Develop a creative recipe-generating LLM model based on user ingredients

TAEJOON LEE, University of Ottawa, Canada
KHALID JUHI, University of Ottawa, Canada

We implemented a recipe recommendation service that leverages large-scale language models (LLMs) to generate creative recipes based on user-input ingredients. We fine-tuned our Llama model using curated datasets incorporating low-rank adaptation (LoRA) to optimize learning efficiency and memory usage. The model not only determines if a user is asking for a recipe and generates it but also enhances user interaction by allowing users to search for recipes on Google and YouTube. Quantitative evaluation using BLEU and ROUGE metrics validated the model's performance, and qualitative user surveys showed that it generates recipes that satisfy users. Notably, the model also showed that it can generate recipes from untrained data, which was the goal of the service: to generate creative cooking methods. However, our implementation of the service was experimental, and we identified issues such as limitations in the dataset, biased user ratings, and variability in the specificity of user input. The recipe generator we created highlighted the importance of user-centered design in recipe generation models and identified possibilities for future improvements, including support for multi-modal input and personalized recommendations based on user expertise. In the future, by addressing the limitations described here, the proposed service can be further improved to provide a wider variety of recipes and enhance the user experience.

Additional Key Words and Phrases: Recipe generation, large language model, Ollama

1 Introduction

With the sheer growth in data available on the internet in various areas, the world of culinary science has also evolved. This, along with the advancement of natural language processing(NLP), Recipe generation is carving its niche.

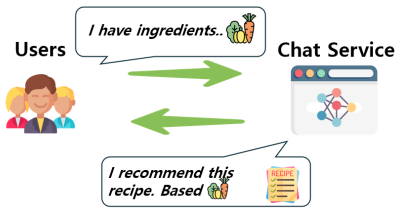


Fig. 1. The overview of recipe generator

Large language models(LLMs), a milestone in NLP, have assisted the rise of recipe generation. Transformer-based models such as OpenAI's GPT [16] and Google's BERT [1] have shown promising results in understanding and generating human language. These models can be developed to handle images or text as input to produce well-constructed instructions for cooking. The recipes can be adapted to user preference according to the dietary preferences/restrictions and ingredient availability. Available large datasets enable the LLMs to fine-tune cooking instructions based on ingredient lists and culinary techniques, assisting them in understanding the nuances of food preparation. LLMs can help chefs, home cooks, and food enthusiasts improve and reform their kitchens.

Authors' Contact Information: Taejoon Lee, tjlee096@uottawa.ca, University of Ottawa, Ottawa, Ontario, Canada; Khalid Juhi, jkhal027@uottawa.ca, University of Ottawa, Ottawa, Ontario, Canada.

For the on-device interface, Ollama [12] has emerged as a good option for LLM, ensuring fast, efficient responses without much usage costs. It supports different models with different parameter sizes, which can be fine-tuned according to the application. Here, we introduce a recipe generator based on Ollama to suggest recipes based on the ingredients the user enters.

This report is organized as follows. Section 2 discusses related work in the field of large-scale language models (LLMs) for recipes, while Section 3 describes our methodology, including dataset preparation, model fine-tuning, and backend integration. Section 4 evaluates the performance of our fine-tuned model with quantitative and qualitative evaluation, and Section 5 discusses the results, highlighting key observations and user feedback. Section 6 explores the limitations of our current approach and suggests areas for improvement, and Section 7 outlines future developments.

2 Related Work

Large language modes have increasingly become prominent for natural language processing tasks, outperforming recurrent neural networks. They can be finetuned for specific applications of text generation, and one of these is recipe generation. This is assisted by the introduction of large-scale food datasets such as Foood-101, RecipeIM, etc. It can be seen that Open AI's GPT is a fine contender for such applications.

RecipeGPT [8] is a model built on pre-trained transformer GPT-2, finetuned on Recipe1M dataset. It can generate cooking instructions based on the ingredients and recipe title and ingredients based on the recipe title and cooking instructions. It provides a comparison between machine-generated and human-written recipes. Human evaluation allows users to comment/ rate the recipes, which further assists in evaluating the recipes generated. Opposed to GPT, RecipeGM [17] uses the hierarchical generation proposed in [2] , along with sequence-to-sequence (seq2seq) modeling for recipe generation from input ingredients. Recipe1M+ is the dataset used for training . Compared to RecipeGPT, Recipe GPT remains the bigger model with better performance, except at n-gram repetition.

Long-short-term memory (LSTM) can be designed to model sequences of speech or text for natural language processing. Character level LSTM model, word-level LSTM model, and transformer-based pre-trained GPT-2 model are compared in Ratatouille [3] to find that the GPT model performs better with the higher BLUE score. Routing Enforced Generative Model(RGM) [22] has routing and generation modules. The ingredients are clustered into different dish categories. Then, the recipe is generated depending on the selected category. LSTM encoder is used to calculate the representation of the input ingredients. BLEU-4 and human evaluation indicate good performance.

