Name: Sunidi Vijayakrishna Kumar

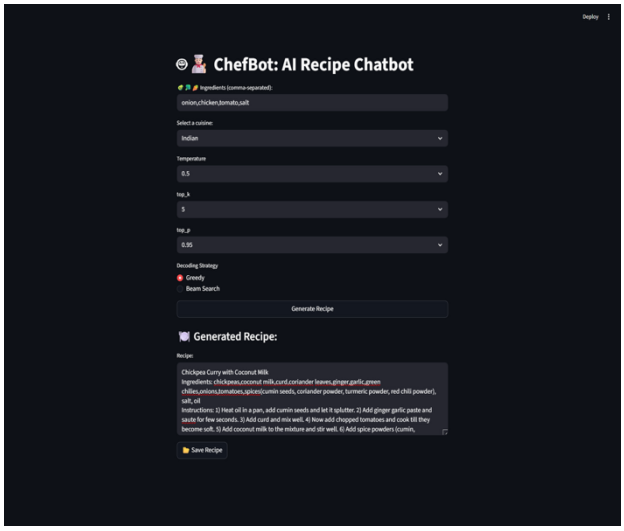
Nuid: 002474285

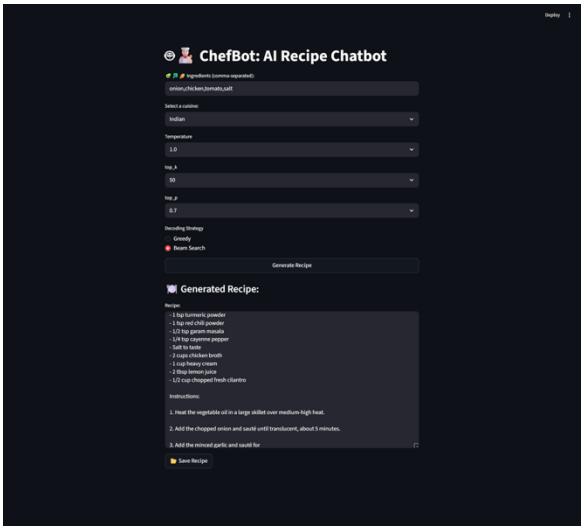
Homework 3

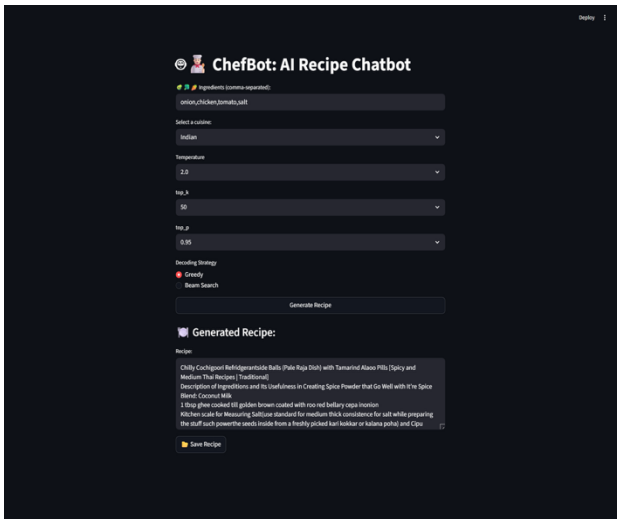
ChefBot: AI Recipe Chatbot

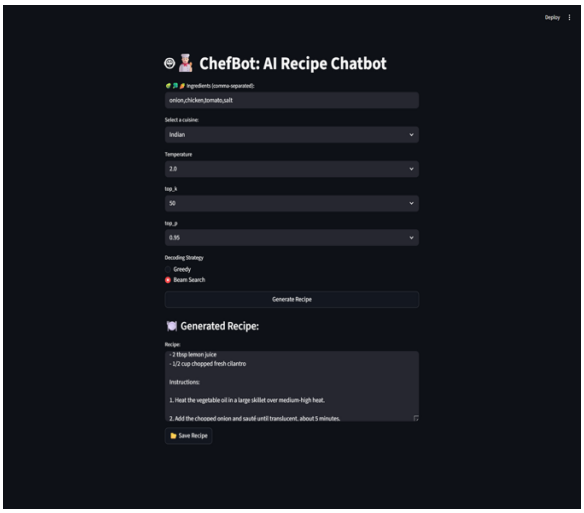
Task 1: Experimenting with Text Generation Parameters
Objective: Modify text generation parameters to understand their impact on AI-generated recipes.

