Style Sheet

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The Generative AI-Based Recipe Generator and Cook Assistant is a web application designed to make cooking fun and stress-free. With this smart assistant, you can easily find and create recipes that suit your needs, match what you have in your kitchen, or help you try something new. Whether you're an experienced cook or just starting, this tool simplifies your time in the kitchen and helps you enjoy the process.

One of its best features is how flexible it is. You can type in a list of ingredients you have, upload a photo of your food or Food cabinet, or just name a dish you're interested in. The tool uses advanced AI to recognize what you've shared and quickly suggests recipes based on your input. This is ideal for making the most of your leftovers, experiment with new ideas, or recreate a dish you love.

This recipe generator also helps you every step of the way. It provides clear instructions, useful tips, and guidance to make sure your dish turns out great, even if it's your first time trying it. Plus, it can adjust to special diets, food allergies, or specific cuisines, so it's a helpful companion for all types of cooking needs.

With this tool, cooking becomes easy, enjoyable, and creative. It encourages you to try new recipes, make the most of what you already have, and take pride in preparing meals. Whether you're whipping up a quick dinner or planning a special feast, the Generative AI-based recipe Generator and Cook Assistant turns cooking into an exciting and rewarding experience.

1.1 PURPOSE OF THE PROJECT

- **Image Recognition:** The app recognizes food items from images using machine learning, making it easier to identify ingredients and suggest recipes.
- **Reduce Food Waste:** Address the issue of food waste by helping users make the most of what they have in their kitchen. The AI analyzes leftover ingredients and provides practical recipe suggestions.
- **Personalized Recipe Generation:** The app creates custom recipes based on the ingredients provided, giving users customized cooking instructions.

- **Support Dietary Needs:** Adhere to individual dietary preferences, allergies, and cultural requirements. The project ensures that the recipes provided are inclusive, offering solutions that align with specific health goals or restrictions, making it ideal for diverse users.
- **Integrate Smart Solutions:** Use AI-powered features like ingredient recognition, image analysis, and smart recipe generation to seamlessly merge technology with everyday cooking. This integration creates a modern, intuitive tool that bridges the gap between advanced technology and practical kitchen needs.

1.2 OBJECTIVE

The key purpose of this project is the development of a web application that uses advanced AI models to help users in the kitchen; the key area of focus of the app revolves around ease of cooking, wherein recipes will be generated based on the ingredients a user has on hand, either text or images. The app will identify food items by using machine learning models like Convolutional Neural Networks (CNNs) for image recognition, to recommend recipes.

CHAPTER 2: LITERATURE SURVEY

Recipe Generation Based on Food Image Using Machine Learning [1]. This paper discusses the use of machine-learning techniques to generate recipes from food images. The author focuses on how computer algorithms can be trained to recognize various food items within an image. Once the food items are identified, the system uses this information to suggest relevant recipes based on the ingredients present in the image. The study demonstrates the potential of using technology to simplify meal preparation by making recipe suggestions based on simple inputs, such as a photo of the ingredients.

FIRE: Food Image to Recipe Generation [2]. This research presents the FIRE system, which is designed to convert food images into recipe suggestions. The system uses deep learning models, particularly convolutional neural networks (CNNs), to analyze food images and identify the ingredients. Once the ingredients are recognized, the model generates a recipe based on the detected ingredients, enabling users to easily find recipe ideas just by uploading a photo of the food. This paper focuses on improving the accuracy of food image recognition and recipe generation using AI.

Recipe Generation from Food Images Using Deep Learning [3]. This paper explores how deep learning techniques can be applied to generate recipes from food images. The research emphasizes training deep learning models to detect and understand different food items from images and generate a corresponding recipe. The study highlights the challenges in creating a system that can accurately identify various ingredients and then use that information to create detailed and appropriate cooking instructions. This helps users who may not know what to cook with the ingredients they have on hand.

The Multimodal and Modular AI Chef: Complex Recipe Generation from Imagery [4]. This paper presents an AI-powered system called the “AI Chef,” which can generate complex recipes from food images. Unlike simpler models, the AI Chef integrates multiple data sources, such as both text and image inputs, to create detailed recipes. The system can combine food images with ingredient lists to suggest not just basic dishes but also more complex meals that require a variety of ingredients. The research shows how multimodal input (images, text) can improve recipe generation, offering more accurate and diverse recipe suggestions.

Recipe Recommendation Using Image Classification [5]. This research explores the use of image classification for recommending recipes. The study shows how image recognition can be used to classify food items in images and then suggest recipes based on those items. The focus is on recognizing the food within the image and understanding its characteristics to recommend suitable recipes. The paper emphasizes the efficiency of using image classification to quickly identify ingredients and find recipes, thus making meal planning easier for users.

Multimodal Feature Fusion and Exploitation with Dual Learning and Reinforcement Learning for Recipe Generation [6]. This paper introduces a sophisticated method that combines different types of input, such as food images and text descriptions, for recipe generation. The study focuses on using dual learning and reinforcement learning techniques to improve the accuracy of recipe suggestions. By combining visual and textual data, the system can generate better recipes that fit the available ingredients. The research highlights how advanced learning techniques can enhance recipe generation, making it more adaptable to various types of input.

Food Text-to-Image Synthesis Using VQGAN and CLIP [7]. This research explores the synthesis of food images from text descriptions using VQGAN (Vector Quantized Generative Adversarial Networks) and CLIP (Contrastive Language-Image Pretraining). The goal of this study is to convert written descriptions of food into visual representations. By doing so, the paper demonstrates how text-based recipe descriptions can be transformed into images, helping users understand what the dish might look like. This approach could be useful for enhancing recipe generation systems by linking text input with visual outputs.

A Rich Recipe Representation as Plan to Support Expressive Multi-Modal Queries on Recipe Content and Preparation Process [8]. This paper discusses a detailed approach to representing recipes in a way that supports various types of input, including images, text, and even voice commands. The system allows users to interact with recipes in a flexible and dynamic way, depending on what input they provide. The focus of the study is to make the recipe search and cooking process more expressive and interactive, using multiple types of queries. This model could be used to improve the user experience of recipe generation systems by supporting more complex forms of interaction.

Self-Attention and Ingredient-Attention Based Model for Recipe Retrieval from Image Queries [9]. This research introduces a model that uses attention mechanisms, specifically self-attention and ingredient-attention, to improve recipe retrieval from images. Self-attention helps the model understand the entire image, while ingredient-attention focuses on recognizing and understanding individual ingredients. This allows the system to match images with appropriate recipes more effectively. The study shows how attention mechanisms can be used to create a more accurate and useful recipe suggestion system.

Device, Method, and System for Recipe Recommendation and Recipe Ingredient Management [10]. This patent discusses a system designed to recommend recipes based on the ingredients users have. It also provides a way to manage and track ingredients, helping users organize their kitchen items. The system simplifies meal planning by suggesting recipes that make use of available ingredients, making it easier for users to cook with what they already have. This idea of ingredient management can be incorporated into modern recipe generation systems to improve their usefulness and efficiency.

2.1 Summary

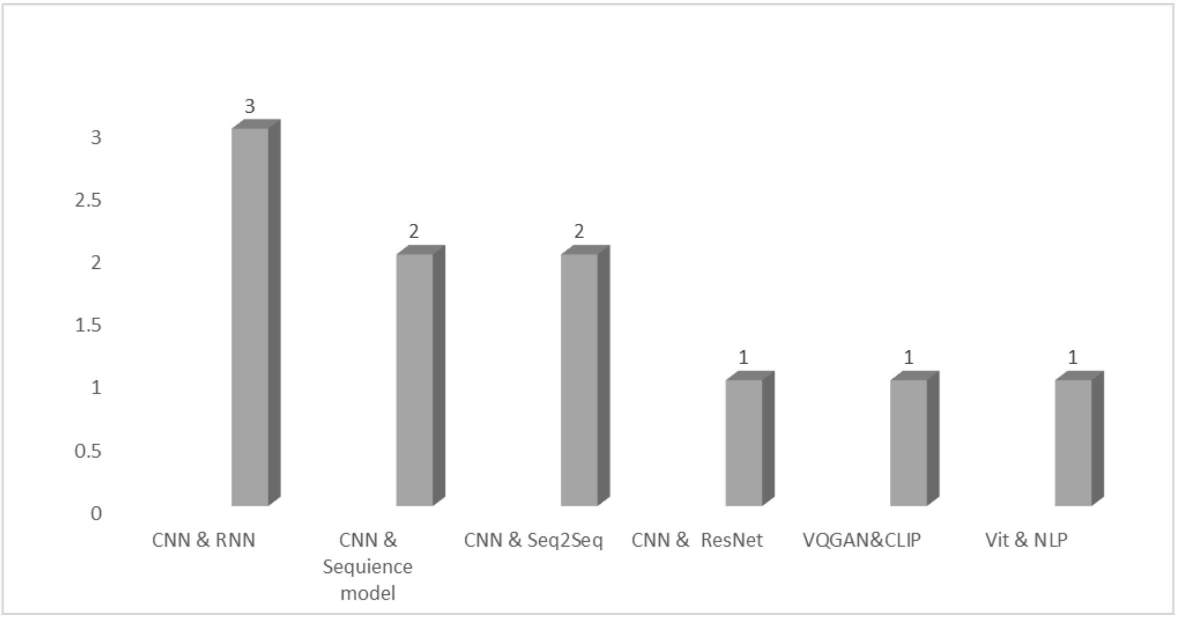


Figure 2.1 Recipe Generation Models based on conventional Algorithms

From Figure 2.1, we observed that the combination of CNN(Convolutional Neural Networks) and RNN(Recurrent Neural Network) stands out as the most prominent model pairing, highlighting its widespread use for tasks involving sequential data and visual inputs. CNN(Convolutional Neural Networks) integrated with sequence models and Seq2Seq approaches follows closely, showcasing their effectiveness in tasks that require generating or processing sequential outputs from images. Combinations like CNN(Convolutional Neural Networks) & ResNet and VQGAN(Vector Quantized Generative Adversarial Networks) & CLIP(Contrastive Language-Image Pretraining) show moderate adoption, likely due to their strengths in deep feature extraction and creative generative tasks, respectively. ViT(Vision Transformer) & NLP(Natural Language Processing) has the lowest representation, which may point to its limited adoption or its emerging significance in blending Vision Transformers with natural language processing. Overall, CNN-based models continue to dominate, while transformer-based approaches appear to play a more specialized role. CNN & RNN, Seq2Seq, and similar models face challenges like vanishing gradients, limited global context, and task-specific limitations. In contrast, CNN and Transformer models excel in handling multimodal tasks with better scalability, versatility, and contextual understanding. "

The combination of the CNN and Transformer model is the better option due to CNN's having the ability to extract visual features from images, for instance, determining some ingredients. Meanwhile, Transformers are good for processing and producing text, for example composing comprehensive recipes. Moreover, Transformers facilitate the integration of multiple types of inputs therefore incorporating vision and language and enhancing the context of a single output. This combination guarantees more accurate, flexible, and personalized results than traditional model combinations.

CHAPTER 3: SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing college management systems are predominantly centralized and rely on traditional database management systems and paper-based processes.