In inverse cooking [18], the instruction decoder attempts to generate cooking instructions based on two input embeddings. One represents the visual features extracted, and the other represents the ingredients extracted from the image. FoodLMM [9] is yet another versatile model with multi-modal capabilities. It is based on LISA, which enables the model to classify food, recognize ingredients, generate recipes, segment, and estimate food. Llava-Chef [20] along with Vicuna(base model LaMA), integrates Contrastive Language-Image Pre-training(CLIP) used as the visual encoder. This multi-modal model is trained to predict cooking instructions from images, titles, and ingredients. The final model, Llava-Chef3, shows good promise, except regarding food substitutions. On similar grounds, Smart cuisine [7] uses OpenAI's GPT-3 API to communicate with the user, who inputs an image from which the ingredients are detected via computer vision, and AI generates recipes. It personalizes the recipe recommendation by adding ingredients and instructions to the existing dataset recipes in the Recipe1M dataset.

In our project, we decided to use Ollama, unlike the work mentioned above, which use larger models like GPT. When it comes to small-scale execution, Ollama has its strengths, such as its focus on local deployment. This reduces the latency, enhances data privacy, and has no ongoing API cost.

Model	Model based on:	Dataset	Evaluating metric
RecipeGPT	GPT-2	Recipe1M	BLEU,ROUGE,NTED
RecipeGM	Seq2seq	Recipe1M	MLUP,Jaccard,NR
RecipeMC	GPT-2	RecipeNLG	BLUE, ROUGE
RGM	LSTM	Allrecipes, Yummly	BLEU
LLaVA-Chef	Vicuna	Recipe1M	BLEU, ROUGE
Ratatouille	LSTM and GPT-2	RecipeDB	BLEU

Table 1. Comparison of some prominent models

Different models are available providing up to 405B parameter size. Also, we were able to generate structured outputs (e.g., JSON) optimized for integration with external systems like YouTube or Google, providing the user with an alternative to the generated recipe, Open Web UI is used for real-time streaming responses.

3 Methods and Methodology

In this section, we describe our approach and how we fine-tuned the LLAMA language model to develop the service we wanted to create. The description covers each step from preparing the model, to deploying the service, to how we evaluated the model.

3.1 Dataset

For this project, we planned to use two datasets 7000+ International Cuisine [14] and Recipe Dataset (2M+) Food [19].

Comparison Item	Dataset 1	Dataset 2
Dataset Name	7000+ International Cuisine	Recipe Dataset (over 2M) Food
Number of Entries	6,704	2,231,142
Number of Columns	11	7
Key Columns	name, description, cuisine, course, diet, ingredients name, ingredients quantity, prep time, cook time, instructions, image URL	title, ingredients, directions, link, source, ingredients name without quantity(NER), site

Table 2. Comparison of Two Recipe Datasets

The former is suitable for use during the model development and testing phase. At the same time, the latter is a larger dataset that allows the model to provide a wider variety of recipes. Both datasets contain a wide variety of well-documented recipes and cooking processes, making them ideal for combining with large-scale language models (LLMs). However, in this project, we only used the 7000+ International Cuisine dataset [14] for the following reasons:

(1) **Limitations of computing resources:**
The computing environment we utilized consisted of a desktop with a GPU and Google Colab (Table 4). However, even with optimized training parameters, we estimated that training on over 2 million data points would take at least 800 hours.

(2) **Ease of model experimentation and performance evaluation:**
Using the 7000+ International Cuisine dataset significantly reduced the training time to less than 4 hours, enabling multiple experiments. This short training time allowed us to

thoroughly evaluate the model's performance and ensure its readiness for deployment as a user-facing service.

3.2 Dataset Preparation

To fine-tune the model for our purposes, we used the publicly available dataset "7000+ International Cuisines" [14] consisting of columns such as recipe name, course, ingredient name, and ingredient quantity, which we preprocessed into two columns: instructions and the output to train the Llama model. Examples of datasets used to train the model are shown in Table 3. The preprocessing was done by batching the Instruction column with "I have the following ingredients: ingredients name", and the output column merged the rest of the columns into a single text.

Instruction	I have the following ingredients: Curd (Dahi / Yogurt), Ripe Bananas, Anardana Powder (Pomegranate Seed Powder), Red Chilli powder, Sugar, Cumin powder (Jeera), Dijon Mustard, Salt. Please provide recipes I can make with them.
Output	{""is_recipe_request"": true, ""recipe_name"": ""Kela Anar Raita Recipe (Banana Pomegranate Yogurt Dip Flavoured With Mustard)""} **Recipe Name:** Kela Anar Raita Recipe (Banana Pomegranate Yogurt Dip Flavoured With Mustard) **Ingredients:** 1 Curd (Dahi / Yogurt): 1 cup 2 Ripe Bananas: 1 cup, ripe, diced 3 Anardana Powder (Pomegranate Seed Powder): 1/4 cup 4 Red Chilli powder: 1/4 teaspoon 5 Sugar: 1 teaspoon 6 Cumin powder (Jeera): 1/2 teaspoon, roasted 7 Dijon Mustard: 1 teaspoon 8 Salt: to taste **Instructions:** 1 Whisk yogurt, salt, red chilli powder, sugar, dijon mustard, and cumin powder until smooth 2 Stir in diced bananas and pomegranate 3 Refrigerate for two hours before serving 4 Serve with Himachali Chana Madra and Steamed Rice