Screenshots of the output:-









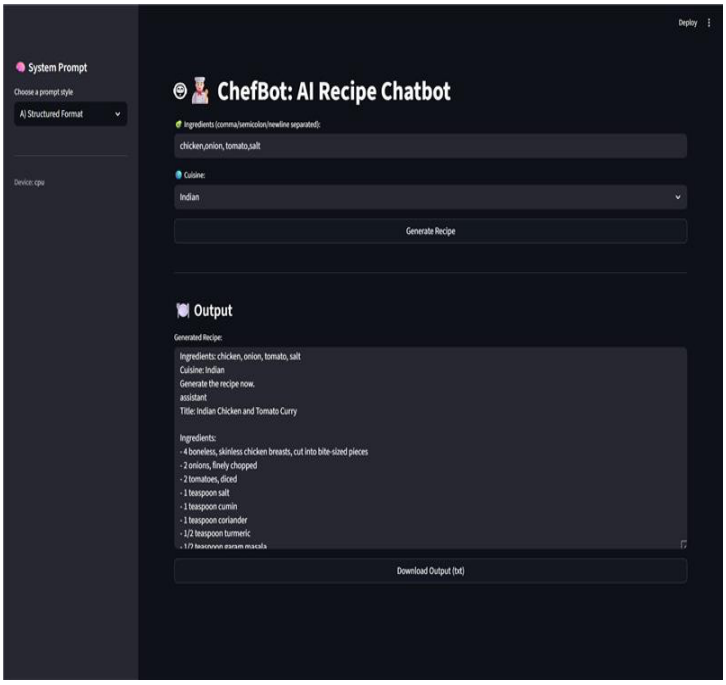
Observations: -

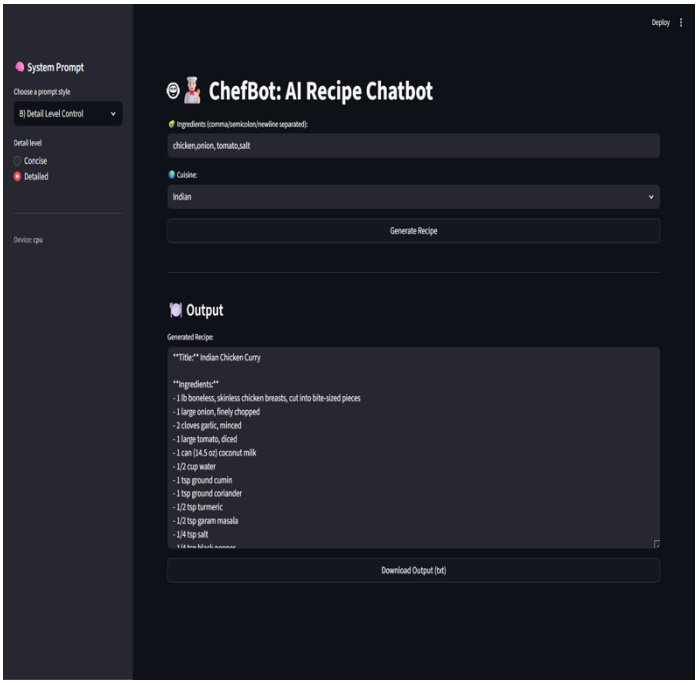
In task 1 we looked into different text generation parameters, structure of the recipes given by the language model. Given temperature parameter we can see different fluctuations in the result. As in the outcome, the temperature (0.5),the chatbot has generated more deterministic and repetitive recipes it has all direct instructions ,as increased the temperature to 1.0 the result can be seen that it was little coherent and also by the temperature increased to 2.0 the recipes became more creative ,and also it there me no relation or connection in between the ingredients and also less elaborative. Also, we have the sampling controls where we use top_k, top_p, in which lower top k value will put a limit to the probable tokens where it will give proper recipes. Also, the higher top k value (50) and top p(0.95) will have more descriptive and way of interpretation is also different. We also have the greedy decoding and beam search ,where greedy will give fast outputs but it does not complete properly the sentences or the instructions. Beam search gives more depth structured recipes with the clear ingredients and also instructions will be in an order. Hence overall we could see how the temperature and top k, top p values matter in generating the recipes from the chatbot impacting creativity diversity and the structure of the generated recipes.

