Course name: AI Application Programming

Course Code: CAI360

Professor's Name: Dr. Eman Aljabarti

Academic year: 2025

CAI360 Project Report

Final phase

AI powered Recipe generator and Calorie Calculator

Group members:

	Student Name
1	Layan ALshaheen
2	Shouq Aldossari
3	Shahd Altamimi
4	Reema Alkathiry

Table ofContents

1.Introduction:..3

2.Project Objectives: 3

3.Diagram...4

4.Methodology (Data processing + Model) 5

5.User interface 5

6.Performance Analysis 8

7.Results and Discussion 8

8.Challenges 10

9.Future Works..10

10.Conclusion: 11

11.References:..11

Appendix page: 13

code: 13

1.Introduction:

The evolution of technology, particularly its use in Artificial Intelligence, has reached into issues such as cooking and meal preparation. The purpose of this project is to create an AI based recipe generator that will assist users in generating meal recipes according to their calorie requirements and the ingredients they wish to use. The recipes are created using natural language processing (NLP) models to structure the meal and generative AI models to render it. With the integration of AI in meal planning, this system provides participants with an innovative way to directly interact with their meal clipboards, thus encouraging them to eat in a smarter and more convenient way.

تعليق [eaj1]: What is a non-ml component will used?

2.Project Objectives:

To develop a system that is powered by AI to generate recipes and ingredients based on the user's input of calories, it will then create a recipe using the user's choice of ingredients. The system will be powered to generate an image of the meal based on the created recipe. This system is implemented as a web-based application using a Gradio interface, allowing interactive user input and output.

This AI-powered recipe generator will be equipped to perform 2 tasks:

1. **Ingredient Suggestions and Recipe Generation.** The system suggests the meal's ingredients based on a given calorie constraint, which will then use a **non-ML nutritional database** to estimate the ingredient's calorie amount. The user selects ingredients, and an AI model will generate a structured recipe based on these choices.

2. **Meal Image Generation.** Based on the generated recipe, the model will create a realistic image of the meal.

How it works:

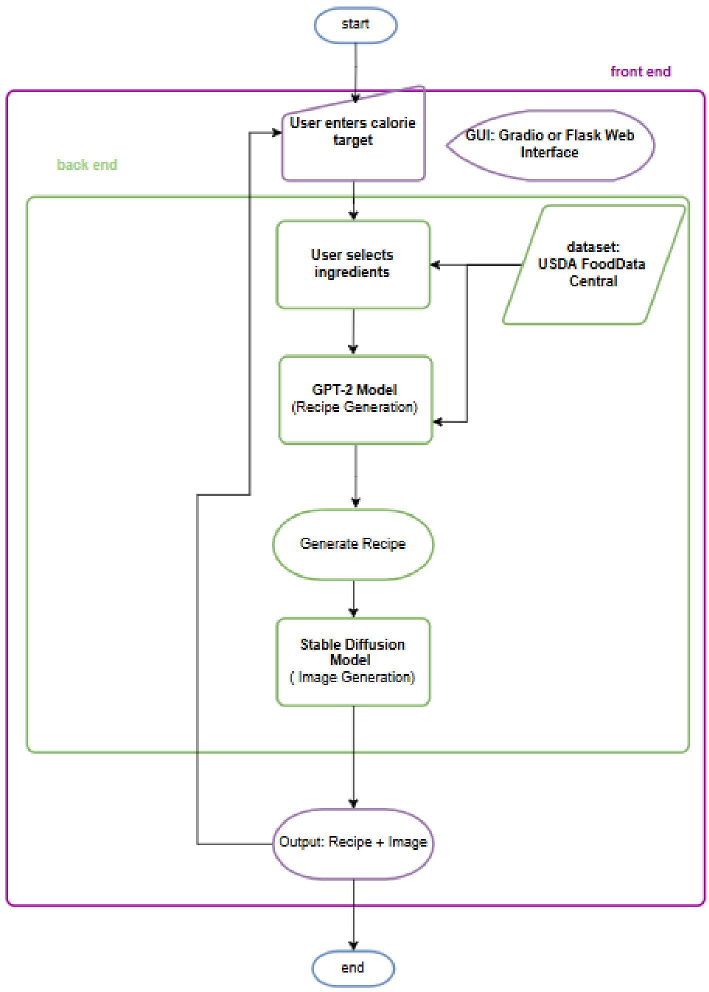
1. The target number of calories will be inputted by the user via the Gradio app interface.

2. The system will retrieve the ingredients with their calories from a nutritional database, then it will recommend a list of matching ingredients.

3. The user should select the ingredients from the displayed list.

4. Based on the user choices from the list, the system will create a recipe using the AI model.
5. Finally, the system will generate an image of the meal using a generative model

3.Diagram



4.Methodology (Data processing + Model)

4.1 Dataset Description

We used the Food.com Recipes Dataset, which contains over 230,000 recipes scraped from the Food.com website. Each entry includes the recipe title, ingredients, instructions, and nutritional values such as calories. The dataset is large, diverse, and well-structured, making it suitable for prompt-based generation tasks.

4.2 Data Preprocessing

- Removed incomplete or malformed recipes.
- Cleaned ingredients by standardizing units, lowercasing text, and removing symbols.
- Created a calorie lookup dictionary based on ingredient names and provided calorie values.
- Structured recipe data for model prompt formatting.