Table 3. Example of Preprocessed Dataset

To use the dataset in the model training code, SFTrainer, we loaded it with the `load_dataset()` function from the `datasets` library, and used ShareGPT to extend the simple prompt-response pair to a longer one, which allowed us to make the model's response to the user's request more interactive. The chat template was formatted as a user request (user input), the system's role definition, and the assistant's response. The template allowed for structured language model training. An example of the template we used is shown in Listing 1.

Listing 1. Chat Template

```
### Instruction:
{INPUT}

### Response:
{OUTPUT}
```

3.3 Model Preparation and Fine-Tuning

To create an LLM model that could provide interesting recipes for users, we decided to fine-tune an LLM model that had already been trained at scale rather than train a model from scratch. To do

Specification	Desktop	Google Colab
GPU	NVIDIA GeForce RTX 3080	Tesla T4
GPU Memory(VRAM)	10GB	15GB
Memory	32GB	12.7GB
CPU	AMD Ryzen 5 5600X 3.7Ghz	Intel Xeon 2.0Ghz
Operating System	Ubuntu 22.04	Ubuntu 20.04

Table 4. Desktop and Google Colab Specification

this, we considered the system specifications provided in table 4 and used Llama 3.2 (3B), Llama 3 (8B) [11] and Tinyllama (1.1B) [23] to reduce training time and lower GPU memory utilization. However, through repeated experiments, Tinyllama (1.1B) performed poorly and was excluded from the model for the final service.

Comparison Metric	TinyLlama	Llama 3.2 (3B)	Llama 3 (8B)
Model Size (Parameters)	1.1 billion	3 billion	8 billion
Context Length	Up to 2K	Up to 130K	Up to 8K
Pre-trained Data	Open-source data	Diverse domains	Diverse domains
VRAM for Training	Approx. 4GB	Approx. 8GB	Approx. 16GB
Inference Speed	Fast	Moderate	Slow

Table 5. Comparison of TinyLlama, Llama 3.2 (3B), and Llama 3 (8B).

The code for fine-tuning is mainly based on [5]. **SFTrainer** is a framework implemented to simplify the task of fine-tuning models using supervised learning by handling tasks such as data loading, tokenization, and training parameter management. Through several training experiments, we chose appropriate parameters, such as a batch size of 1 and a training period of 5 epochs, to ensure that the model was effective when learning the dataset we prepared. We chose hyperparameters such as a learning rate 1e-5, AdamW 8-bit optimization, and a linear learning rate scheduler to balance learning efficiency and stability. We also chose mixed-precision training, utilizing FP16 for semi-precision floating-point operations and BF16 for a more comprehensive numerical range. For the GPUs we had available, the NVIDIA GeForce RTX 3080 supports FP16, and the Tesla T4 from Google Colab supports both FP16 and BF16, so we set it to apply FP16 or BF16 depending on the GPU model.

Listing 2. Training Script for Fine-Tuning

```
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 11,
    packing = False,
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 3,
        num_train_epochs = 5,
        learning_rate = 1e-5,
```

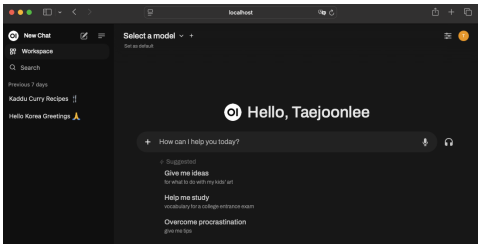
```
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.1,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
    ),
)
```

For efficient training, we utilized low-rank adaptation (LoRA) [6], a technique designed to insert learnable weights into the transformer layer in a memory-efficient manner. LoRA compresses the large weight matrix of the transformer into a smaller, low-rank matrix. This method can significantly reduce the number of parameters that need to be updated during training, allowing us to use large batch sizes without increasing memory usage. This results in faster training and improved training efficiency.

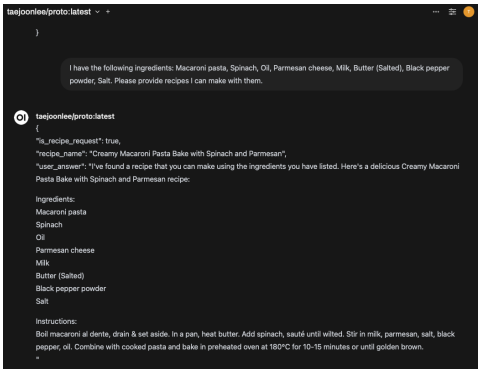
In the case of gradient checkpointing, we used the recently developed Unsloth [4] to save intermediate steps during training so that they can be recomputed during backpropagation without storing them in memory. This method allowed us to reduce GPU memory utilization by about 50%. Using LoRA and Unsloth to optimize memory usage and computation, we significantly improved model performance using less memory.