- **Manual Recipe Searching:** Users rely on traditional methods like cookbooks, internet searches, or cooking apps to find recipes, which can be time-consuming and less personalized.
- **Ingredient Wastage:** Leftover ingredients are often wasted due to a lack of tools to suggest recipes based on what's already available in the kitchen.
- **Limited Personalization:** Current solutions often fail to cater effectively to dietary restrictions, allergies, or specific preferences, limiting their usability for diverse audiences.
- **Lack of Visual Recognition:** Existing systems typically don't use image recognition to identify ingredients or suggest recipes, requiring users to manually input information.
- **Generic Suggestions:** Recipe recommendations are often broad and not tailored to specific user needs or local cuisine preferences.

3.2 PROPOSED SYSTEM

- **AI-Powered Recipe Generation:** Use generative AI to provide personalized recipe suggestions based on ingredients, dish names, or even uploaded images.
- **Reduce Food Waste:** Analyze available ingredients and leftover items to generate recipes, helping users make the most of their kitchen inventory.
- **Enhanced Personalization:** Offer recipes customized to individual dietary needs, allergies, and cultural or regional preferences, ensuring inclusivity for all users.
- **Visual Recognition Capabilities:** Utilize image recognition to identify ingredients or dishes from uploaded photos, simplifying the process of finding recipes.
- **Culinary Creativity and Exploration:** Encourage users to try new cuisines and experiment with dishes by providing innovative recipe ideas and diverse options.

- **Seamless Technology Integration:** Leverage AI features to create a modern, intuitive tool that bridges the gap between technology and everyday cooking needs, making it highly user-friendly.

3.3 FEASIBILITY STUDY

A feasibility study is carried out before the project is developed to assess the likelihood that the suggested system will be successful. To ascertain whether developing a new or enhanced system is compatible with cost, benefits, operation, and technology, a feasibility study is required.

3.3.1 Technical Feasibility

- **Advanced AI Integration:** Utilizes generative AI models for personalized recipe suggestions and real-time assistance. Proven technologies like image recognition and natural language processing (NLP) are readily available and can be implemented effectively.
- **Cloud Infrastructure:** Leverages cloud platforms for scalability, storage, and processing power to handle image analysis, AI model execution, and user data securely.
- **Data Security:** Implementing encryption techniques and inherent security features ensures the technical feasibility of safeguarding student data and transactions.

3.3.2 Social Feasibility

- **Promoting Sustainability:** Helps reduce food waste by suggesting recipes based on leftover ingredients, aligning with global sustainability goals.
- **Inclusivity:** Supports diverse dietary needs, allergies, and cultural cuisines, making it socially relevant and inclusive for a wide audience.

3.3.3 Operational Feasibility

- **User-Friendly Interface:** Designed for ease of use, with simple inputs like text, images, or voice commands to make the system accessible for users with varying tech skills.
- **Maintenance and Updates:** Regular updates can be easily deployed through automated systems to add new recipes, improve AI performance, and fix bugs.
- **Training AI Models:** Feasible with readily available datasets and scalable machine learning platforms to ensure high accuracy and performance.

3.3.4 Economic Feasibility

- **Low Development Costs:** Utilizes existing AI tools, open-source frameworks, and cloud solutions to reduce initial development expenses.
- **Cost Savings for Users:** Helps users save money by reducing food waste and promoting efficient use of ingredients.
- **Market Demand:** Aligns with the growing trend of AI-driven solutions in daily life, ensuring a strong user base and potential for high returns on investment.

CHAPTER 4: SYSTEM REQUIREMENTS SPECIFICATION

4.1 SOFTWARE REQUIREMENTS

4.1.1 Flask

Flask is a lightweight Python web framework that will serve as the backend of the recipe generation application. As a micro-framework, Flask provides the basic functionality needed to build a web application without the overhead of more complex frameworks. This allows for quick development and easier maintenance of the project, making it ideal for our purpose. Flask supports the development of RESTful APIs, which is essential for our system, as the application will need to handle multiple user requests, such as uploading food images and retrieving recipes based on those images. By using Flask, we can create a server that will respond to these requests efficiently, processing the images and sending back the relevant recipe suggestions.

One of Flask’s key strengths is its extensibility. It supports a wide variety of third-party extensions, such as database connectors, authentication tools, and form validation libraries. This makes it easy to integrate additional features into the application, like user management, favorite recipe saving, and even meal planning. The simplicity of Flask's routing system allows us to quickly define endpoints that the front end can use to send requests. For instance, the user interface could call an endpoint to upload an image, which Flask will then process and respond to with recipe suggestions. Since Flask is written in Python, it integrates seamlessly with Python-based machine learning libraries like TensorFlow and PyTorch, allowing us to connect the backend with the AI models that will process the food images and generate the recipes.

Flask is also highly compatible with various cloud platforms and can easily be deployed to servers or cloud services like AWS, Google Cloud, or Heroku. This ensures that the application can scale as needed, handling an increasing number of users and image requests without significant performance issues. The framework’s minimalistic approach makes it an excellent choice for projects where customization and flexibility are required, providing just enough functionality to support the application without unnecessary complexity.

4.1.2 AJAX

AJAX (Asynchronous JavaScript and XML) will be used on the front end to facilitate seamless communication between the user and the server. With AJAX, the web application will be able to exchange data with the server without requiring a full page reload, making the user experience faster and more interactive. When users upload a food image, the application will send the image data to the Flask backend using an AJAX request, allowing the server to process it and return the recipe suggestions dynamically. This means that the page will not need to refresh each time an image is uploaded or a recipe is requested, creating a smoother experience for users.

The ability to update parts of a webpage asynchronously is one of the key advantages of using AJAX. It ensures that users can interact with the application in real time without disruptions. For example, when a user uploads a food image, AJAX will send the data to the backend, and once the server processes the image and generates a recipe, the results will be displayed instantly on the same page. This eliminates the wait time associated with traditional page reloads and keeps the user engaged with the app. Additionally, AJAX works in the background, allowing the user to continue interacting with other elements of the website while waiting for the response.

AJAX also supports multiple data formats, such as JSON and XML, which are lightweight and easy to handle. This is particularly useful when dealing with large sets of recipe data that need to be exchanged between the client and server. By using JSON to format the recipe information, the server can send back a structured response that the front end can easily parse and display. This will enable the application to show recipe names, ingredients, instructions, and even images, all without the need for page reloads. In this way, AJAX ensures that the recipe generator remains responsive and user-friendly, providing a modern, seamless experience.

4.1.3 Image Processing Library

OpenCV (Open-Source Computer Vision Library) will play a critical role in processing the food images before they are passed to the machine learning models. OpenCV is a highly efficient library that provides a wide range of tools for image manipulation and analysis, making it ideal for our image preprocessing needs. When users upload food images, OpenCV will first resize and standardize them to ensure they are ready for processing by the machine learning models. Resizing images ensures uniformity and prevents large file sizes from affecting processing time. OpenCV can also adjust the contrast and brightness of images to make key features more prominent, ensuring that the AI models can detect ingredients and identify dishes with higher accuracy.

In addition to basic preprocessing, OpenCV also offers advanced image processing capabilities such as edge detection, object recognition, and feature extraction. These capabilities can help improve the quality of the data fed into the machine-learning model. For example, OpenCV's feature extraction techniques can identify certain shapes or textures in the image that may correlate with specific ingredients. These features will be used to enhance the accuracy of the recipe generation system. Additionally, OpenCV can help identify and separate different components of a dish, such as individual ingredients or food items, which will aid the model in understanding the context of the image.

OpenCV can be seamlessly integrated with deep learning models built using TensorFlow or PyTorch, allowing for the image data to be pre-processed efficiently before being passed on for classification. This library will work as a pre-processing layer, ensuring that the food images are ready for analysis by the AI systems. The ability to quickly and efficiently handle food image data is vital to the project's success, and OpenCV's speed and precision will ensure that the recipe generation process remains quick and accurate.

4.1.4 AI/ML Framework

The machine learning framework used in this project will be either TensorFlow or PyTorch, both of which are powerful, widely used tools for building deep learning models. TensorFlow is particularly known for its scalability and robustness, making it suitable for large-scale applications. It provides a comprehensive ecosystem, including tools for building, training, and deploying machine learning models. TensorFlow's capabilities in handling complex data structures and performing operations on large datasets will be essential for training the AI models to identify ingredients from food images. Once trained, the models will be capable of recognizing ingredients in new images and generating relevant recipes based on that data.

PyTorch, on the other hand, is a dynamic framework that is often favored for research and experimentation due to its flexibility and ease of use. Unlike TensorFlow, PyTorch uses dynamic computation graphs, which makes it more intuitive for researchers and developers to modify and experiment with models on the fly. This flexibility allows for quick iterations when refining the models, which is essential for fine-tuning the recipe generation process. PyTorch's integration with other Python libraries and its strong community support make it an excellent choice for developing the machine-learning models required for this project.

Both frameworks provide extensive support for deep learning techniques like convolutional neural networks (CNNs) for image recognition. For this project, these frameworks will be used to train models that can accurately identify ingredients in food images and generate corresponding recipe instructions. TensorFlow and PyTorch both offer tools for model optimization, ensuring that the deployed models can handle the real-time demands of recipe generation while delivering accurate results quickly. Whether TensorFlow or PyTorch is chosen, both frameworks will provide the necessary capabilities to create an AI-driven recipe generation system.

4.1.5 Testing Frameworks

PyTest is a powerful and flexible testing framework for Python that will be used to test the backend functionality built using Flask. PyTest simplifies the process of writing test cases by supporting fixtures, parameterized tests, and plugins. It can be easily integrated into Flask applications to test individual routes, API endpoints, and business logic. For example, PyTest will be used to verify that the Flask backend correctly processes image uploads, interacts with the machine learning models, and returns accurate recipe suggestions. It also helps in testing the database interactions, ensuring that user data, such as saved recipes and preferences, is handled correctly. With PyTest, we can automate the testing process, run test cases with different inputs, and easily handle any potential edge cases.

4.2 HARDWARE REQUIREMENTS

- **CPU:** A multi-core processor (e.g., Intel Xeon or AMD Ryzen) with sufficient processing power to handle concurrent user requests.
- **GPU:** A minimum of 4 GB of VRAM to ensure smooth perfomance.
- **RAM:** A minimum of 16 GB of RAM to ensure smooth performance, but the actual requirement may increase based on usage patterns.
- **Storage:** SSD (Solid-State Drive) storage for fast data access.
- **Network:** Gigabit Ethernet or higher for reliable data transfer between servers.

4.3 LANGUAGE DESCRIPTION

4.3.1 HTML

HTML is the standard markup language for creating web pages and web applications. It defines the structure and layout of web content using elements and tags. HTML is not a programming language but rather a markup language for structuring web documents. HTML documents consist of a collection of elements, each represented by tags that denote the beginning and end of the element. These elements can include headings, paragraphs, images, links, forms, lists, and more. Tags are crucial in defining the relationships between different parts of the content and specifying how they should be rendered on the user's browser. Web developers use HTML as a foundation for creating responsive and engaging websites and web applications. The language's simplicity, along with its ability to integrate seamlessly with other technologies, makes it an essential tool for building a wide range of digital experiences. As the internet continues to evolve, HTML remains a cornerstone for shaping the structure and presentation of online content.