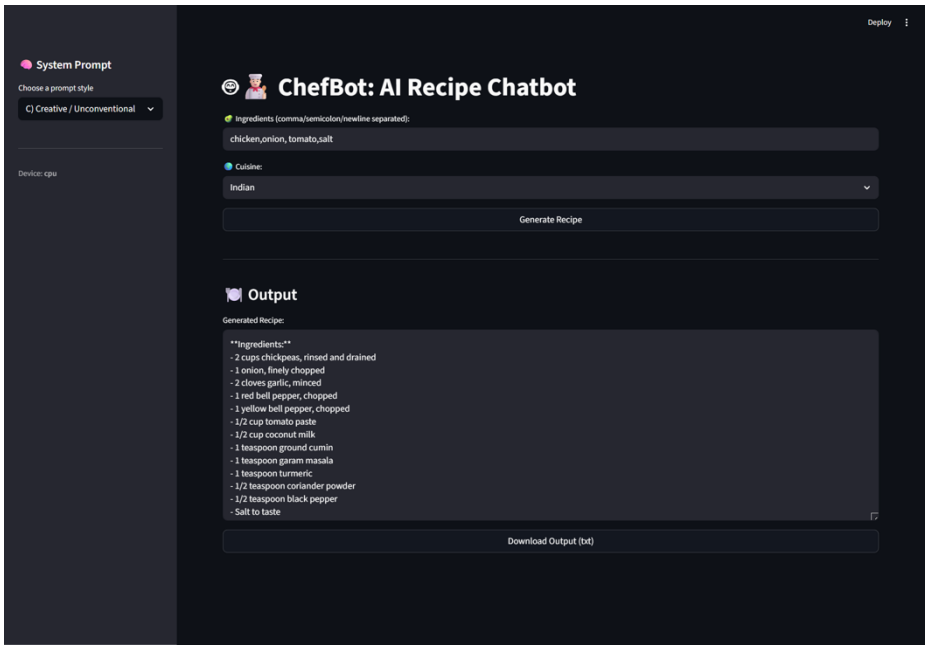
Task 2: Controlling AI Responses through System Prompts

Objective: Modify the system prompt to control format, detail level, and creativity in the AI-generated recipes.

Screenshots of the output:-







Observations:-

In this task 2 we have three different system prompt styles which is structured format ,detailed level control and also creative prompt. Here we can observe the clear difference on how it will influence the recipes based on the prompts.

Under structured format we can see that the model strictly follows the prompt or the instructions given the format is pre-defined and it produces the section title ingredients all of them by step by step.

In the detailed level control prompt, the recipes give shorter steps too short description with only few steps involved in the procedure, since the other one had little longer cooked instructions which demonstrated over long output length and depth.