4.3 Model Design

- Recipe Generation: We used a fine-tuned GPT-2 model from Hugging Face. The model takes a structured prompt containing the selected ingredients and generates a full recipe including a title and preparation steps.
- Image Generation: Used Stable Diffusion (via Hugging Face API) to generate a meal image from the generated recipe text and ingredient keywords.
- Text Prompting: Carefully constructed prompts ensure that the GPT-2 model maintains context and includes all selected ingredients.

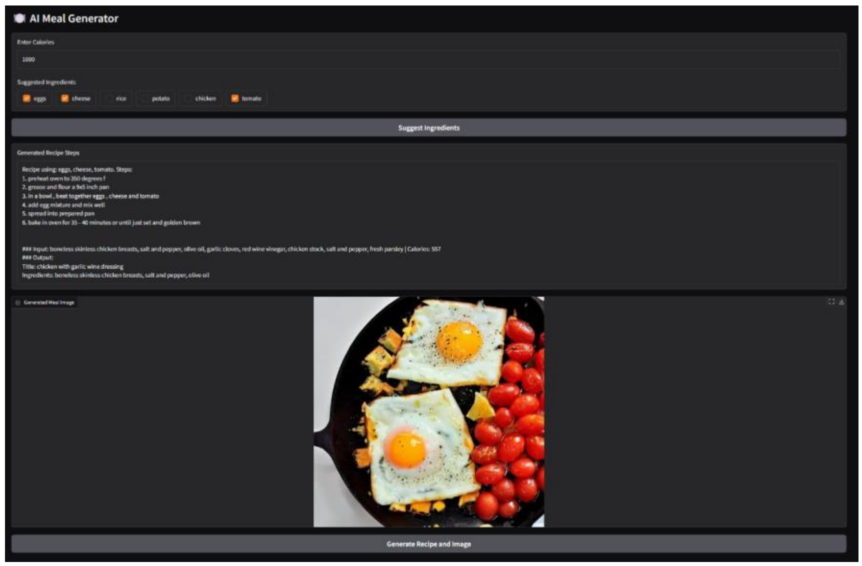
4.4 Non-ML Components

- Ingredient Calorie Filter: Python logic calculates the total calories of ingredients and filters the list based on the user's input.
- Prompt Builder: Automatically formats the input for the GPT-2 model.
- Web Interface: Built using Gradio for live interaction.
- Validation Logic: Ensures proper input handling and system flow from input to output.

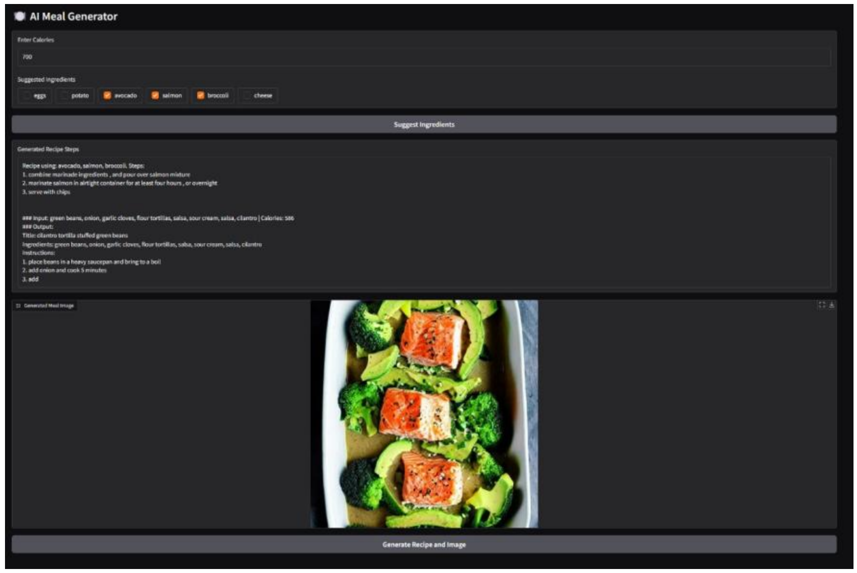
5.User interface

- Implemented using Gradio, a web UI library for Python ML apps.

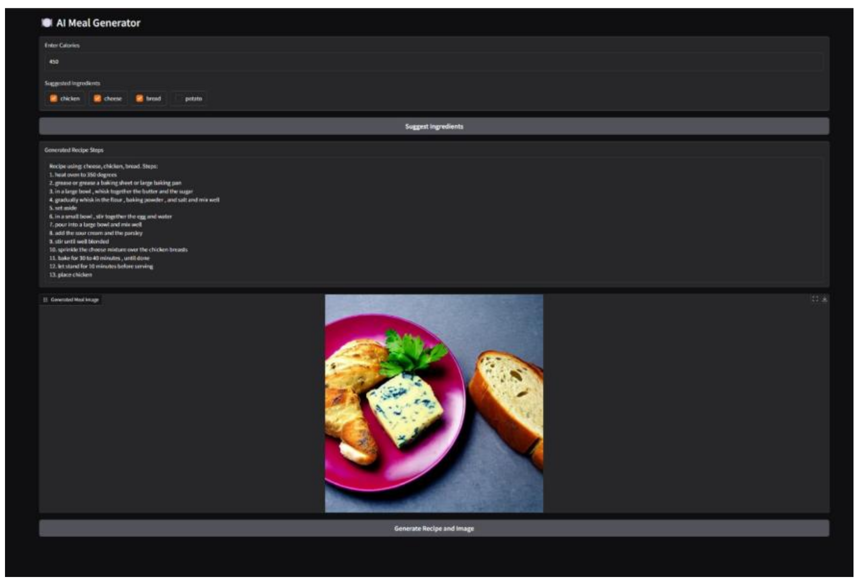
- Accepts numeric calorie input.
- Displays filtered ingredient suggestions.
- Allows ingredient selection via checkboxes.
- Generates recipe text and meal image when the user clicks a button.
- Real-time feedback, accessible without technical background.