4.3.2 CSS (Cascading Style Sheets)

CSS is used for styling web pages and controlling the presentation of HTML elements. It allows developers to define the colours, fonts, spacing, and layout of web content. CSS is crucial for creating visually appealing and responsive web designs. Selectors are a key component of CSS, specifying which HTML elements are targeted by a particular style rule. CSS provides a wide range of selectors, including element selectors, class selectors, and ID selectors, allowing for fine-grained control over styling.

Media queries in CSS enable responsive web design by allowing developers to apply different styles based on characteristics such as screen size, resolution, or device orientation. This ensures a consistent and optimal user experience across various devices and screen sizes.

4.3.3 JavaScript

JavaScript is a versatile programming language primarily used for adding interactivity and behaviour to web pages. It can be executed in web browsers, making it ideal for creating dynamic web applications. JavaScript is an essential component of modern web development. Being an interpreted language, JavaScript executes code on-the-fly, without the need for a compilation step. This characteristic allows for quick development cycles and immediate feedback during the coding process. Additionally, JavaScript is dynamically typed, meaning variables can hold values of any type without explicit type declarations. While this flexibility can lead to concise code, developers need to be mindful of potential type-related issues.

JavaScript's event-driven and asynchronous nature is another notable feature. It responds to various events triggered by user interactions, such as clicks or keyboard input. The language's asynchronous capabilities, facilitated by features like callbacks and promises, enable the execution of non-blocking code. This is particularly valuable for handling tasks like fetching data from servers without freezing the user interface.

Beyond the browser, JavaScript has expanded its reach through environments like Node.js, allowing developers to use the language for server-side programming. This versatility has made JavaScript a prominent language in full-stack development, where it can be employed on both the client and server sides, providing a unified language stack for building modern web applications. Overall, JavaScript's adaptability, coupled with its vibrant ecosystem of libraries and frameworks, has solidified its position as a foundational technology in the world of web development.

4.3.4 Python

Python is a high-level, interpreted, and general-purpose programming language known for its simplicity, readability, and versatility. It is one of the most widely used programming languages across various domains, including web development, data analysis, machine learning, artificial intelligence, automation, and scientific computing. Python’s design philosophy emphasizes code readability, making it easier for developers to write clean and maintainable code.

Python supports multiple programming paradigms, including object-oriented, imperative, functional, and procedural programming. It provides a rich standard library, which offers modules and packages that handle various tasks such as file I/O, system operations, and even web development. Python’s syntax is clean and straightforward, with a strong emphasis on indentation rather than braces or semicolons, which makes code easier to understand and maintain.

One of Python's key strengths is its large ecosystem of third-party libraries and frameworks. For example, in the context of this project, Python can be used for building the backend server using Flask or Django, handling image processing using libraries like OpenCV, and integrating machine learning frameworks such as TensorFlow or PyTorch for the AI model. Python also supports integration with other languages like C, C++, and Java, allowing developers to leverage high-performance code when needed.

Python is dynamically typed, meaning you don’t need to declare variable types explicitly. This makes development faster, but it also requires more attention during testing and debugging. Python is interpreted, meaning that the code is executed line by line, which helps during the development process but might impact performance in computationally heavy applications. However, Python’s vast libraries, efficient memory management, and ease of use make it suitable for rapid prototyping and large-scale application development alike.

Due to its ease of learning, large community support, and widespread adoption, Python is often chosen for both small and large-scale projects. Its flexibility and powerful libraries make it an excellent choice for a wide range of applications, from simple scripts to complex AI systems.

4.4 DATASETS & ALGORITHMS

4.4.1 Recipe1M Dataset

The Recipe1M dataset is a large-scale dataset designed for research in recipe generation, understanding, and recommendation. It is one of the most widely used datasets in the field of culinary AI and computer vision. The dataset is built to bridge the gap between computer vision and natural language processing, offering valuable data for training models to understand and generate recipes based on both images and text.

https://pic2recipe.csail.mit.edu/

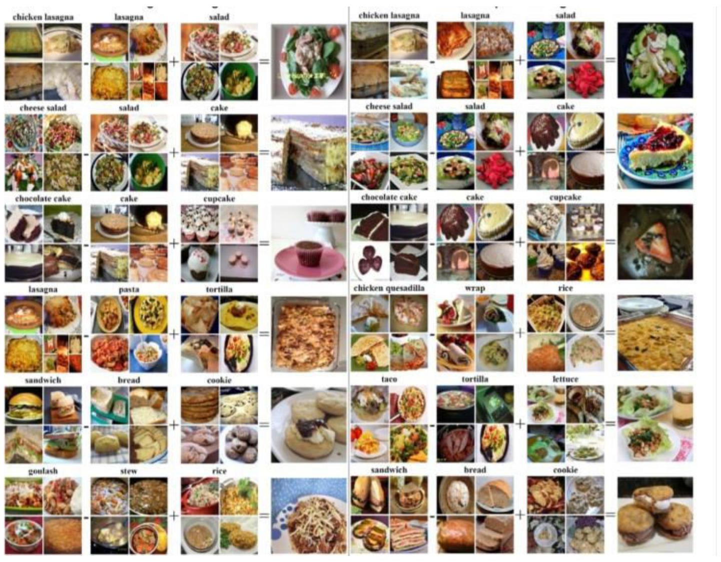
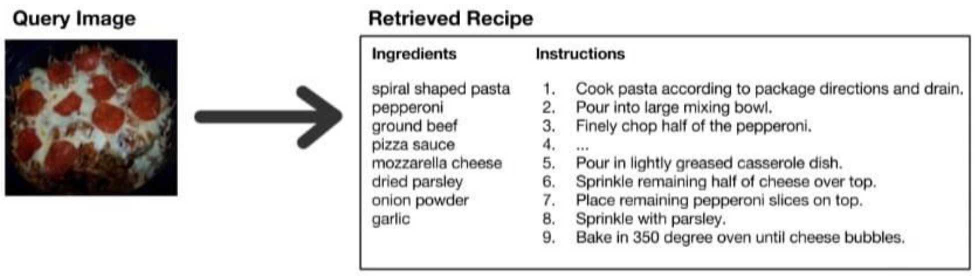


Figure 4.1 (a) Recipe 1M dataset



Ingredients

spiral shaped pasta
pepperoni
ground beef
pizza sauce
mozzarella cheese
dried parsley
onion powder
garlic

Instructions

1. Cook pasta according to package directions and drain.
2. Pour into large mixing bowl.
3. Finely chop half of the pepperoni.
4. ...
5. Pour in lightly greased casserole dish.
6. Sprinkle remaining half of cheese over top.
7. Place remaining pepperoni slices on top.
8. Sprinkle with parsley.
9. Bake in 350 degree oven until cheese bubbles.

Figure 4.1 (b) Recipe 1M dataset – Retrieved Recipe

Key Features of Recipe1M:

Size and Scope:

- The Recipe1M dataset contains over 1 million recipes.
- It includes 1 million cooking images, each associated with a textual recipe description.
- The recipes cover a wide variety of dishes from different cuisines around the world.

Data Components:

- Images: Each recipe has an image showing the dish or a key ingredient, which can be used for vision-based tasks like ingredient recognition, dish classification, or recipe generation.
- Text Descriptions: The dataset provides rich textual descriptions, including ingredients, preparation steps, and sometimes cooking methods and times. This text data is essential for natural language processing tasks, such as recipe recommendation or generation.
- Ingredients: The recipes include a list of ingredients along with their quantities, helping models understand how ingredients relate to specific cooking steps.

Categories:

- The recipes in Recipe1M span a wide range of food types, from appetizers to desserts, and include various cuisines such as Italian, Chinese, Indian, and more.
- It also contains both simple and complex dishes, offering diversity for training models to handle a variety of recipe types.

4.4.2 Food & Ingredients Dataset

The Food and Ingredients Dataset typically refers to collections of data used for tasks like food recognition, ingredient identification, recipe generation, and dietary analysis. These datasets are essential for building AI models that can understand food items, recognize ingredients from images or text, and generate relevant recommendations. Below is an explanation of the general features and use cases of such datasets, with a focus on typical examples like the Food-101, Food-11, and ingredient-specific datasets.

https://www.kaggle.com/datasets/pes12017000148/food-ingredients-and-recipe-dataset-with-images

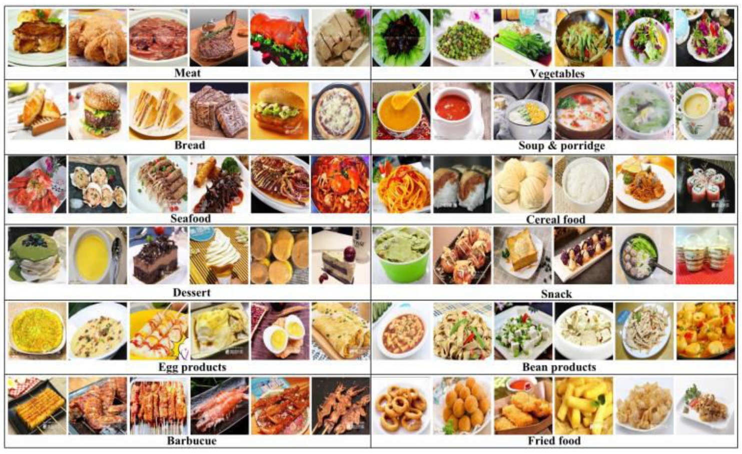


Figure 4.4 Food & Ingredients dataset

Key Features of Food and Ingredients Datasets:

Image Data:

- Ingredient Images: Datasets can also include images of raw ingredients, like tomatoes, onions, or spices. These images help models understand the relationship between ingredients and dishes.

4.4.3 Convolutional Neural Networks

In the Recipe Generation from Food Images project, Convolutional Neural Networks (CNNs) are employed to analyse and understand food images in order to generate relevant recipes. CNNs are particularly well-suited for this task due to their ability to automatically detect and learn patterns, textures, and objects in images, which is crucial for identifying ingredients and cooking styles from food pictures.

The process begins when the food image is input into the CNN model. The CNN first applies a series of convolutional layers that filter the image to extract basic features such as edges, shapes, and textures. These initial features are then progressively combined in deeper layers, allowing the network to detect more complex patterns, such as specific ingredients, food items, or even the arrangement of those ingredients. The power of CNN lies in its ability to capture these hierarchical features, which is essential for understanding the visual structure of a dish.

After the relevant features have been extracted from the image, the CNN uses them to classify or categorize the food item. Once identified, the CNN's output can be mapped to a recipe suggestion by linking the detected ingredients and their respective cooking methods. This connection is crucial for the recipe generation aspect of the project, where the model, based on the recognized ingredients and their combinations, recommends recipes that match the visual data. By using CNNs, the system can automate the process of recognizing and understanding food images, making recipe generation efficient and dynamic without requiring manual input for each dish.

This methodology allows for highly accurate and scalable recipe generation, as CNNs can process large numbers of images and their associated features in parallel, significantly reducing the need for manual labelling or recipe input. Thus, CNNs play a pivotal role in enabling the system to interpret and generate recipes from visual data, enhancing the overall functionality and user experience of the Recipe Generation and Cook Assistant tool.