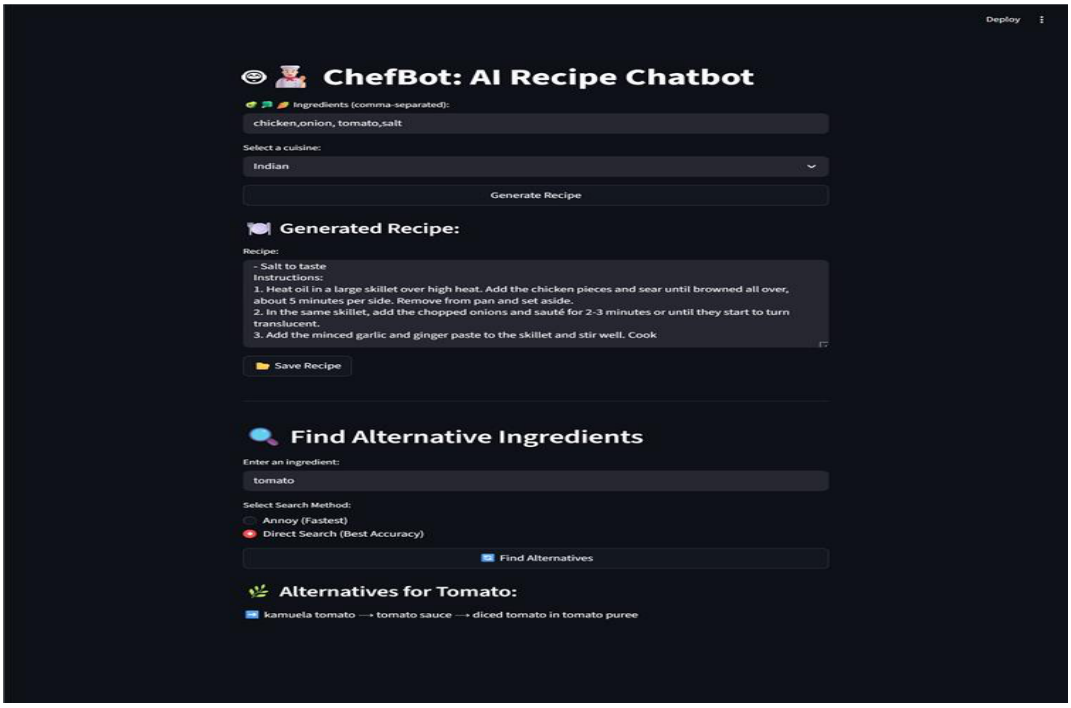
With the creative or unconventional prompt the model showed how it has the thinking ability and to think beyond the imagination of the unique ingredients combinations ,fusion dishes also it elaborated more in depth or descriptive language .This creativity might also reduce the structure of the procedure exhibited .But it shows way more unique dishes varieties and give creative recipes.

Hence these prompts will shape the LLM behavior which is structured prompts where we can read out the instructions properly, detailed prompt there is way too more depth and lengthy procedure, where as the creative prompts enhance the original format and gives more unique content .

Task 3: A Simple Alternative Ingredient Finder (Direct Search)

Objective: Add a feature that suggests alternative ingredients using direct search with cosine similarity.