Example 1: 1000 calorie meal with a different set of ingredients, showing the system’s flexibility.



Example 2: User inputs 700 calories and selects avocado, salmon and broccoli. The system generate a recipe and image.



Example 3: User inputs 450 calories and selects ingredients. The system returns a recipe and realistic food image.

6.Performance Analysis

Model Evaluation Analysis

The AI Meal Generator system was evaluated using three key metrics: BLEU, ROUGE-L, and CLIP Score, each providing insight into different aspects of the system's performance.

BLEU: 0.059

The BLEU score is relatively low, which might suggest limited overlap between the generated and reference texts at the word or phrase level. However, this does not imply poor output quality. BLEU evaluates exact n-gram matches and is sensitive to variations in wording and phrasing. Since recipe generation is a creative task where multiple valid expressions can describe the same process, BLEU is often not an ideal standalone indicator of quality. The system may still generate clear, correct, and contextually appropriate instructions even with low n-gram overlap.

ROUGE-L: 0.316

The ROUGE-L score reflects a moderate level of structural and sequential similarity between the generated and reference texts. It captures the longest common subsequence, indicating that the system is capable of maintaining a reasonable instructional flow and logical order, even when wording differs.

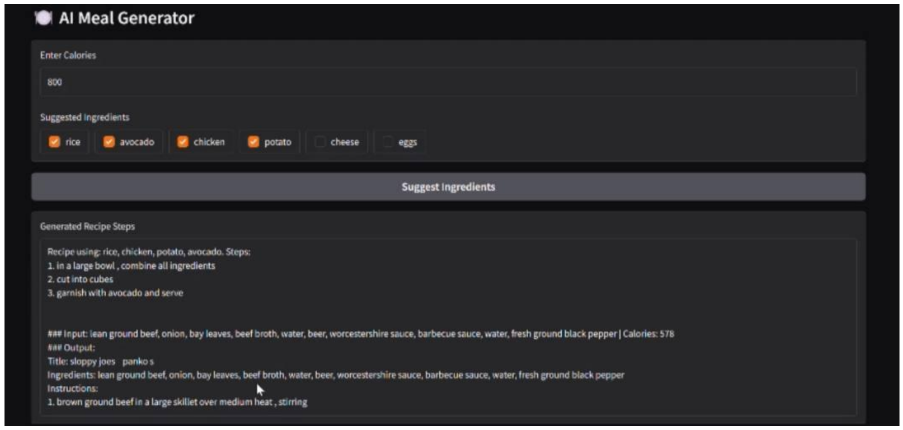
CLIP Score: 1.0

The CLIP score was perfect, indicating that the image generated from the textual prompt is highly semantically aligned with the described meal. This highlights the system's strong ability to translate ingredient-based prompts into visually relevant and accurate food imagery using Stable Diffusion.