3.4 Backend Integration and Deployment

To integrate our fine-tuned LLAMA model into a user interface web service, we used the Open Web UI [21] provided by Ollama [12]. We chose this web service because it supports custom models and is easy to deploy.



(a) Open Web UI Main Page



(b) Open Web Ui Chat Page

Fig. 2. Open Web UI Usage

The frontend of Open Web UI is similar to the popular ChatGPT website, so we decided that no modifications were needed for our project. The backend is implemented as a FastAPI. In this FastAPI, we will implement a service to identify if the model's answer is a recipe and connect with external services.

3.5 Recipe Response Logic

The model we want to create in this project has two functions: to recommend recipes based on the ingredients entered by the user, and to link those recipes with external services (e.g., YouTube, Google) to show search results such as videos or webpages.

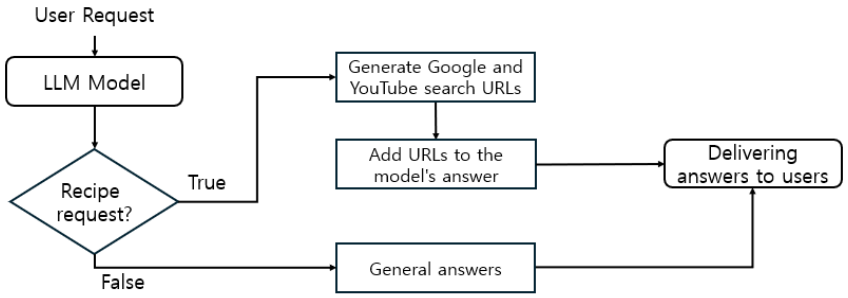


Fig. 3. The architecture of service flow

The model can be trained on a prepared dataset to generate answers for recipe recommendations. However, for external service search, the problem is that the model needs to determine if it is recommending a recipe and, if so, extract only the name of the recipe. To solve this problem, we used a prompt pattern that forces the model to generate a structured response (e.g., in JSON format), which we want to consist of fields like `is_recipe_request`, `recipe_name`.

We want to make this structured output easy for our backend to parse and process, and here is what we expect to happen: when the model provides output, our backend service will check if `is_recipe_request` is True before responding to the user, and will use `recipe_name` to search YouTube or Google to add the result to the user response, to show the user(see Figure 3).

The Open Web UI used FastAPI to associate responses generated by models downloaded from Ollama with the Web UI. So, we need to parse the JSON and shape the answers generated by our fine-tuned model into the answers we want before delivering them to the user via the Web UI. However, the process by which we deliver the answer to the user should remain a tokenized streaming technique that allows the user to see the answer in real-time. The Open Web UI can preserve this technique if implemented without breaking the asynchronous HTTP structure it uses.

We identified an asynchronous function, **post_streaming_url**, that provides the answer to the user in the Open Web UI. The **post_streaming_url** function starts by sending a POST request to the model's endpoint with a JSON payload containing the user's input. The model's response is streamed back to the backend line by line, and each line is decoded and inspected for JSON structure.

The JSON structure detects opening and closing curly braces ({ and }) and attempts to identify JSON objects within the answers that our trained model outputs. Once the JSON is fully reconstructed, we parse it to extract key fields such as `is_recipe_request` and `recipe_name`.

For instance, the core part of the function processes the JSON as follows:

Listing 3. Processing JSON Block from Streamed Response

```
async def post_streaming_url(
    url: str, payload: Union[str, bytes], stream: bool = True, content_type=None
):
    # Skip initialization..
    async for line in r.content:
```

```
        decoded_line = line.decode("utf-8")
        line_json = json.loads(decoded_line)
        content = line_json.get("message", {}).get("content", "")
        if "{" in content:
            sg_is_json_object = True
            sg_json_buffer = ""
        if sg_is_json_object:
                        sg_json_buffer += content
                        line_json['message']['content'] = ""
                        line = (json.dumps(line_json)+"\n").encode('utf-8')
                        if '}' in content:
                            sg_is_json_object = False
                            try:
                                json_response = json.loads(sg_json_buffer)
                                sg_is_json_object = False
                            except json.JSONDecodeError as e:
                                print("Error decoding JSON:", e)
                            is_recipe_request = json_response.get('
                                is_recipe_request', None)
                            recipe_name = json_response.get('recipe_name', None)
                            print("DEBUG: ", is_recipe_request, recipe_name)
                            encoded_recipe_name = urllib.parse.quote(recipe_name)
```

If the `is_recipe_request` field is set to True, generate URLs that search Google, YouTube, etc., for the recipe name. These URLs will contain results on making the recipe the model recommends. Here is how we generated the URLs.