CHAPTER 5: METHODOLOGY

Rigorous data preprocessing ensures the effectiveness and reliability of our proposed methodology for a recipe generator using generative AI. The dataset, comprising a diverse collection of recipe text data and images of ingredients or dishes, serves as the foundation of our analysis.

5.1 DATA PRE-PROCESSING

Image Pre-Processing

- **Resizing:**

  1.**Load the image**

  Take the image and resize it to a fixed size (48x48 pixels) to ensure consistency. Convert it to black-and-white (grayscale) to reduce complexity.

  2. **Convert to numbers**

  Transform the image into a format that a computer can understand, turning it into a grid of numbers representing pixel intensities.

  3. **Normalize the values**

  Adjust the pixel values to a smaller range (between 0 and 1) so the model can process them more easily and effectively.

  4. **Add a batch dimension**

  Even if there's only one image, prepare it as part of a "batch" since the CNN expects to receive multiple images at once.

  5. **Shape it for the Model**

  Ensure the image has the right structure for the model, specifically indicating it's a single, grayscale image of 48x48 pixels.

- **Normalisation:** Scale pixel values to the range [0, 1] by dividing each pixel intensity by 255. This enhances numerical stability during model training by standardizing the data.

  ```
  image_array = np.array(resized_image)
  normalized_image = image_array / 255.0
  ```

- **Data Augmentation:** Data augmentation techniques are employed to enhance the diversity of the image and improve the model's ability to recognize.

- Horizontal and vertical flips are applied to simulate different viewing angles of the image, effectively increasing the variety of perspectives.
- Random rotations, typically within a range of ±15 degrees, are introduced to account for images taken from slightly tilted positions.
- Translation, or shifting images along the x or y axis, helps simulate variations in ingredient placement within a frame. To mimic real-world lighting and visual conditions, colour adjustments are implemented, which include modifying brightness, contrast, saturation, or hue.
- Additionally, zooming is simulated by randomly cropping and resizing portions of the image, which helps the model focus on different levels of detail within an image.

5.2 RECOGNITION USING TRAINED VISION MODEL

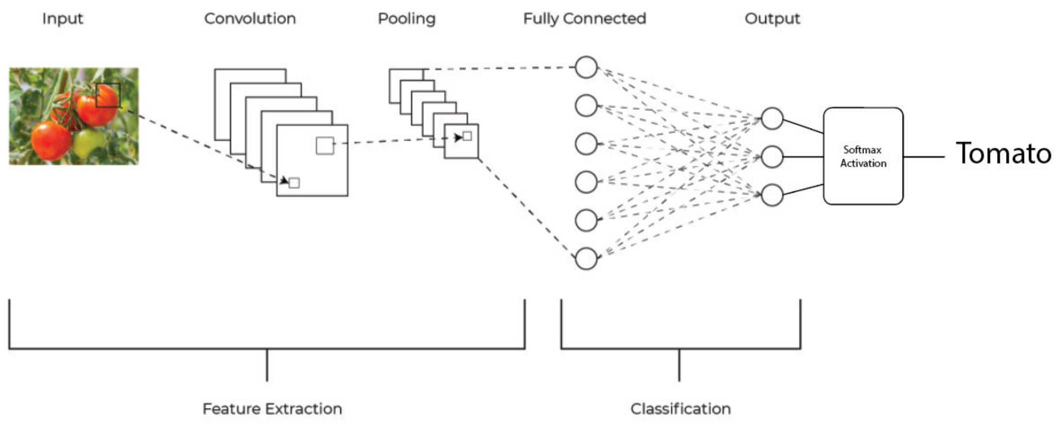


Figure 5.1 CNN Architecture

The process begins with the input layer, where the pre-processed image of a dish or ingredient is fed into the CNN. This ensures compatibility and stability during processing.

The core of the CNN lies in its convolutional layers, which perform feature extraction by scanning the image using a set of learnable filters (kernels). These filters detect patterns such as edges, textures, and shapes:

- Early layers capture basic features like edges and corners.

- Deeper layers capture complex patterns like specific textures or unique features of ingredients (e.g., the seeds in a tomato or the shape of pasta).

After each convolutional layer, an activation function (typically ReLU, or Rectified Linear Unit) is applied. ReLU introduces non-linearity, enabling the network to learn complex relationships and focus on meaningful features while ignoring irrelevant information.

Pooling layers are used to down sample feature maps, reducing their dimensions and computational complexity.

- Max pooling selects the most prominent feature in a small region (e.g., a 2x2 grid).
- Pooling helps retain important features while discarding less significant details, ensuring the model focuses on essential patterns.

After several convolutional and pooling layers, the extracted features are flattened into a one-dimensional vector and passed to fully connected (dense) layers. These layers learn to map the extracted features to specific labels, such as “tomato,” “carrot,” or “pasta.”

The final layer is the output layer, which uses an activation function such as softmax for multi-class classification. Softmax outputs probabilities for each class, indicating the likelihood that the image belongs to a particular category. For instance, the model might output:

- Tomato: 85%
- Carrot: 10%
- Potato: 5%

The class with the highest probability is chosen as the predicted label.

5.3 GENERATING OUTPUT USING TRANSFORMER MODEL

Input Gathering

Image Input: The image is processed through the trained CNN to identify the dish name or key ingredients. For example, the CNN might recognize “tomato,” “pasta,”and “basil” as key ingredients from an uploaded image.

Text Input (Optional): If the user provides text data, such as dietary preferences ("vegetarian"), cuisine preferences ("Italian"), or additional ingredients ("cheese"), this is tokenized and prepared for processing.

System Prompt Preparation

The system prompt is a structured input designed to guide the transformer model. It combines outputs from the CNN and user text input into a cohesive instruction. For example:

Example Prompt: "*Generate a recipe using tomato, pasta, and basil for an Italian vegetarian dish. Include cheese as an additional ingredient.*"

The prompt is designed to provide context, constraints, and a clear task to the transformer model, ensuring it generates a recipe aligned with user needs.

Transformer Model Processing

- **Contextual Understanding:** The model uses its pre-trained knowledge of recipes, cooking techniques, and ingredient combinations to understand the task.
- **Attention Mechanism:** The transformer's attention layers focus on important parts of the prompt, such as specific ingredients, dietary restrictions, or cuisine preferences.
- **Recipe Generation:** Using its decoder, the model outputs a step-by-step recipe, including ingredient quantities, preparation methods, and cooking steps. For example:
    - **Ingredients:**
        - 2 cups pasta,
        - 1 cup tomato puree,
        - fresh basil leaves,
        - 1 cup shredded cheese.
    - **Steps:**
        - Boil the pasta until it's just tender.
        - In a pan, sauté tomato puree with olive oil and garlic.
        - Mix the pasta with the tomato sauce and top with cheese and basil.

Integration of Additional Text Data

If the user provides text input (e.g., “Add a spicy flavor”), the transformer integrates this into the recipe generation process. It might include an additional step such as: “*Add a teaspoon of red chili flakes to the tomato sauce*.”

Output Formatting

The generated recipe is formatted into a user-friendly structure suitable for the interface. This includes sections such as:

Dish Name: “Italian Tomato Basil Pasta.”

Ingredients: A detailed list.

Preparation Steps: A step-by-step guide.

CHAPTER 6: SYSTEM DESIGN

6.1 ARCHITECTURE DIAGRAM

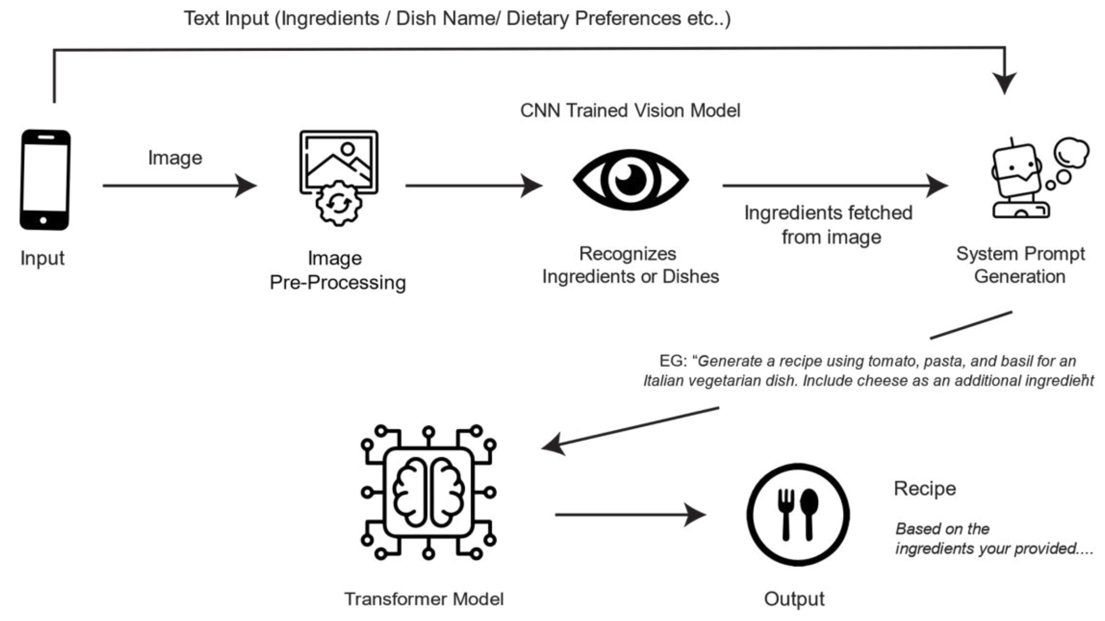

Text Input (Ingredients / Dish Name/ Dietary Preferences etc..)
Image
CNN Trained Vision Model
Input
Image Pre-Processing
Recognizes Ingredients or Dishes
Ingredients fetched from image
System Prompt Generation
EG: "Generate a recipe using tomato, pasta, and basil for an Italian vegetarian dish. Include cheese as an additional ingredient
Recipe
Based on the ingredients your provided....
Transformer Model
Output

Figure 6.1 Architecture Diagram

Figure 6.1 illustrates a system that creates recipes using user-supplied data, including food photos, component lists, dish names, and dietary restrictions. An input, which could be a word or an image, is used to begin the procedure. A Convolutional Neural Network (CNN) trained vision model analyzes a picture after it has undergone pre-processing if one is supplied. This model extracts pertinent ingredient information by identifying the plates or components in the image. Text input from the user, such as dietary restrictions or an ingredient list, is also handled immediately.

After the ingredients are determined, the system creates a well-formulated prompt that outlines the requirements of the recipe, i.e., "Create a recipe based on tomato, pasta, and basil for an Italian vegetarian meal. Add cheese as another ingredient." The prompt is then sent to a Transformer-based model, which utilizes it to produce a recipe. The final result is a specified recipe based on the given ingredients and conditions. The system successfully integrates computer vision and natural language processing to generate personalized recipes from text or images.