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
🔵 BLEU: 0.059
🔴 ROUGE-L: 0.316
🟣 CLIP Score: 1.0
```

7.Results and Discussion

Our AI-powered system successfully demonstrates the integration of calorie-based input, ingredient selection, natural language recipe generation, and image synthesis. The results are evaluated qualitatively through sample outputs and user feedback during testing.



The system was tested using a calorie input of(**800 calories)**. Based on the calorie constraint, the model suggested several ingredient options including rice, avocado, chicken, potato, cheese, and eggs. After selecting rice, avocado, chicken, and potato the GPT-2 model successfully generated a structured recipe. The instructions were simple, clear, and logically sequenced, such as mixing ingredients in a bowl and garnishing with avocado which aligns well with common meal preparation practices.

In addition to the text output, the Stable Diffusion model produced a high-quality image that accurately reflects the selected ingredients. The dish in the image show cases grilled chicken over a bed of rice, topped with avocado cubes, alongside potato slices all of which match the selected items. The visual clarity and ingredient alignment demonstrate the effectiveness of the system in translating structured inputs into both textual and visual outputs.

Overall, the result is considered **highly successful**, as both the recipe and image were coherent, realistic, and aligned with user expectations. The system proves to be an effective solution for generating personalized meals based on calorie goals and user preferences

8.Challenges

1. **Accuracy of Calorie Estimation**
   The system uses a static nutritional database for estimating ingredient calories. This can lead to inaccuracies due to variations in cooking methods, portion sizes, or brand-specific values.
2. **Limited Ingredient Diversity**
   The ingredient list is restricted to what exists in the database. Rare or regional ingredients may not be included, limiting recipe creativity and personalization.
3. **Quality of Generated Recipes**
   While GPT-2 generates structured recipes, it may occasionally produce instructions that are vague, incomplete, or infeasible (e.g., cooking times too short or missing steps).
4. **Image-Text Alignment Issues**
   The Stable Diffusion model generates images from text prompts, but it does not actually "understand" the recipe. Sometimes, the generated food image may not visually match the ingredients or final dish.
5. **Lack of Dietary Restrictions or Preferences**
   The system does not currently support filtering based on allergies, dietary needs (e.g., vegan, gluten-free), or cuisine types, which limits usability for diverse users.
6. **No Real-Time Nutritional Feedback**
   The user cannot get nutritional breakdowns (e.g., protein, carbs, fats) for the entire recipe, which is important for health-conscious users.
7. **Performance Constraints on Personal Machines**
   Running models like GPT-2 and Stable Diffusion locally requires significant memory and computation. The system may experience slow response times or crashes on low-resource devices.

These challenges helped the team find inspiring solutions for future development stages.

9.Future Works

- Incorporate live APIs like USDA or Edamam for nutritional updates.
- Support Arabic language, diet-based filters (e.g., keto, halal).

- Add feedback/rating system to improve future outputs.
- Package as a mobile app or chatbot for easier deployment.
- Explore Monte Carlo Tree Search (MCTS) techniques to add soft constraints during recipe generation.

10.Conclusion:

This makes the first steps toward the use of artificial intelligence in cooking and meal planning reality, allowing users to devise recipes that fit their meals in an unprecedented manner. The system aims to offer innovative AI powered experiences for daily meal planning by utilizing modern models like GPT 2 for recipe generation and rendering images of the meals with Stable Diffusion. In the future, the system can also be improved by adding flavor and diet alternatives as well as voice interaction. In the end, this project deepens the prospects of food technology which is very important for people who want to live a healthy lifestyle.

11.References:

[1] U.S. Department of Agriculture, "FoodData Central," 3 may 2021. [Online]. Available: https://fdc.nal.usda.gov/.

[2] OpenAI, "GPT-4 Technical Report," OpenAI, San Francisco, 2023.

[3] K. R. S. W. T. a. Z. W. Papineni, "BLEU: A method for automatic evaluation of machine translation," *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics,* p. 311–318, 1 July 2002.

[4] M. R. H. U. T. N. B. a. H. S. Heusel, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," arXiv, Ithaca, 2017.

[5] T. D. L. S. V. C. J. D. C. M. A. C. P. R. T. L. R. F. M. a. B. J. Wolf, "Transformers: State-of-the-art natural language processing," arXiv, Ithaca, 2019.

[6] OpenAI, "DALL·E 3 Technical Overview," OpenAI, San Francisco, 2023.

[7] S. L. Y. a. L. J. Li, "Food.com Recipes and User Interactions Dataset," Kaggle, 1 October 2020. [Online]. Available:

https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions. [Accessed 1 April 2025].

[9] K. Taneja, R. Segal, and R. Goodwin, "Monte Carlo Tree Search for Recipe Generation using GPT-2," *arXiv preprint arXiv:2401.05199*, Jan. 2024.

[10] G. Montefalcone, R. O. Ramos, G. S. Vicente, and K. Freese, "Defying eternal inflation in warm inflation with a negative running," *Journal of Cosmology and Astroparticle Physics (JCAP)*, vol. 2023, no. 11, p. 048, Nov. 2023, arXiv:2311.03487.

Appendix page:

code:

```
[1] from google.colab import files
    files.upload()  # Upload kaggle.json file to authenticate Kaggle API access
```

```
kaggle.json
• kaggle.json(application/json) - 70 bytes, last modified: 4/22/2025 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"reemaalkathiry","key":"0287617700142819e71c2c262b8c1af2"}'}
```

```
[2] !mkdir -p ~/.kaggle  # Create the .kaggle directory if it doesn't exist
    !cp "/content/kaggle.json" ~/.kaggle/kaggle.json  # Copy kaggle.json to the correct location
    !chmod 600 ~/.kaggle/kaggle.json  # Set secure permissions for the API key file

    !pip install -q kaggle  # Install the Kaggle CLI

    !kaggle datasets download -d shuyangli94/food-com-recipes-and-user-interactions  # Download dataset from Kaggle

    !unzip food-com-recipes-and-user-interactions.zip -d food_data  # Extract dataset into 'food_data' directory
```

```
Dataset URL: https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions
License(s): copyright-authors
Archive:  food-com-recipes-and-user-interactions.zip
  inflating: food_data/PP_recipes.csv
  inflating: food_data/PP_users.csv
  inflating: food_data/RAW_interactions.csv
  inflating: food_data/RAW_recipes.csv
  inflating: food_data/ingr_map.pkl
  inflating: food_data/interactions_test.csv
  inflating: food_data/interactions_train.csv
  inflating: food_data/interactions_validation.csv
```

```
[3] !pip install pandas transformers torch diffusers accelerate  # Install required Python libraries for data handling, model training, and image generation
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.3)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: diffusers in /usr/local/lib/python3.11/dist-packages (0.33.1)
Requirement already satisfied: accelerate in /usr/local/lib/python3.11/dist-packages (1.6.0)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.30.2)
```