Listing 4. Appending Search Results to the Response

```
line_json["message"]["content"] = f"**Search Results** \n"
line_json["message"]["content"] += f"- Google: https://www.google.com/search?q={
    recipe_name} \n"
line_json["message"]["content"] += f"- YouTube: https://www.youtube.com/results?
    search_query={recipe_name} \n"
```

4 Experiments

4.1 Quantitative Evaluation

BLEU [13] score is an evaluation metric originally developed for machine translations, which is now also used for other natural language processing tasks. It measures n-grams (word sequences) overlap between generated and reference texts. There can be different scores. i.e., BLEU-1, BLEU-2, BLEU-3, BLEU-4, depending on the n-gram level. The scores can vary from 0 to 1, where 1 denotes perfect overlap between the generated and reference text. The quality of the score depends on the number of references used as well as the length of the generated/reference sentences. For creative text generation, the scores penalize the text for rewordings or synonyms. Therefore, BLEU scores alone cannot be used as the evaluation metric.

Rouge [10] is an evaluation metric primarily used for summaries. The generated text is compared with the reference summaries, and it primarily measures the recall, as opposed to precision for BLEU. With multiple references, the pairwise scores between each reference and generated text are calculated and the maximum is taken as the final score. For creative text generation, such as recipe generation, ROUGE can confirm the coverage of ingredients without penalizing differences in phrasing or rewording. in our case, it computes ROUGE-1 (unigram overlap), ROUGE-2 (bigram overlap), and ROUGE-L (longest common subsequence). They compute the recall, precision, and F1 score. The scores can vary from 0 to 1, similar to BLEU.

To get a valid idea of the performance of our model, we need to find the BLEU and ROUGE scores for around 100 recipes with multiple references. This requires a database of multiple references for each recipe and computing the scores for each. Due to the limitation of time and resources, we rely more on human evaluation. An example of a single recipe is shown. 4 reference recipes (available online) are used as references.

Recipe name	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
Carrot Kheer	0.529	0.287	0.151	0.075	0.467	0.133	0.261

Table 6. BLUE and ROUGE scores for a single generated recipe

4.2 Qualitative Evaluation

BLEU and ROUGE are not good enough indicators for user experience and human evaluation is imperative for our model. The recipe should not only be comprehensive but also preferable to human consumption. So, we have collected survey responses from 10 people on their experience in using our model. The survey includes questions aligned with the QoE criteria of effectiveness, efficiency, satisfaction, enjoyment, and social presence, as detailed in Table 7.

Question	QoE Criterion
Q1. How easy is it to follow the instructions in the generated recipes?	Effectiveness
Q2. Was all the ingredients entered included in the recipe?	Effectiveness
Q3. How do the generated recipes compare to human-written recipes?	Effectiveness
Q4. Did it save you time in finding a recipe?	Efficiency
Q5. Was the recommended cooking method appropriate?	Effectiveness
Q6. Were you satisfied with the dish made using the recommended method?	Satisfaction
Q7. Did you enjoy using the recipe generator?	Enjoyment
Q8. How likely are you to use this recipe generator for cooking?	Social Presence

Table 7. Survey Questions Categorized by QoE Criteria

The user experience was evaluated using five QoE criteria:

(1) **Effectiveness:** 90% of users said the generated recipe instructions were easy to follow and included all the ingredients they entered. Furthermore, 60% rated the recipes as "better" or "much better" than those written by humans, confirming that our recipe generator can generate practical recipes.
(2) **Efficiency:** All users said the recipe generator saved time looking for recipes, confirming its role as a practical and time-saving solution.
(3) **Satisfaction:** All users were either "satisfied" or "very satisfied" with the results.
(4) **Enjoyment:** 80% of users were "satisfied" or "very satisfied" with the results.
(5) **Social presence:** 80% of respondents said they were "very likely" or "likely" to use the recipe generator again, demonstrating its relevance and acceptance, showing that it can be perceived as a meaningful tool in the user's cooking.

Users performed well across all criteria. However, the small sample size of 10 people may have biased the results.