6.2 USE CASE DIAGRAM

6.2.1 User

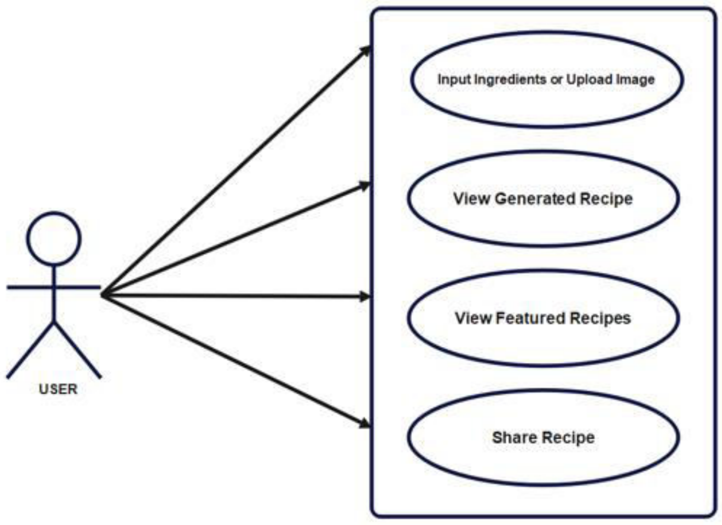


Figure 6.2 Use Case Diagram

Figure 6.2 is a Use Case Diagram of a user's interaction with a recipe program. The User (denoted by a stick figure) may have four key actions in the system, placed within a rectangle boundary. The actions are to enter ingredients or upload an image to produce a recipe, view the produced recipe, view featured recipes, and share a recipe. Every action is depicted as an oval (use case), and arrows show the user's capability to interact with every feature. The figure depicts the system's main functionality in a straightforward and organized way.

6.3 USER INTERFACE DESIGN

6.3.1 Home Page

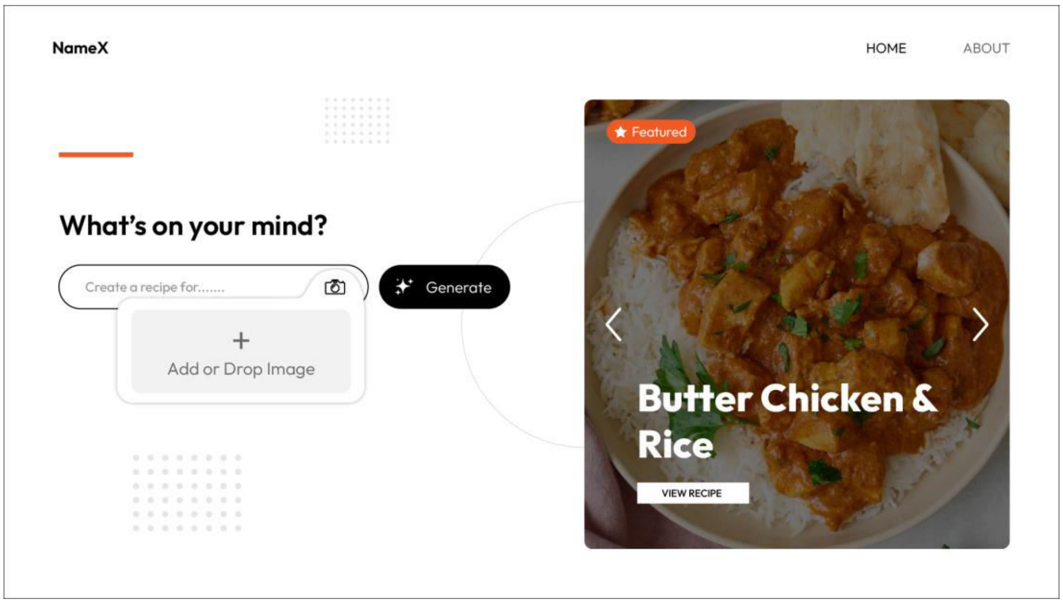
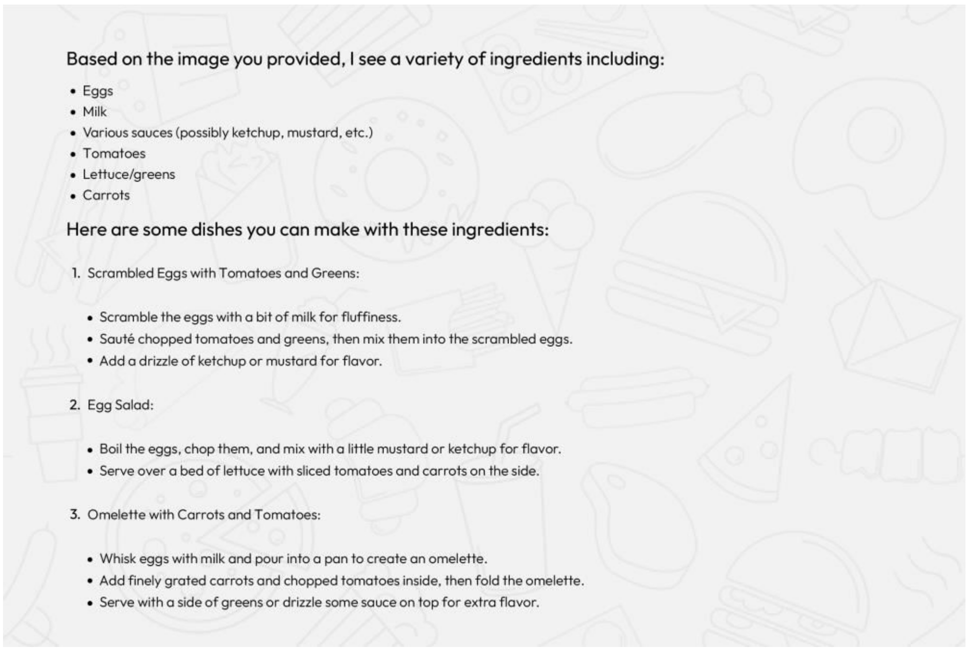


Figure 6.3 Home Page

6.3.2 Recipe Page



Based on the image you provided, I see a variety of ingredients including:

- Eggs
- Milk
- Various sauces (possibly ketchup, mustard, etc.)
- Tomatoes
- Lettuce/greens
- Carrots

Here are some dishes you can make with these ingredients:

1. Scrambled Eggs with Tomatoes and Greens:
    - Scramble the eggs with a bit of milk for fluffiness.
    - Sauté chopped tomatoes and greens, then mix them into the scrambled eggs.
    - Add a drizzle of ketchup or mustard for flavor.
2. Egg Salad:
    - Boil the eggs, chop them, and mix with a little mustard or ketchup for flavor.
    - Serve over a bed of lettuce with sliced tomatoes and carrots on the side.
3. Omelette with Carrots and Tomatoes:
    - Whisk eggs with milk and pour into a pan to create an omelette.
    - Add finely grated carrots and chopped tomatoes inside, then fold the omelette.
    - Serve with a side of greens or drizzle some sauce on top for extra flavor.