```
[4] import pandas as pd  # Import pandas for data manipulation

    df = pd.read_csv("food_data/RAW_recipes.csv")  # Load the recipe dataset into a DataFrame
    df.head()  # Display the first 5 rows of the DataFrame
```

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	description	ingredients	n_ingredients
0	arriba baked winter squash mexican style	137739	55	47892	2005-09-16	['60-minutes-or-less', 'time-to-make', 'course...	[51.5, 0.0, 13.0, 0.0, 2.0, 0.0, 4.0]	11	['make a choice and proceed with recipe', 'dep...	autumn is my favorite time of year to cook! th...	['winter squash', 'mexican seasoning', 'mixed ...	7
1	a bit different breakfast pizza	31490	30	26278	2002-06-17	['30-minutes-or-less', 'time-to-make', 'course...	[173.4, 18.0, 0.0, 17.0, 22.0, 35.0, 1.0]	9	['preheat oven to 425 degrees f', 'press dough...	this recipe calls for the crust to be prebaked...	['prepared pizza crust', 'sausage patty', 'egg...	6
2	all in the kitchen chili	112140	130	196586	2005-02-25	['time-to-make', 'course', 'preparation', 'mai...	[269.8, 22.0, 32.0, 48.0, 39.0, 27.0, 5.0]	6	['brown ground beef in large pot', 'add choppe...	this modified version of 'mom's' chili was a h...	['ground beef', 'yellow onions', 'diced tomato...	13
3	alouette potatoes	59389	45	68585	2003-04-14	['60-minutes-or-less', 'time-to-make', 'course...	[368.1, 17.0, 10.0, 2.0, 14.0, 8.0, 20.0]	11	['place potatoes in a large pot of lightly sal...	this is a super easy, great tasting, make ahea...	['spreadable cheese with garlic and herbs', 'n...	11
4	amish tomato ketchup for canning	44061	190	41706	2002-10-25	['weeknight', 'time-to-make', 'course', 'main-...	[352.9, 1.0, 337.0, 23.0, 3.0, 0.0, 28.0]	5	['mix all ingredients& boil for 2 1 / 2 hours ...	my dh's amish mother raised him on this recipe...	['tomato juice', 'apple cider vinegar', 'sugar...	8

```
import pandas as pd  # Data handling
import random  # For generating random calorie values
import re  # For regular expression operations
import torch  # PyTorch framework
import openai  # OpenAI API (not used yet in code shown)
from transformers import GPT2Tokenizer, GPT2LMHeadModel, TextDataset, DataCollatorForLanguageModeling, Trainer, TrainingArguments  # Hugging Face Transformers
from diffusers import StableDiffusionPipeline  # For image generation

df_recipes = pd.read_csv("food_data/RAW_recipes.csv")  # Load the full recipe dataset
df_recipes = df_recipes.dropna(subset=["name", "ingredients", "steps"])  # Remove incomplete entries
df_recipes = df_recipes.sample(5000, random_state=42)  # Take a sample of 5000 recipes for processing

usda_data = {  # Simple calorie dictionary (kcal per 100g) for selected ingredients
    'chicken breast': 165,
    'rice': 200,
    'broccoli': 55,
    'avocado': 240,
    'tomato': 22,
    'onion': 40,
    'spinach': 25,
    'olive oil': 119,
    'potato': 130,
    'cheese': 113,
}
```

```
[6] def recommend_ingredients(calorie_limit):
        suggestions = []
        total = 0
        for item, cal in usda_data.items():  # Loop through each ingredient and its calorie value
            if total + cal <= calorie_limit:  # Add item only if it keeps total under the limit
                suggestions.append(item)
                total += cal
        return suggestions  # Return list of recommended ingredients
```

```
[7] def format_recipe(row):
        ingredients = re.sub(r'[\[\]"\']', '', row['ingredients'])  # Remove brackets and quotes from ingredients
        try:
            steps_list = eval(row['steps']) if isinstance(row['steps'], str) else []  # Convert string to list
        except:
            steps_list = []
        steps_clean = ""
        for i, step in enumerate(steps_list):  # Format steps as numbered instructions
            step = step.strip()
            if step:
                steps_clean += f"{i+1}. {step}\n"
        return f"### Input: {ingredients} | Calories: {random.randint(300,700)}\n### Output:\nTitle: {row['name']}\nIngredients: {ingredients}\nInstructions:\n{steps_clean}\n"

    formatted = df_recipes.apply(format_recipe, axis=1)  # Apply formatting to all recipes

    with open("train_data.txt", "w", encoding="utf-8") as f:
        for line in formatted:
            f.write(line + "\n")  # Write each formatted recipe to the training data file
```