Screenshots of the output:-



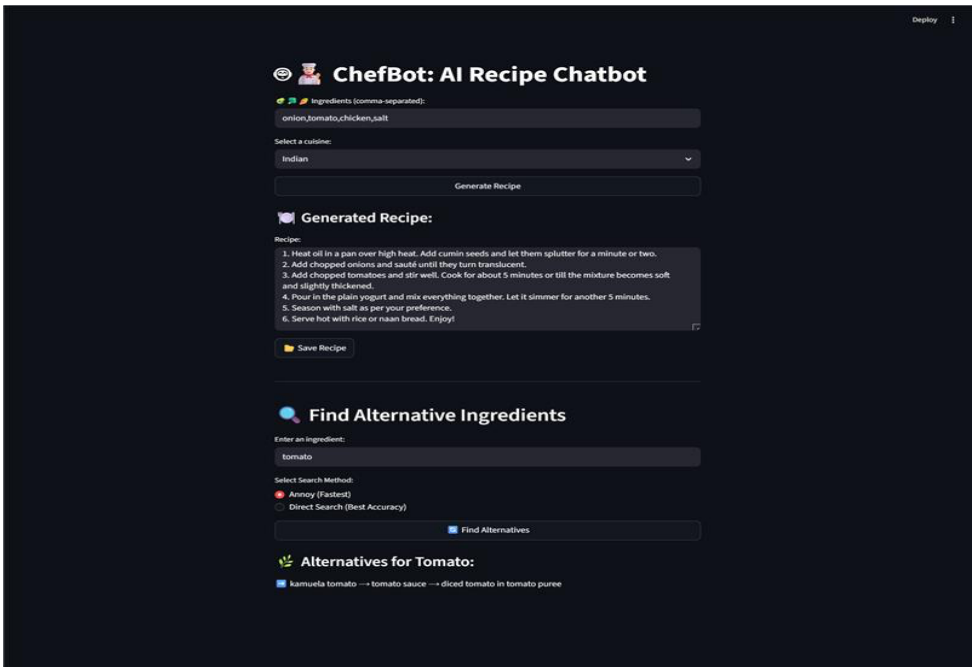
Observations:-

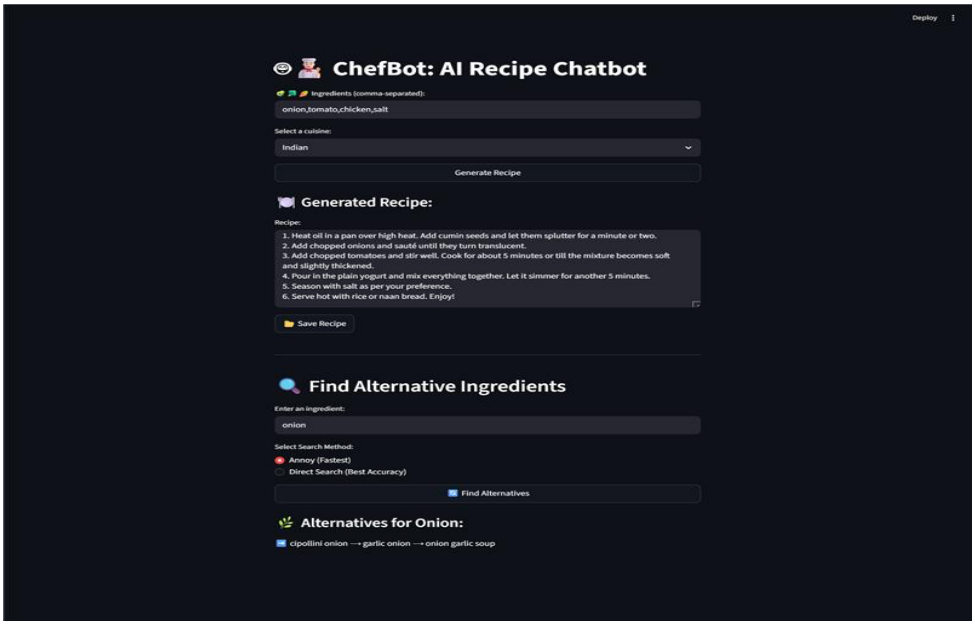
In task 3 ,we have implemented a direct cosine similarity search using spaCy word embeddings where we can find the alternative ingredients .in this method we have computed cosine similarity between the input ingredient vector and all the ingredients vectors in the dataset, where it returns the top most three which is the most suitable one it puts forth the result. From the above picture or the screenshot we can see that I have given tomato where it suggested the alternative has tomato sauce or tomato puree where it resulted in the similarity search where it successfully selects the semantic meaning ,it also forms the ingredient where we need to effectively the similarity. The direct search is very accurate where come\mutationally very slow and also compares the query vector which is against the vector. In the dataset it could provide proper details since it was small dataset and also it may require more alternatives and also better results if it is for the larger datasets.

Task 4: Faster Alternative Ingredient Finder (ANN by Annoy)

Objective: Improve search speed and user experience by using Approximate Nearest Neighbors (ANN) with Annoy.