Figure 6.4 Recipe Page

Chapter 7: SYSTEM IMPLEMENTATION

7.1 SOURCE CODE

7.1.1 Main.py

```
import base64
import mimetypes
import requests
import threading
import time
import random
import json
import tensorflow as tf
import numpy as np
import cv2
import torch
import torchvision.transforms as transforms
from PIL import Image
from flask import Flask, render_template, jsonify, request as req
from scipy.ndimage import rotate
import nltk

nltk.download('punkt')

app = Flask(__name__)

MODEL_PATH = "models/model.h5"

def load_graph():
    if not tf.io.gfile.exists(MODEL_PATH):
        raise FileNotFoundError("Model file is missing!")
    graph = tf.Graph()
    with tf.io.gfile.GFile(MODEL_PATH, "rb") as f:
        graph_def = tf.compat.v1.GraphDef()
        graph_def.ParseFromString(f.read())
        with graph.as_default():
            tf.import_graph_def(graph_def, name="")
    return graph

graph = load_graph()

def preprocess_image(image_data):
    image = cv2.imdecode(np.frombuffer(image_data.read(), np.uint8), cv2.IMREAD_COLOR)
    pil_image = Image.fromarray(image)
    image = np.array(pil_image)
    image = rotate(image, angle=random.choice([0, 90, 180, 270]), reshape=False)
    image = cv2.resize(image, (224, 224))
    image = np.expand_dims(image, axis=0)
    image = image / 255.0
    return image
```

```
def predict_image(image_data):
    with graph.as_default():
        with tf.compat.v1.Session(graph=graph) as sess:
            input_tensor = graph.get_tensor_by_name("input:0")
            output_tensor = graph.get_tensor_by_name("final_result:0")
            processed_image = preprocess_image(image_data)
            predictions = sess.run(output_tensor, {input_tensor: processed_image})
            prediction_list = predictions.tolist()
            flattened_predictions = [item for sublist in prediction_list for item in sublist]
            top_index = np.argmax(flattened_predictions)
            confidence_score = flattened_predictions[top_index]
            return f"Detected item with confidence {confidence_score:.5f}"

def get_type(file_name):
    mime_type, _ = mimetypes.guess_type(file_name)
    return mime_type if mime_type else "unknown/unknown"

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/recipe/<int:recipe_id>')
def recipe(recipe_id):
    return render_template('recipe.html', recipe_id=recipe_id)

def async_task(func, *args):
    thread = threading.Thread(target=func, args=args)
    thread.start()
    return thread

@app.route('/send', methods=['POST'])
def send():
    try:
        form = req.form
        image = req.files.get('image')
        text_input = form.get('text_input', "").strip()
        recognized_text = ""
        if image:
            async_task(lambda img: print("Recognized:", predict_image(img)), image)
            recognized_text = predict_image(image)
        if text_input:
            text_input += f" Also, {recognized_text}." if recognized_text else ""
        else:
            text_input = recognized_text if recognized_text else "Suggest a recipe."
        headers = {
            "Authorization": "Bearer api03-gNgWi3dkhU8US-
j3ZC5MUqeNh8zdrRJZIx80dV7oqCxAcZUieM",
            "Content-Type": "application/json"
        }

        response = requests.post(
            "https://chat.cloxo.co/api/chat/completions",
            headers=headers,
            json={
                "model": "cloxoai/cloxogpt",
            "messages": [{"role": "user", "content": text_input}]
          }
        )
        response_data = response.json()
        message = response_data.get("choices", [{}])[0].get("message", {}).get("content", "No
response.")
        return jsonify({"recipe": message})
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

7.1.2 Index

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ChefX</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='/css/style.css') }}">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.5.1/css/all.min.css">
    <script src="{{ url_for('static', filename='/js/modules/jquery-3.7.1.min.js')}}"></script>
    <link rel="stylesheet" href="https://unpkg.com/swiper/swiper-bundle.min.css">
</head>
<body>
    <!-- <header>
      <a href="/">
        <p class="logo">ChefX</p>
      </a>
      <div class="navRight">
        <a href="/" class="active">Home</a>
        <a href="/">How It Works</a>
      </div>
    </header> -->
    <main>
      <section class="home">
        <div class="formSection">
          <div class="line"></div>
          <h3>What's on your mind?</h3>
          <form id="recipe-form" enctype="multipart/form-data">
              <div class="inputField">
                <input type="text" name="text_input" placeholder="Create a recipe for.....">
                <i class="fas fa-camera camIcon" onclick="toggleUploadImage()"></i>
                <div class="uploadImage" id="uploadImage" style="display: none;">
                  <div onclick="triggerFileInput()">
                      <img id="displayImage" src="{{url_for('static',filename='images/jojo.jpg')}}"
alt="Image" style="display: none; width: 200px; height: auto;">
                      <input type="file" id="imageInput" name="image" accept="image/*" style="display:
none;" onchange="showSelectedImage(event)">
                      <i class="fas fa-plus"></i>
                      Upload Image
                  </div>
      </div>
     </div>
     <button type="submit" name="button"> <i class="fas fa-star"></i> Generate</button>
  </form>
</div>
<div class="featuredSection">
  <div class="swiper-container">
    <div class="swiper-wrapper" id="swiperWrapper">
    </div>
    <div class="swiper-button-next" style="color:white"></div>
    <div class="swiper-button-prev" style="color:white"></div>
    <div class="swiper-pagination"></div>
    <style media="screen">
    .swiper-pagination-bullet {
      background-color: white;
      }
      .swiper-pagination-bullet-active {
      background-color: white;
      }
    </style>
  </div>
</div>

<script src="https://unpkg.com/swiper/swiper-bundle.min.js"></script>
<script>
  const slides = [
    {
      img: "https://www.themealdb.com/images/media/meals/1544384070.jpg",
      title: "Peanut Butter Cookies",
      link: "recipe/52958"
    },
    {
      img: "https://www.themealdb.com/images/media/meals/wyxwsp1486979827.jpg",
      title: "Chicken Handi",
      link: "recipe/52795"
    },
    {
      img: "https://www.themealdb.com/images/media/meals/t8mn9g1560460231.jpg",
      title: "Tunisian Lamb Soup",
      link: "recipe/52972"
    },
    {
      img: "https://www.themealdb.com/images/media/meals/wvqpwt1468339226.jpg",
      title: "Mediterranean Pasta Salad",
      link: "recipe/52777"
    }
  ];

  const swiperWrapper = document.getElementById("swiperWrapper");

  slides.forEach((slide) => {
    const slideElement = document.createElement("div");
    slideElement.classList.add("swiper-slide");
    slideElement.innerHTML = `
      <img src="${slide.img}" alt="${slide.title}">
```

```
          <div class="sw-overlay">
            <p class="sw-overlay-featured"><i class="fas fa-star"></i> Featured</p>
            <div class="sw-overlay-desc">
              <p>${slide.title}</p>
              <a href="${slide.link}" target="_blank">
                <button type="button">Learn More</button>
              </a>
            </div>
          </div>
        `;
        swiperWrapper.appendChild(slideElement);
      });

      const swiper = new Swiper('.swiper-container', {
        loop: true,
        speed: 500,
        effect: 'fade',
        fadeEffect: {
          crossFade: true,
        },
        autoplay: {
          delay: 2000,
          disableOnInteraction: false,
        },
        navigation: {
          nextEl: '.swiper-button-next',
          prevEl: '.swiper-button-prev',
        },
        pagination: {
          el: '.swiper-pagination',
          clickable: true,
        },
        slidesPerView: 1,
        spaceBetween: 20,
      });
    </script>

  </section>
  <section id="generatedOutput">
      <div id="result" class="results" style="display: none;">
        <div id="loader" class="loader-overlay" style="display: none;">
          <div class="loader"></div>
        </div>
      </div>
  </section>

</main>



<script>
    window.onload = function() {
      document.getElementById('recipe-form').reset();
    };
    function triggerFileInput() {
```

```
    document.getElementById('imageInput').click();
    }
    function showSelectedImage(event) {
    var file = event.target.files[0];
    if (file) {
        var imgUrl = URL.createObjectURL(file);
        var imgElement = document.getElementById('displayImage');
        imgElement.src = imgUrl;
        imgElement.style.display = 'block';
    }
}

function toggleUploadImage() {
  var uploadDiv = document.getElementById("uploadImage");
  if (uploadDiv.style.display === "none") {
    uploadDiv.style.display = "flex";
  } else {
    uploadDiv.style.display = "none";
  }
}

$('#recipe-form').submit(function(event) {
    event.preventDefault();

    $('#result').stop().slideDown('slow', function() {
        $('#loader').show();
        $('html, body').animate({
        scrollTop: $('#result').offset().top
    }, 'slow');
    });

    var formData = new FormData(this);

    $.ajax({
        type: 'POST',
        url: '/send',
        data: formData,
        contentType: false,
        processData: false,
        success: function(response) {
            console.log(response);

            $('#loader').hide();

            $('#result').html(response.recipe);

            $('html, body').animate({
                scrollTop: $('#result').offset().top
            }, 'slow', function() {
                let windowHeight = $(window).height();
                $('#result').css('min-height', windowHeight + 'px');
            });
        },
        error: function(xhr, status, error) {
            console.error(error);
```

```
                $('#loader').hide();
            }
        });
    });
  </script>
</body>
</html>
```

7.1.3 Recipe

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ChefX</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='/css/style.css') }}">
  <link rel="stylesheet" href="{{ url_for('static', filename='/css/recipe.css') }}">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/all.min.css">
  <script src="{{ url_for('static', filename='/js/modules/jquery-3.7.1.min.js')}}"></script>
  <link rel="stylesheet" href="https://unpkg.com/swiper/swiper-bundle.min.css">
</head>
<body>
  <div id="loader-overlay" class="loader-overlay-rp" style="display: flex;">
    <div class="loader"></div>
  </div>
  <header>
    <a href="/">
      <p class="logo">ChefX</p>
    </a>
    <div class="navRight">
      <a href="/" class="active">Home</a>
      <a href="/">How It Works</a>
    </div>
  </header>
  <main>
    <section class="recipeDisplay">
      <div class="recipeDisplayHero">
        <div class="imageSection">
          <img src="" alt="" id="recipeImage">
        </div>
        <div class="detailSection">
          <p class="sw-overlay-featured" style="width: fit-content"><i class="fas fa-star"></i> Featured</p>
          <p class="name" id="recipeName"></p>
          <p class="category"><i class="fas fa-utensils"></i> <span id="recipeCategory"></span></p>
          <p class="region"><i class="fas fa-globe"></i> <span id="recipeRegion"></span></p>
          <div class="tags" id="recipeTags"></div>
        </div>
      </div>
      <div class="recipeDisplayDetails">
        <div class="ingredients">
      <p class="header">Ingredients.</p>
      <div class="items" id="ingredientsList"></div>
    </div>
    <div class="instructions">
      <p class="header">Directions.</p>
      <div class="steps" id="recipeSteps"></div>
    </div>
  </div>
 </section>
</main>

<script type="text/javascript">
 function getRecipeIdFromUrl() {
   const path = window.location.pathname;
   const segments = path.split("/");
   return segments[segments.length - 1];
 }

 async function fetchRecipe() {
   const recipeId = getRecipeIdFromUrl();
   if (!recipeId) return console.error("Recipe ID not found in URL");

   const apiUrl = `https://www.themealdb.com/api/json/v1/1/lookup.php?i=${recipeId}`;

   try {
      const response = await fetch(apiUrl);
      const data = await response.json();

      if (data.meals) {

         displayRecipe(data.meals[0]);
         document.getElementById("loader-overlay").style.display = "none";
      } else {
         console.error("Recipe not found");
      }
   } catch (error) {
      console.error("Error fetching recipe:", error);
   }
 }

 function displayRecipe(meal) {
   document.getElementById("recipeImage").src = meal.strMealThumb;
   document.getElementById("recipeName").innerText = meal.strMeal;
   document.getElementById("recipeCategory").innerText = meal.strCategory;
   document.getElementById("recipeRegion").innerText = meal.strArea;

   const tagsContainer = document.getElementById("recipeTags");
   meal.strTags && meal.strTags.split(',').forEach(tag => {
     const tagElement = document.createElement("p");
     tagElement.innerHTML = `<i class="fas fa-tags"></i> ${tag}`;
     tagsContainer.appendChild(tagElement);
   });

   const ingredientsList = document.getElementById("ingredientsList");
   for (let i = 1; i <= 20; i++) {
          const ingredient = meal[`strIngredient${i}`];
          const measure = meal[`strMeasure${i}`];
          if (ingredient && ingredient.trim() !== "") {
            const ingredientElement = document.createElement("div");
            ingredientElement.classList.add("item");
            ingredientElement.innerHTML = `
              <p class="itemQ">${measure || ''}</p>
              <p class="itemN">${ingredient}</p>
            `;
            ingredientsList.appendChild(ingredientElement);
          }
        }

        const recipeSteps = document.getElementById("recipeSteps");
        meal.strInstructions.split('\n').forEach((step, index) => {
          const stepElement = document.createElement("div");
          stepElement.classList.add("step");
          stepElement.innerHTML = `
            <p class="stepN">${index + 1}.</p>
            <p class="stepD">${step}</p>
          `;
          recipeSteps.appendChild(stepElement);
        });
      }
      window.onload = fetchRecipe;
    </script>
</body>
</html>
```

7.1.4 Config.py

```
import os
DATASET_PATH = "C:\\Users\\aredc\\Desktop\\project-dataset-recipe"
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS = 20
LEARNING_RATE = 0.001
MODEL_PATH = os.path.join(os.getcwd(), "models", "model.h5")
```

7.1.5 Dataset_loader.py

```
import tensorflow as tf
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from config import DATASET_PATH, IMAGE_SIZE, BATCH_SIZE

def load_data():
    datagen = ImageDataGenerator(
        rescale=1.0 / 255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        validation_split=0.2
    )
    train_generator = datagen.flow_from_directory(
        DATASET_PATH,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        subset='training'
    )

    val_generator = datagen.flow_from_directory(
        DATASET_PATH,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        subset='validation'
    )

    return train_generator, val_generator
```

7.1.6 model.py

```
import tensorflow as tf
from tensorflow.keras import layers, models

def build_model(num_classes):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

7.1.7 train.py

```
import tensorflow as tf
from dataset_loader import load_data
from model import build_model
from config import MODEL_PATH, EPOCHS

train_data, val_data = load_data()

num_classes = len(train_data.class_indices)

model = build_model(num_classes)
```

```
history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=EPOCHS
)

model.save(MODEL_PATH)
print(f"Model saved at: {MODEL_PATH}")
```