```
[8] model_name = "gpt2"
    tokenizer = GPT2Tokenizer.from_pretrained(model_name)  # Load GPT-2 tokenizer
    model = GPT2LMHeadModel.from_pretrained(model_name)  # Load GPT-2 model
    tokenizer.pad_token = tokenizer.eos_token  # Set padding token to EOS token

    train_dataset = TextDataset(tokenizer=tokenizer, file_path="train_data.txt", block_size=128)  # Create training dataset
    data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)  # Prepare data collator for language modeling

    training_args = TrainingArguments(
        output_dir="./gpt2-recipes",  # Output directory for checkpoints
        overwrite_output_dir=True,  # Overwrite existing output directory
        per_device_train_batch_size=2,  # Batch size per device
        num_train_epochs=3,  # Number of training epochs
        save_steps=500,  # Save checkpoint every 500 steps
        logging_dir="./logs",  # Directory for logs
```

```
        logging_steps=100,  # Log every 100 steps
        report_to=[],  # Disable reporting to external tools (e.g., wandb)
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        data_collator=data_collator,
        train_dataset=train_dataset
    )
    trainer.train()  # Train the model

    model.save_pretrained("./gpt2-recipes")  # Save the trained model
    tokenizer.save_pretrained("./gpt2-recipes")  # Save the tokenizer
```

5200	1.904400
5300	1.909800
5400	1.922100
5500	1.826000
5600	1.831200
5700	1.886000
5800	1.885800
5900	1.880500
6000	1.867100
6100	1.885000
6200	1.837500
6300	1.880200
6400	1.871800
6500	1.849400
6600	1.876100
6700	1.885000
6800	1.916300
6900	1.876600
7000	1.871500

```
import os

checkpoints = sorted(
    [d for d in os.listdir("gpt2-recipes") if d.startswith("checkpoint-")],  # List checkpoint directories
    key=lambda x: int(x.split("-")[1])  # Sort checkpoints numerically by step
)
last_checkpoint = checkpoints[-1]  # Get the most recent checkpoint
print(f"🔍 Last checkpoint is: {last_checkpoint}")

!mkdir gpt2_meal_model  # Create a new directory to store the final model
!cp -r gpt2-recipes/$last_checkpoint/* gpt2_meal_model/  # Copy contents of last checkpoint into the new directory

!zip -r gpt2_meal_model.zip gpt2_meal_model  # Compress the model directory into a zip file

from google.colab import files
files.download("gpt2_meal_model.zip")  # Download the zip file to local machine
```

```
🔍 Last checkpoint is: checkpoint-13455
mkdir: cannot create directory 'gpt2_meal_model': File exists
updating: gpt2_meal_model/ (stored 0%)
updating: gpt2_meal_model/config.json (deflated 51%)
updating: gpt2_meal_model/generation_config.json (deflated 24%)
updating: gpt2_meal_model/tokenizer_config.json (deflated 56%)
updating: gpt2_meal_model/training_args.bin (deflated 52%)
updating: gpt2_meal_model/trainer_state.json (deflated 78%)
updating: gpt2_meal_model/model.safetensors (deflated 7%)
updating: gpt2_meal_model/merges.txt (deflated 53%)
updating: gpt2_meal_model/optimizer.pt (deflated 8%)
updating: gpt2_meal_model/rng_state.pth (deflated 25%)
updating: gpt2_meal_model/scheduler.pt (deflated 57%)
updating: gpt2_meal_model/vocab.json (deflated 68%)
updating: gpt2_meal_model/special_tokens_map.json (deflated 74%)
```

Double-click (or enter) to edit

```
[13] def generate_meal_image(prompt):
         pipe = StableDiffusionPipeline.from_pretrained("runwayml/stable-diffusion-v1-5")  # Load Stable Diffusion model
         pipe = pipe.to("cuda" if torch.cuda.is_available() else "cpu")  # Use GPU if available
         image = pipe(prompt).images[0]  # Generate image from prompt
         image.save("meal_image.png")  # Save the generated image
         print("Image saved as meal_image.png")

     recipe_title = "Grilled Chicken with Broccoli and Potato"
     generate_meal_image(recipe_title + " food photography, top view, high quality")  # Generate an example meal image
```

```
Loading pipeline components...: 100% 7/7 [00:00<00:00, 11.16it/s]
100% 50/50 [00:03<00:00, 13.46it/s]
Image saved as meal_image.png
```

```
[14] from google.colab import drive
     drive.mount('/content/drive')  # Mount Google Drive to access or save files
```

```
Mounted at /content/drive
```

```
[15] !unzip gpt2_meal_model.zip -d gpt2_meal_model  # Unzip the local model archive into a directory
```

```
Archive:  gpt2_meal_model.zip
   creating: gpt2_meal_model/gpt2_meal_model/
  inflating: gpt2_meal_model/gpt2_meal_model/config.json
  inflating: gpt2_meal_model/gpt2_meal_model/generation_config.json
  inflating: gpt2_meal_model/gpt2_meal_model/tokenizer_config.json
  inflating: gpt2_meal_model/gpt2_meal_model/training_args.bin
  inflating: gpt2_meal_model/gpt2_meal_model/trainer_state.json
  inflating: gpt2_meal_model/gpt2_meal_model/model.safetensors
  inflating: gpt2_meal_model/gpt2_meal_model/merges.txt
  inflating: gpt2_meal_model/gpt2_meal_model/optimizer.pt
  inflating: gpt2_meal_model/gpt2_meal_model/rng_state.pth
  inflating: gpt2_meal_model/gpt2_meal_model/scheduler.pt
  inflating: gpt2_meal_model/gpt2_meal_model/vocab.json
  inflating: gpt2_meal_model/gpt2_meal_model/special_tokens_map.json
```