Screenshots of the output:-





Observations:-

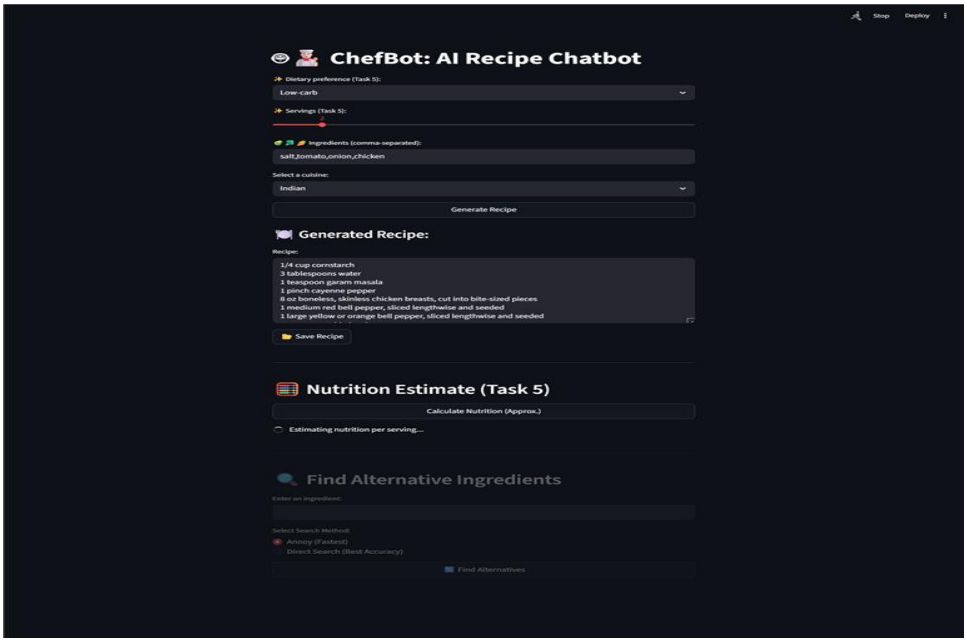
In the task 4,implemented ingredient substitution using Annoy search. Its different from the direct cosine similarity ,Annoy here it builds a tree based index of ingredient embeddings where it allows the faster retrieving the data. From the screenshot we can see that tomato alternative showed the same result as the same direct cosine since it build or retrieved the same alternative as best but it was complied faster when compared to the previous direct search .Hence I tried out with the onion too it had different type of onion for the replacement of the actual onion which showed some variations.where as the tomato result did not show any variation since it was the toppest.

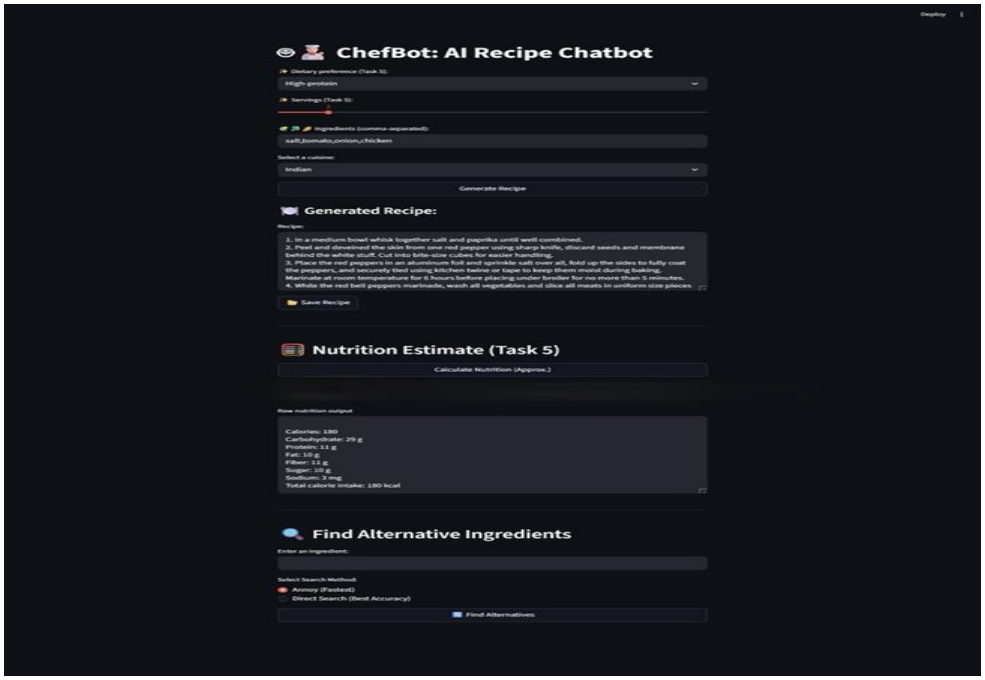
Performance if we compare Annoy was the fastest since it does not compare the query against each query the implementation is different ,it retrieves the nearest neighbors where it is efficient when we build using the pre built index. The results will change it may be less precise than the direct search but it is more scalable and the speed improvement can be consideration in case we have larger datasets. Hence Annoy will provide a strong balance between accuracy and performance.