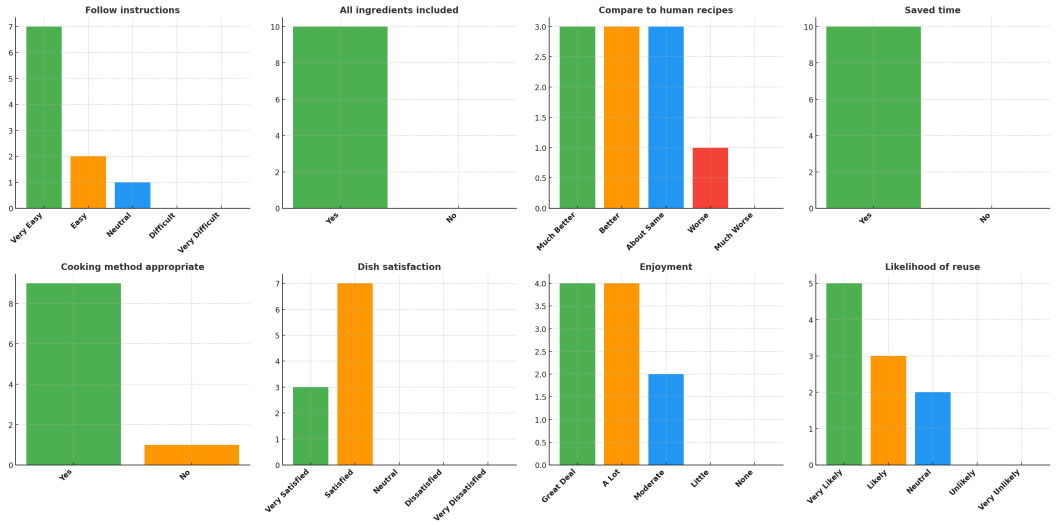


Fig. 4. Survey results

5 Results

This section demonstrates the functionality and performance of the LLama 3 (8B) model integrated into the Open Web UI. Generate recipes based on user requests and verify that the answers are streamed.

5.1 User Interaction with the Web UI

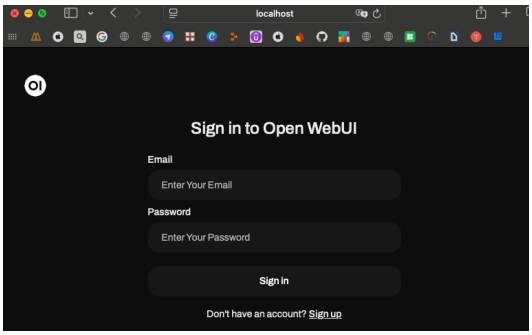


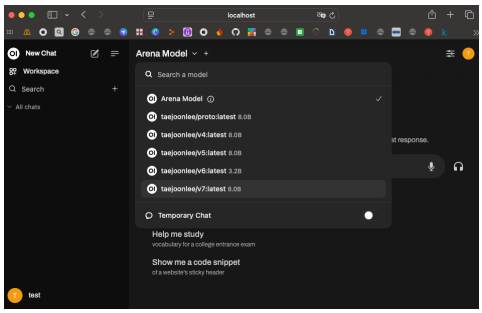
Fig. 5. Login Page

Users must connect to the Open Web UI server via a web browser to get recipe recommendations. To use the service, users need to log in with their user account. After successful login, they will see the home screen.

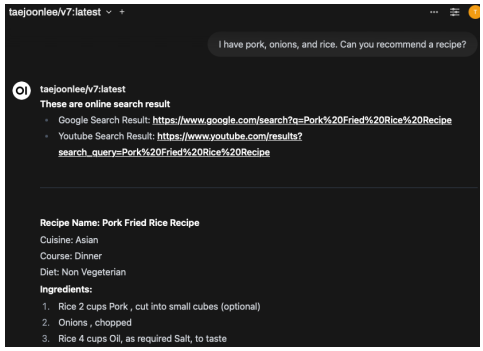
Users can choose any of the models installed on the Open Web UI server, and for this demonstration, we chose the LLama 3 (8B) model as the v7 model fine-tuned with our dataset. Once the user has selected a model, they will be asked to write a phrase to recommend a recipe based on the ingredients they have on hand. Here is an example `I have pork, onions, and rice. Can you recommend a recipe`?

5.2 Model Response Streaming

When the user types a question, the web application provides the input to the LLM model the user selects. The LLM model starts generating a response in a streaming format. This streaming feature increases interactivity and reduces latency by not having the user wait for the entire response to be generated.



(a) Model Select



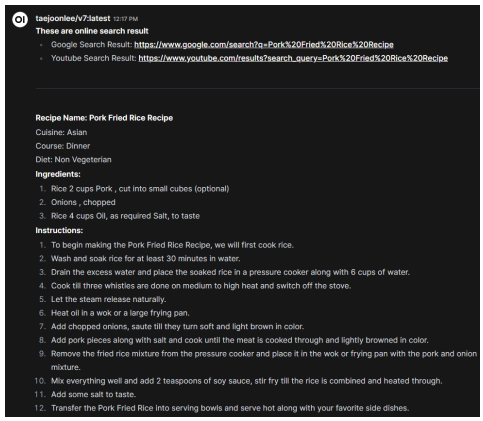
(b) Enter user questions

Fig. 6. Usage screen

Before the backend delivers the model's response to the user, it checks for the presence of a JSON structure and attempts to parse it to integrate external search results. In the logs, JSON objects like `is_recipe_request` and `recipe_name` were successfully detected and reconstructed in the model's output. This JSON parsing process works consistently with the implementation described in Section 4.3 and shows that the integration between the model and the Web UI is stable.