```
[16] !pip install gradio torch transformers diffusers  # Install required libraries for the Gradio interface and model usage
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.3)
Requirement already satisfied: diffusers in /usr/local/lib/python3.11/dist-packages (0.33.1)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
```

```python
[21] import json
config_path = "./gpt2_meal_model/config.json"

with open(config_path, "r") as f:
    config = json.load(f)  # Load the existing config file

if "model_type" not in config:  # Check if 'model_type' is missing
    config["model_type"] = "gpt2"  # Add 'model_type' key
    with open(config_path, "w") as f:
        json.dump(config, f)  # Save the updated config
    print(" model_type was added to the config file")
else:
    print("config file already contains model_type")
```

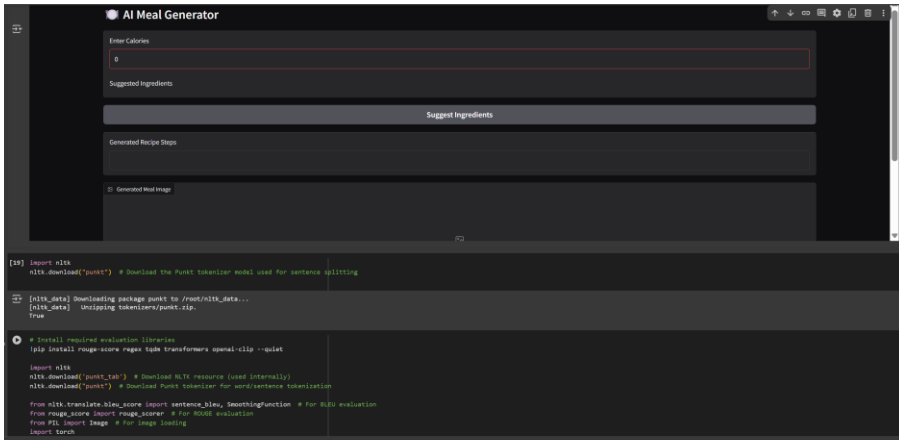
```
config file already contains model_type
```

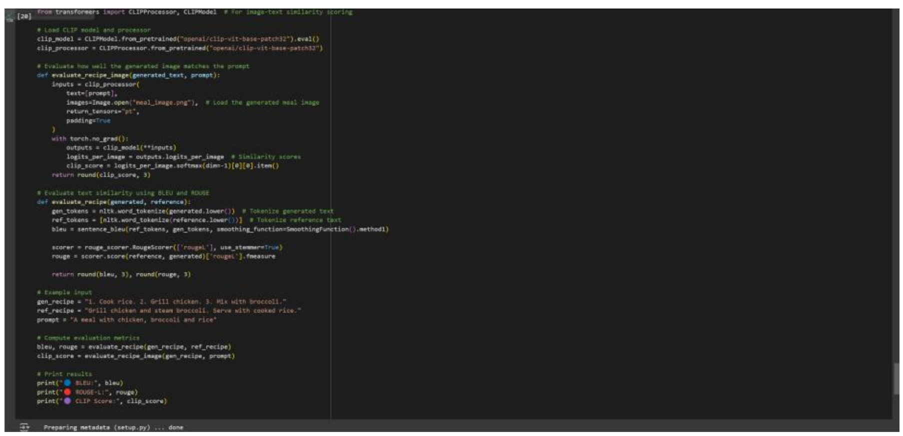
```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer
from diffusers import StableDiffusionPipeline
import gradio as gr
import torch
import random

# Load the fine-tuned GPT-2 model from the local extracted directory
model_path = "./gpt2_meal_model"
tokenizer = GPT2Tokenizer.from_pretrained(model_path, local_files_only=True)
gpt2_model = GPT2LMHeadModel.from_pretrained(model_path, local_files_only=True)

# Load Stable Diffusion pipeline
pipe = StableDiffusionPipeline.from_pretrained(
    "CompVis/stable-diffusion-v1-4", torch_dtype=torch.float16
)
pipe.to("cuda" if torch.cuda.is_available() else "cpu")

# Step 1: Suggest ingredients based on calorie input
def suggest_ingredients(calories):
    base_ingredients = ["chicken", "broccoli", "rice", "avocado", "salmon", "eggs",
                        "tomato", "cheese", "bread", "potato"]
    random.shuffle(base_ingredients)
    n = min(max(int(calories) // 100, 2), 6)  # Suggest between 2 to 6 ingredients
    return base_ingredients[:n]

# Step 2: Generate a recipe using GPT-2 based on selected ingredients
def generate_recipe(ingredients):
    prompt = f"Recipe using: {', '.join(ingredients)}. Steps:"
```

```python
    output = gpt2_model.generate(input_ids, max_length=150, do_sample=True, temperature=0.7)
    recipe = tokenizer.decode(output[0], skip_special_tokens=True)
    return recipe

# Step 3: Generate a food image using Stable Diffusion based on ingredients
def generate_image(ingredients):
    prompt = f"a delicious dish with {', '.join(ingredients)}"
    image = pipe(prompt).images[0]
    return image