Task 5: Let’s add ONE new AI-powered Feature of Your Choice

Objective: Extend the app with a new feature of your choosing

Screenshots of the output:





Observations:-

In task 5 added new feature is The Dietary preference and also automatic nutrition estimation calculator.Chefbot has been extended where the users can select dietary restrictions such as vegan,vegetarian,gluten free ,low carb, or high protein. The system modifies the generation prompt so that the ingredient will look upon the selected oreferences the model will update it appropriately,this will increase the personalization and also usability.

Also Nutrition estimate feature is added it is for the gym people or the fitness freak to keep the track of the nutrients and also to choose to eat or not. It would be easier for the user to willing to eat .Here it calculates the approximate per serving values for calories,carbohydrates,protein,fat,fiber,sugar and sodium. As the user clicks the calculate nutrition ,the model will generate a structured nutritional data also it will parse and display a table which shows the clean table format .Here the app is more practical where it combines the recipe generation with the basic dietary analysis. Hence this enhancement will transform the chatbot from a simple recipe generator into more intelligent and health aware cooking assistant. This will be really helful feature to learn about the different nutrients present in the food we eat.Below the code that has been added as the new feature.

```
DIET_OPTIONS = [
    "None",
    "Vegetarian",
    "Vegan",
    "Gluten-free",
    "Dairy-free",
    "Low-carb",
    "High-protein",
]
```

```
def estimate_nutrition_llm(recipe_text: str, servings: int):
    prompt = (
        "You are a nutrition estimator.\n"
        "Estimate APPROXIMATE nutrition PER SERVING from the recipe.\n"
        "Return EXACTLY 7 lines:\n"
        "Calories (kcal): <number>\n"
        "Carbs (g): <number>\n"
        "Protein (g): <number>\n"
        "Fat (g): <number>\n"
        "Fiber (g): <number>\n"
        "Sugar (g): <number>\n"
        "Sodium (mg): <number>\n\n"
        f"Servings: {servings}\n"
        f"Recipe:\n{recipe_text}\n\n"
        "Nutrition per serving:\n"
    )
    inputs = tokenizer(prompt, return_tensors="pt")
    with torch.no_grad():
        outputs = model.generate(
            input_ids=inputs["input_ids"],
            attention_mask=inputs.get("attention_mask", None),
            max_new_tokens=160,
            do_sample=True,
            temperature=0.2,
            top_k=50,
            top_p=0.95,
            repetition_penalty=1.2,
            pad_token_id=tokenizer.eos_token_id,
        )
    decoded = tokenizer.decode(outputs[0], skip_special_tokens=True)
    marker = "Nutrition per serving:"
    if marker in decoded:
        decoded = decoded.split(marker, 1)[-1]
    def parse_line(label, text):
        m = re.search(
```

```
        rf"^{re.escape(label)}\s*:\s*([0-9]+(?:\.[0-9]+)?)",
        text,
        flags=re.IGNORECASE | re.MULTILINE
    )
    return float(m.group(1)) if m else None
nutrition = {
    "calories_kcal": parse_line("Calories (kcal)", decoded),
    "carbs_g": parse_line("Carbs (g)", decoded),
    "protein_g": parse_line("Protein (g)", decoded),
    "fat_g": parse_line("Fat (g)", decoded),
    "fiber_g": parse_line("Fiber (g)", decoded),
    "sugar_g": parse_line("Sugar (g)", decoded),
    "sodium_mg": parse_line("Sodium (mg)", decoded),
}
if all(v is not None for v in nutrition.values()):
    return nutrition, None
return None, decoded
```