7.2 SCREENSHOT

7.2.1 Home page

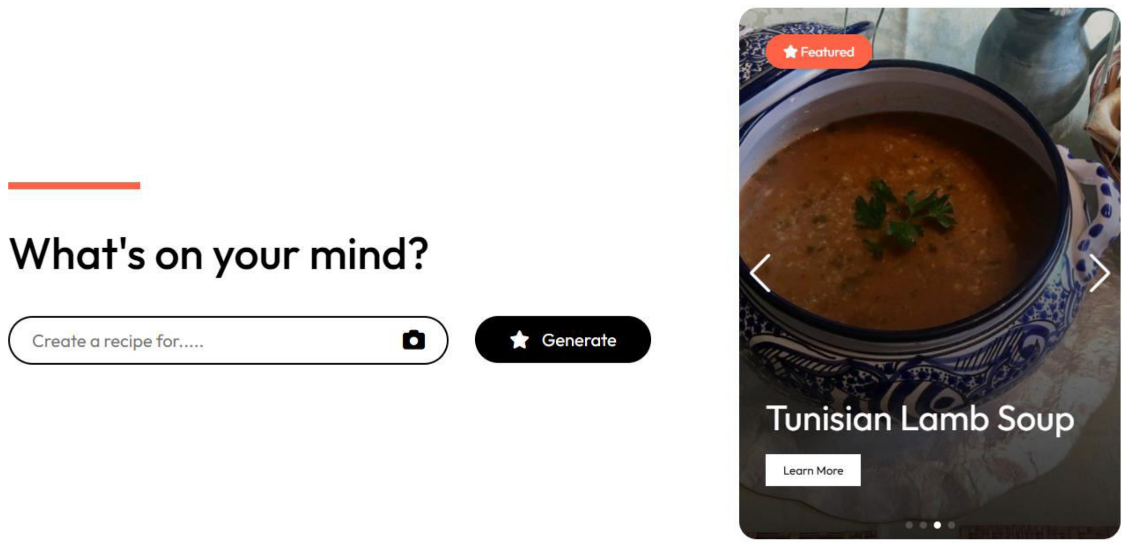


Figure 7.1 Home page

7.2.2 Featured recipes

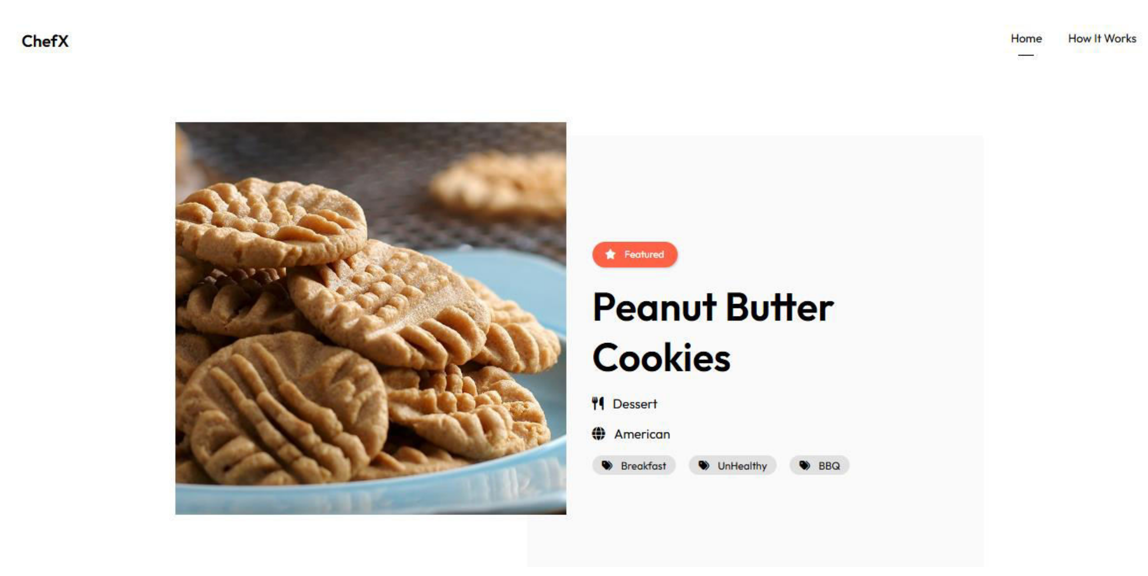
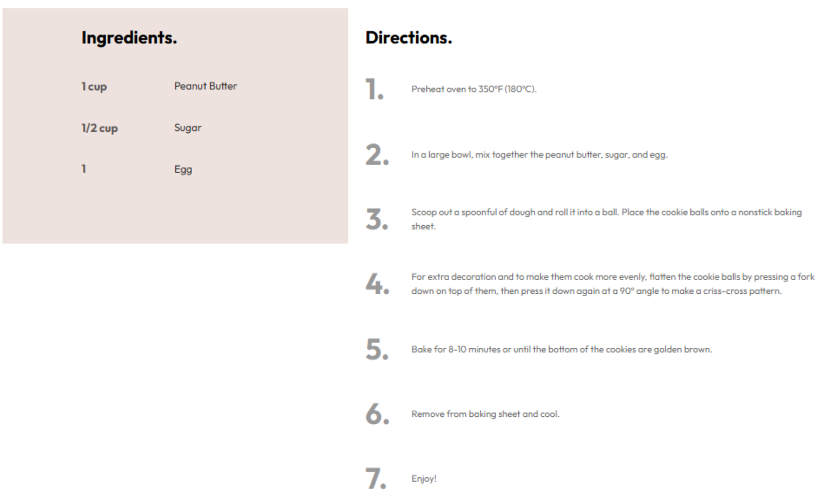


Figure 7.2.(a) Featured recipe



Ingredients.

1 cup	Peanut Butter
1/2 cup	Sugar
1	Egg

Directions.

1. Preheat oven to 350°F (180°C).
2. In a large bowl, mix together the peanut butter, sugar, and egg.
3. Scoop out a spoonful of dough and roll it into a ball. Place the cookie balls onto a nonstick baking sheet.
4. For extra decoration and to make them cook more evenly, flatten the cookie balls by pressing a fork down on top of them, then press it down again at a 90° angle to make a criss-cross pattern.
5. Bake for 8-10 minutes or until the bottom of the cookies are golden brown.
6. Remove from baking sheet and cool.
7. Enjoy!

Figure 7.2.(b) Featured recipe

7.2.3 Recipe generating

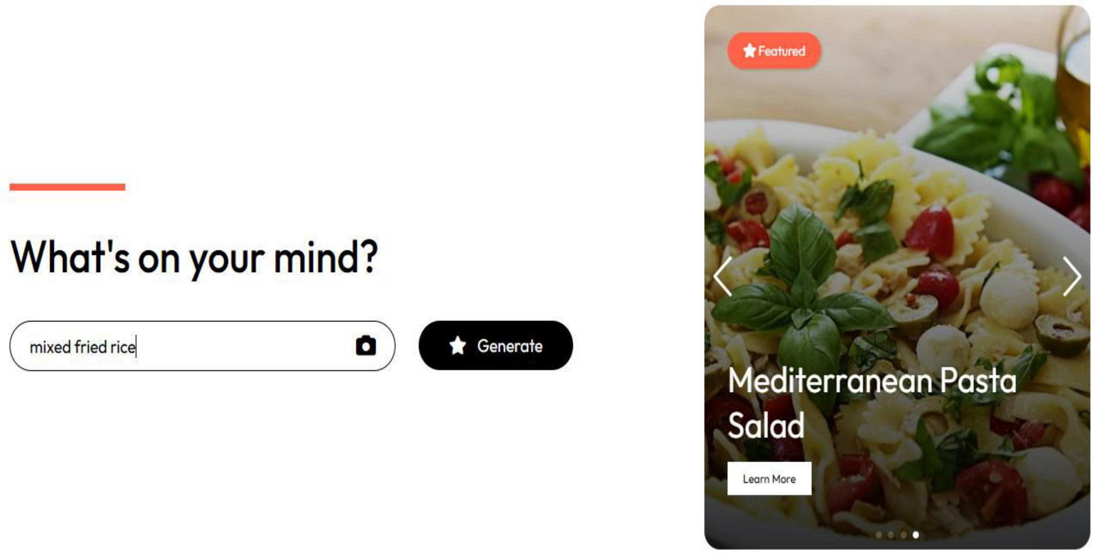
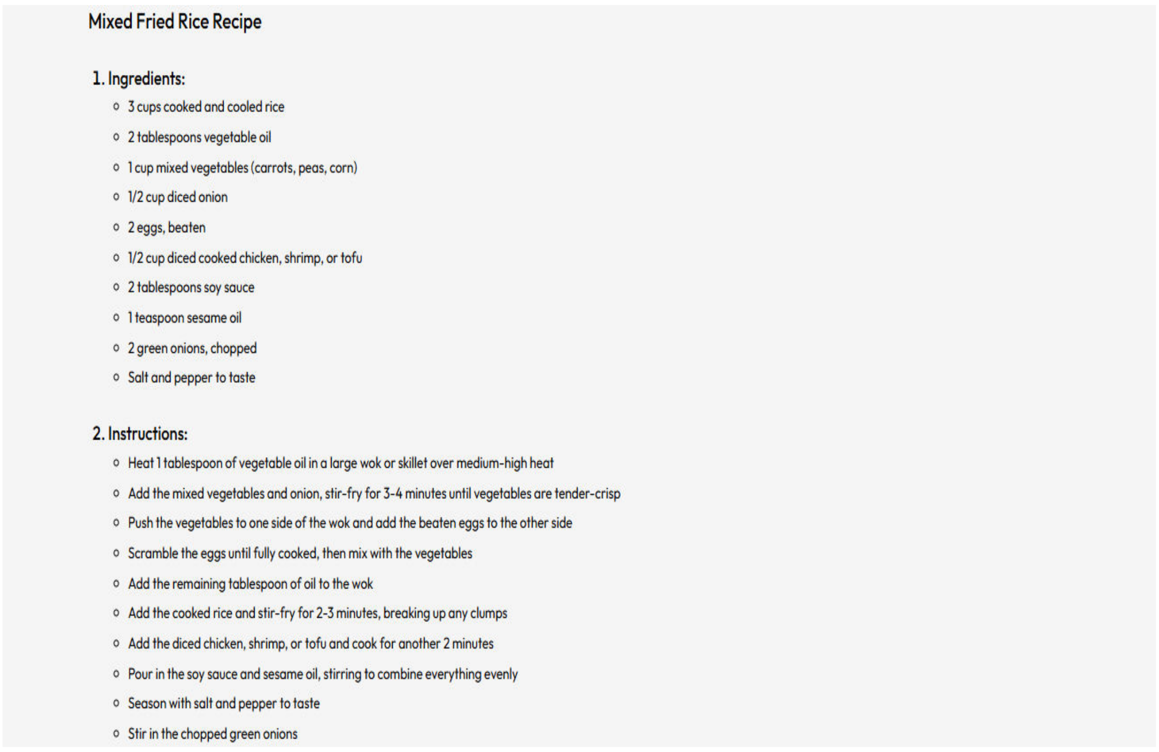


Figure 7.3.(a) Recipe generating



Mixed Fried Rice Recipe

1. Ingredients:
   - 3 cups cooked and cooled rice
   - 2 tablespoons vegetable oil
   - 1 cup mixed vegetables (carrots, peas, corn)
   - 1/2 cup diced onion
   - 2 eggs, beaten
   - 1/2 cup diced cooked chicken, shrimp, or tofu
   - 2 tablespoons soy sauce
   - 1 teaspoon sesame oil
   - 2 green onions, chopped
   - Salt and pepper to taste
2. Instructions:
   - Heat 1 tablespoon of vegetable oil in a large wok or skillet over medium-high heat
   - Add the mixed vegetables and onion, stir-fry for 3-4 minutes until vegetables are tender-crisp
   - Push the vegetables to one side of the wok and add the beaten eggs to the other side
   - Scramble the eggs until fully cooked, then mix with the vegetables
   - Add the remaining tablespoon of oil to the wok
   - Add the cooked rice and stir-fry for 2-3 minutes, breaking up any clumps
   - Add the diced chicken, shrimp, or tofu and cook for another 2 minutes
   - Pour in the soy sauce and sesame oil, stirring to combine everything evenly
   - Season with salt and pepper to taste
   - Stir in the chopped green onions