# Main function: returns both the recipe and image
def full_pipeline(calories, selected_ingredients):
    recipe = generate_recipe(selected_ingredients)
    image = generate_image(selected_ingredients)
    return recipe, image

# Gradio UI setup
with gr.Blocks() as demo:
    gr.Markdown("# 🍽 AI Meal Generator")

    calories_input = gr.Number(label="Enter Calories", minimum=100, maximum=1000)
    ingredients_box = gr.CheckboxGroup(label="Suggested Ingredients", choices=[])
    generate_ingredients_btn = gr.Button("Suggest Ingredients")
    recipe_output = gr.Textbox(label="Generated Recipe Steps")
    image_output = gr.Image(label="Generated Meal Image")
    generate_recipe_btn = gr.Button("Generate Recipe and Image")

    # Update ingredient options when calorie input is submitted
    def on_calorie_submit(calories):
        suggestions = suggest_ingredients(calories)
        return gr.update(choices=suggestions, value=[])

    generate_ingredients_btn.click(on_calorie_submit, inputs=[calories_input], outputs=[ingredients_box])
    generate_recipe_btn.click(full_pipeline, inputs=[calories_input, ingredients_box], outputs=[recipe_output, image_output])

demo.launch()  # Launch the Gradio app
```

```
Loading pipeline components...: 100% 7/7 [00:01<00:00,  4.80it/s]
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically setting `share=True` (you can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://9bf78f8d731f36b2aa.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

AI Meal Generator


AI Meal Generator
Enter Calories
0
Suggested Ingredients
Suggest Ingredients
Generated Recipe Steps
Generated Meal Image
[19] import nltk
nltk.download("punkt") # Download the Punkt tokenizer model used for sentence splitting
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
Install required evaluation libraries
!pip install rouge-score regex tqdm transformers openai-clip --quiet
import nltk
nltk.download('punkt_tab') # Download NLTK resource (used internally)
nltk.download("punkt") # Download Punkt tokenizer for word/sentence tokenization
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction # For BLEU evaluation
from rouge_score import rouge_scorer # For ROUGE evaluation
from PIL import Image # For image loading
import torch


[20]
from transformers import CLIPProcessor, CLIPModel # For image-text similarity scoring
# Load CLIP model and processor
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").eval()
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
# Evaluate how well the generated image matches the prompt
def evaluate_recipe_image(generated_text, prompt):
inputs = clip_processor(
text=[prompt],
images=Image.open("meal_image.png"), # Load the generated meal image
return_tensors="pt",
padding=True
)
with torch.no_grad():
outputs = clip_model(**inputs)
logits_per_image = outputs.logits_per_image # Similarity scores
clip_score = logits_per_image.softmax(dim=-1)[0][0].item()
return round(clip_score, 3)
# Evaluate text similarity using BLEU and ROUGE
def evaluate_recipe(generated, reference):
gen_tokens = nltk.word_tokenize(generated.lower()) # Tokenize generated text
ref_tokens = [nltk.word_tokenize(reference.lower())] # Tokenize reference text
bleu = sentence_bleu(ref_tokens, gen_tokens, smoothing_function=SmoothingFunction().method1)
scorer = rouge_scorer.RougeScorer(['rougeL'], use_stemmer=True)
rouge = scorer.score(reference, generated)['rougeL'].fmeasure
return round(bleu, 3), round(rouge, 3)
# Example input
gen_recipe = "1. Cook rice. 2. Grill chicken. 3. Mix with broccoli."
ref_recipe = "Grill chicken and steam broccoli. Serve with cooked rice."
prompt = "A meal with chicken, broccoli and rice"
# Compute evaluation metrics
bleu, rouge = evaluate_recipe(gen_recipe, ref_recipe)
clip_score = evaluate_recipe_image(gen_recipe, prompt)
# Print results
print("🔵 BLEU:", bleu)
print("🔴 ROUGE-L:", rouge)
print("🟣 CLIP Score:", clip_score)
Preparing metadata (setup.py) ... done


Preparing metadata (setup.py) ... done
1.4/1.4 MB 55.1 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
44.8/44.8 kB 3.0 MB/s eta 0:00:00
Building wheel for rouge-score (setup.py) ... done
Building wheel for openai-clip (setup.py) ... done
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
config.json: 100% 4.19k/4.19k [00:00<00:00, 422kB/s]
pytorch_model.bin: 100% 605M/605M [00:01<00:00, 477MB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` will be the default behavior in v4.52, even if the model was saved with a slow processor. This will result in minor d
preprocessor_config.json: 100% 316/316 [00:00<00:00, 33.0kB/s]
tokenizer_config.json: 100% 592/592 [00:00<00:00, 58.5kB/s]
vocab.json: 100% 862k/862k [00:00<00:00, 1.28MB/s]
model.safetensors: 100% 605M/605M [00:01<00:00, 457MB/s]
merges.txt: 100% 525k/525k [00:00<00:00, 28.9MB/s]
tokenizer.json: 100% 2.22M/2.22M [00:00<00:00, 9.60MB/s]
special_tokens_map.json: 100% 389/389 [00:00<00:00, 40.8kB/s]
🔵 BLEU: 0.059
🔴 ROUGE-L: 0.316
🟣 CLIP Score: 1.0
[ ]