5.3 Analysis of the Model's Response

(a) Backend Logs: Attempt to parse JSON



(b) Model response results

Fig. 7. Results screens (backend logs and user screens)

We can see that the model's response followed the format of the Output structure of the dataset we defined. An additional point to note is that while the structure followed the dataset, the recipes generated were not trained on the dataset. The results show that we can generate creative but realistic recipes, which is what we were going for.

6 Discussion

To analyze the QoE of our model, we performed quantitative and qualitative evaluations in Section 4. In the quantitative experiments, we evaluated the model's performance through BLEU and ROUGE scores, which are important measures of how similar the generated text is to the reference. However, these scores can have limitations in assessing the concept of creativity. They may need to fully reflect the diversity and realism of the results, especially in domains such as recipes. This is because, in the case of recipes, this knowledge category's limits are not always evident in practice, and satisfaction is a subjective judgment. In the qualitative experiment, we used a user survey to evaluate the effectiveness, satisfaction, and overall user experience of the service provided by the model. While most survey respondents were satisfied with the service, we cannot rule out the possibility that the results are biased because the survey was conducted among ten acquaintances. Therefore, further evaluation with a broader and more diverse group of users is warranted.

We also found that the user's input significantly impacted the quality of the model's responses. During the survey, some users entered the ingredients of a dish, such as "eggs, salad," which caused the model to fail to understand the user's needs and generate out-of-context responses. These examples suggest that the quality of the LLM model's responses can vary depending on the user's clarity about what they want to know and how they want to ask the question. For example, a question like "What dish can I make using eggs and salad?" can significantly improve the quality of the model's response compared to simply typing "eggs, salad." Providing pre-defined question examples, such as the popular service Perplexity [15], is a good way to help users interact with models more effectively.

Finally, some respondents needed help judging the recipes' quality during the survey. Users must gain essential knowledge to understand the cooking process, even when referring to commercially available cookbooks. This experience made us realize that LLM needs to know the user's status or knowledge level when answering their questions. Generating adaptive responses tailored to the user's level would be an important factor in increasing the service's effectiveness and improving the user experience.

Based on these discussions, further research and development are needed to improve the model's performance and strengthen its user-centered approach.

7 Conclusion

We implemented an LLM-based service that can recommend creative and practical recipes based on user-input ingredients. We fine-tuned the Llama 3 (8B) model and designed it to generate JSON-structured responses to provide recipe recommendations by connecting with external services such as Google and YouTube. We also leveraged the Open Web UI platform to implement responses in real-time streaming format, enabling interactive interaction with users.

The experimental results confirmed the model's performance through BLEU and ROUGE scores, and a qualitative user survey assessed the service's overall satisfaction. In particular, we found that the model provided creative and realistic recipes while producing significant results on untrained data.

Further developments could consider the user's level of expertise and be extended to support multi-modal input (e.g., photo-based ingredient recognition).

8 Limitation

While this study yielded meaningful results in developing a recipe recommendation model, it has several limitations. Firstly, the "7000+ International Cuisine" [14] dataset used for training contains a limited number of recipes, around 6500, excluding those that contain errors or are written in a foreign language other than English. It is also likely that the recipes in the dataset are biased towards the culture of the contributor. The dataset's unbalanced nature limits its use, as a model trained with it may not fully reflect the diversity and complexity of recipes.

While we used quantitative metrics such as BLEU and ROUGE to evaluate the performance of our models, another limitation is that these scores need to reflect quality in creative text generation fully. In addition, the qualitative evaluation was conducted on a small group of acquaintances, potentially biased the results and making them unreliable.

The format of the user input also determined the quality of the model. The model would likely generate out-of-context responses if the user's question were not specific and unambiguous. If the quality of user questions is low, it could degrade the quality of service in a large user environment, so something needs to be done to help users clarify their questions.

Finally, more testing is needed to reflect different user environments (e.g., differences in cooking utensils or cooking levels), and further research is needed to consider the impact of technical limitations or network performance on the streaming response process. Based on these limitations, future iterations of the service will utilize datasets and evaluation methods with more sophisticated and diverse recipes to improve the generalisability of the model and the user experience.

9 Future Works

In the future, the model can be developed to track eating habits, such as calorie and sugar intake. It could also be trained to make ingredient substitutions for tailored recipes according to food restrictions. Additionally, the model could be developed to include multimodal capabilities, such as taking pictures as inputs to recognize ingredients and suggest recipes or including a picture of the dish along with the generated recipe.