Figure 7.3.(b)Recipe generating

7.2.4 Image uploading

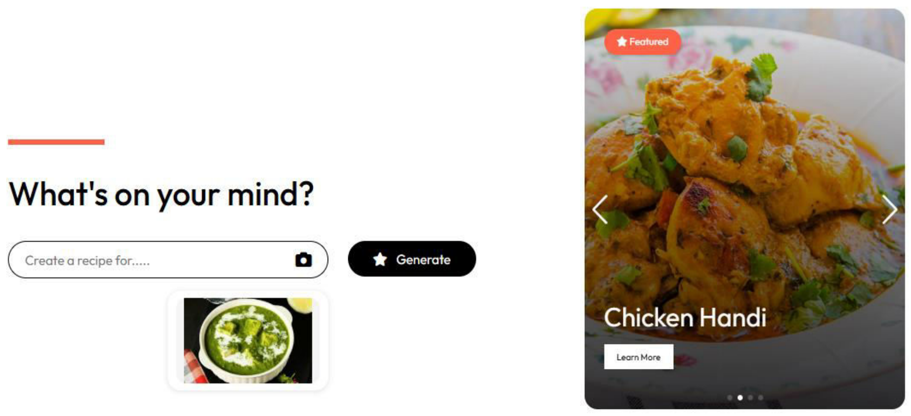
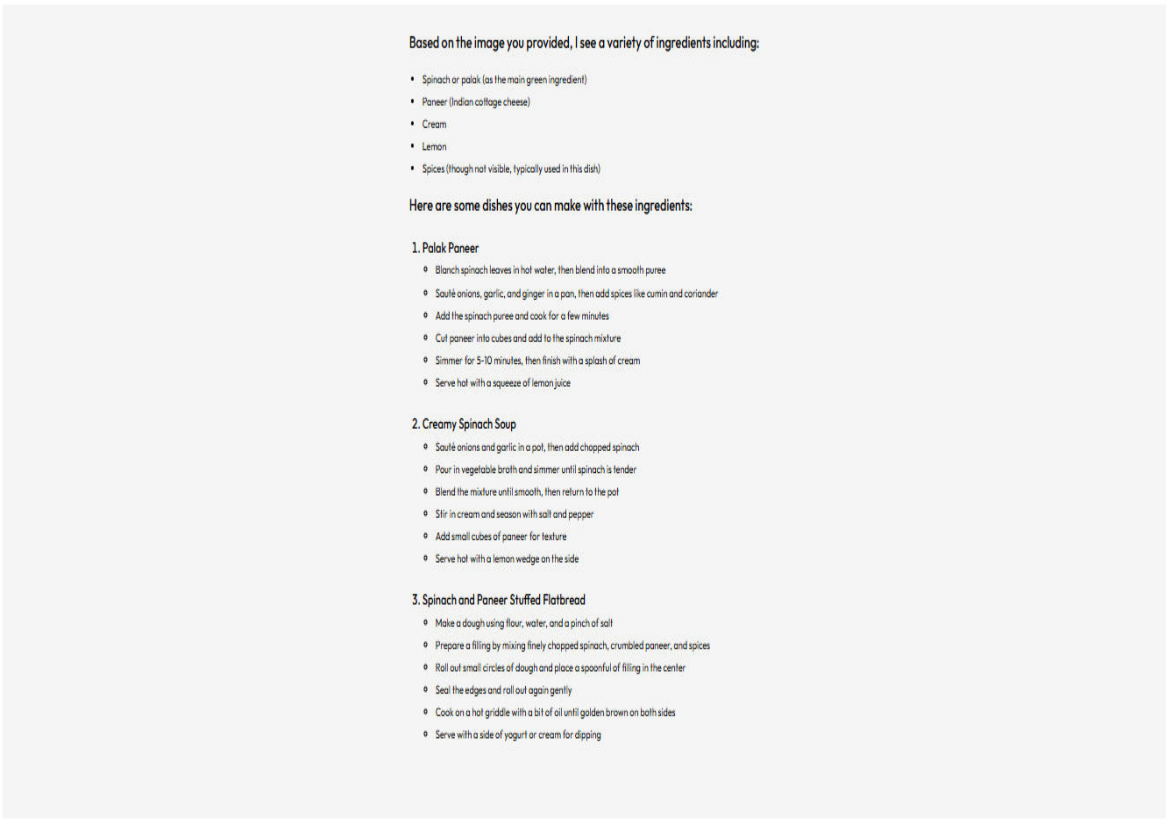


Figure 7.4.(a) Image uploading



Based on the image you provided, I see a variety of ingredients including:

- Spinach or palak (as the main green ingredient)
- Paneer (Indian cottage cheese)
- Cream
- Lemon
- Spices (though not visible, typically used in this dish)

Here are some dishes you can make with these ingredients:

1. Palak Paneer
   - Blanch spinach leaves in hot water, then blend into a smooth puree
   - Sauté onions, garlic, and ginger in a pan, then add spices like cumin and coriander
   - Add the spinach puree and cook for a few minutes
   - Cut paneer into cubes and add to the spinach mixture
   - Simmer for 5-10 minutes, then finish with a splash of cream
   - Serve hot with a squeeze of lemon juice

2. Creamy Spinach Soup
   - Sauté onions and garlic in a pot, then add chopped spinach
   - Pour in vegetable broth and simmer until spinach is tender
   - Blend the mixture until smooth, then return to the pot
   - Stir in cream and season with salt and pepper
   - Add small cubes of paneer for texture
   - Serve hot with a lemon wedge on the side

3. Spinach and Paneer Stuffed Flatbread
   - Make a dough using flour, water, and a pinch of salt
   - Prepare a filling by mixing finely chopped spinach, crumbled paneer, and spices
   - Roll out small circles of dough and place a spoonful of filling in the center
   - Seal the edges and roll out again gently
   - Cook on a hot griddle with a bit of oil until golden brown on both sides
   - Serve with a side of yogurt or cream for dipping

Figure 7.4.(b) Image uploading

Chapter 8: EVALUATION AND PERFORMANCE ANALYSIS

8.1 CONFUSION MATRIX

Total Data : 13567

Table 8.1 Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	5888 (TP)	130 (FP)
Actual Negative	278 (FN)	7271 (TN)

Accuracy: $= \frac{Total number of correct predictions}{Total number of instances}$

Accuracy = 97%

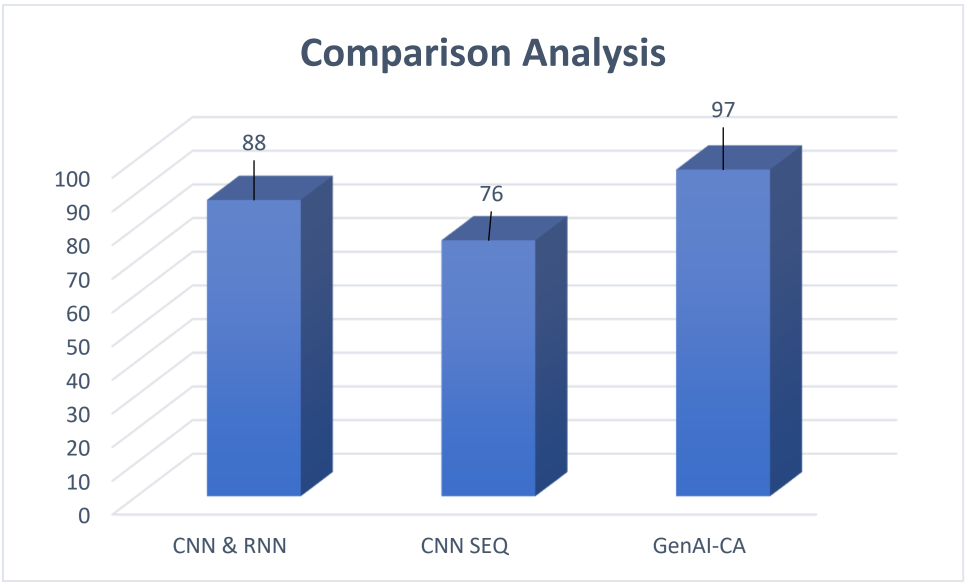


Figure 8.1: Accuracy

Figure 8.1 is the comparison analysis that shows the GenAI-CA model outperforms all other architectures, achieving an accuracy of over 98%, significantly higher than CNN & RNN (88%), and

CNN SEQ (76%). The improved performance of GenAI-CA highlights its superior feature extraction, adaptability, and classification accuracy, making it a promising solution for complex AI tasks. The results confirm that integrating generative AI techniques enhances model efficiency, reducing error rates and improving generalization.

Precision: 97.8%

Accuracy: 97%

Recall: 95.5%

CHAPTER 9: CONCLUSION AND FUTURE WORKS

Conclusion:

In this project, we developed an AI-powered recipe generator that can create recipes based on both text and image inputs. The system effectively identifies food items or ingredients from images and generates relevant recipes, making cooking more accessible and convenient. By integrating machine learning and image recognition, the application enhances user experience by providing personalized and creative recipe suggestions. This project demonstrates the potential of AI in the culinary space, offering a seamless way for users to explore new dishes with minimal effort.

Future works:

Future enhancements will focus on integrating smart kitchen appliances for real-time cooking assistance and expanding the recipe database to include diverse cuisines and dietary-specific options. A subscription-based model will offer premium features like advanced meal planning and personalized recommendations. Multilingual support and voice-assisted cooking guides will improve accessibility for a global audience. AI advancements will enhance ingredient recognition accuracy and context-aware recipe suggestions. A mobile-friendly version will increase accessibility, while community-driven features like social sharing and collaborative cooking will boost user engagement. These upgrades will transform the application into a comprehensive AI-powered cooking assistant.

CHAPTER 10: REFERENCE

[1] T. R. Lekhaa, “Recipe Generation Based on Food Image Using Machine Learning,”2024.

[2] P. Chhikara, Dhiraj Chaurasia, Yifan Jiang, Omkar Masur, and Filip Ilievski, 1Information Scien “FIRE: Food Image to Recipe Generation,” 2024.

[3] R. Solanke, “Recipe Generation from Food Images Using Deep Learning,” 2023.

[4] D. Noever, “The Multimodal and Modular AI Chef: Complex Recipe Generation from Imagery,” 2023.

[5] A. E, S. Nandhini, H. K. Palani and M. M. C, "Recipe Recommendation Using Image Classification," 2023 5th International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2023, pp. 292-295, doi: 10.1109/ICIRCA57980.2023.10220811.

[6] M. Zhang, “Multimodal Feature Fusion and Exploitation with Dual Learning and Reinforcement Learning for Recipe Generation,” Applied Soft Computing, 2022. https://doi.org/10.1016/j.asoc.2022.109281

[7] C. R. Wijaya, “Food Text-to-Image Synthesis Using VQGAN and CLIP,” 2023.

[8] “A Rich Recipe Representation as Plan to Support Expressive Multi-Modal Queries on Recipe Content and Preparation Process,”2022.

[9] “M. Fontanellaz, "Self-Attention and Ingredient-Attention Based Model for Recipe Retrieval from Image Queries,”, IEEE, 2019.

[10] S. S. Chang, “Device, Method, and System for Recipe Recommendation and Recipe Ingredient Management,” 2012.