10 Acknowledgement

We are deeply grateful to Prof. Abdulmotaleb El Saddik for his constant support and guidance throughout the course, which gave us the knowledge and opportunity to carry out the project. We express our gratitude to the TA, Mohammed Faisal, for his valuable feedback and suggestions while developing our model and being extremely helpful with all aspects of the course. We would also like to acknowledge the University of Ottawa for the support and resources required by the course and, by extension, the project. We especially thank our peers for being kind enough to try our recipe generator and provide helpful feedback to evaluate and improve our model. Their constructive criticism was indispensable. Finally, we thank our family and friends for their unwavering motivation and support throughout our academic journey.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[2] Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical Neural Story Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, Melbourne, Australia, 889–898. https://doi.org/10.18653/v1/P18-1082

[3] Mansi Goel, Pallab Chakraborty, Vijay Ponnaganti, Minnet Khan, Sritanaya Tatipamala, Aakanksha Saini, and Ganesh Bagler. 2022. Ratatouille: A tool for Novel Recipe Generation. In *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*. 107–110. https://doi.org/10.1109/ICDEW55742.2022.00022

[4] Daniel Han-Chen. 2024. *Github - Unsloth.* Retrieved Oct 9, 2024 from https://github.com/unslothai/unsloth/tree/main

[5] Daniel Han-Chen. 2024. *Ollama + Unsloth + Llama-3 + Alpaca.ipynb.* Retrieved Oct 9, 2024 from https://colab.research.google.com/drive/1WZDi7APtQ9VsvOrQSSC5DDtxq159j8iZ?usp=sharing#scrollTo=r2v_X2fA0Df5

[6] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv. Retrieved October 9, 2024 from https://arxiv.org/abs/2106.09685 cs.CL.

[7] Ponrawin Kansaksiri, Pongpipat Panomkhet, and Natthanet Tantisuwichwong. 2023. Smart Cuisine: Generative recipe & ChatGPT powered nutrition assistance for sustainable cooking. *Procedia Computer Science* 225 (2023), 2028–2036. https://doi.org/10.1016/j.procs.2023.10.193 27th International Conference on Knowledge Based and Intelligent Information and Engineering Sytems (KES 2023).

[8] Helena H. Lee, Shu Ke, Palakorn Achananuparp, Philips Kokoh Prasetyo, Yue Liu, Ee-Peng Lim, and Lav R. Varshney. 2020. RecipeGPT: Generative Pre-training Based Cooking Recipe Generation and Evaluation System. *Companion Proceedings of the Web Conference 2020* (2020). https://api.semanticscholar.org/CorpusID:212415056

[9] Xiuyu Li, Lu Yu, Zheng Zhang, Xin Eric Wang, Mingyang Tang, Han Xue, Yanan Luo, Siyuan Wu, Xiangnan He, and Xiangyang Xue. 2023. FoodLMM: A Versatile Food Assistant using Large Multi-modal Model. *arXiv preprint arXiv:2308.08197* (2023). https://arxiv.org/abs/2308.08197

[10] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[11] Meta. 2024. *Llama 3.2 Model Cards & Prompt formats.* Retrieved Oct 9, 2024 from https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2

[12] Ollama. 2023. Ollama: AI Model Serving Platform. https://ollama.com/ Accessed: 2024-10-10.

[13] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Pierre Isabelle, Eugene Charniak, and Dekang Lin (Eds.). Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[14] Rumit Pathare. 2024. *Kaggle - 7000+ International Cuisine.* Retrieved Oct 9, 2024 from https://www.kaggle.com/datasets/rumitpathare/indian-recipes

[15] Perplexity AI. 2024. *Perplexity - AI-Powered Search Engine.* Retrieved November 23, 2024 from https://www.perplexity.ai/

[16] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019). https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[17] Anja Reusch, Alexander Weber, Maik Thiele, and Wolfgang Lehner. 2021. RecipeGM: A Hierarchical Recipe Generation Model. In *2021 IEEE 37th International Conference on Data Engineering Workshops (ICDEW)*. 24–29. https://doi.org/10.1109/ICDEW53142.2021.00012

[18] Amaia Salvador, Michal Drozdzal, Xavier Giro-i Nieto, and Adriana Romero. 2019. Inverse Cooking: Recipe Generation From Food Images. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10445–10454. https://doi.org/10.1109/CVPR.2019.01070

[19] Wilmer Arlt Strömberg. 2024. *Kaggle - Recipe Dataset (over 2M) Food.* Retrieved Oct 9, 2024 from https://www.kaggle.com/datasets/wilmerarltstrmberg/recipe-dataset-over-2m

[20] Mohbat Tharani and Mohammed Zaki. 2024. LLaVA-Chef: A Multi-modal Generative Model for Food Recipes. https://doi.org/10.48550/arXiv.2408.16889

[21] Open Web UI. 2024. *Open Web UI.* Retrieved Oct 9, 2024 from https://openwebui.com/

[22] Zhiwei Yu, Hongyu Zang, and Xiaojun Wan. 2020. Routing Enforced Generative Model for Recipe Generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 3797–3806. https://doi.org/10.18653/v1/2020.emnlp-main.311

[23] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An Open-Source Small Language Model. arXiv. Retrieved October 9, 2024 from https://arxiv.org/abs/2401.02385 cs.CL.