WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: COMPUTER SCIENCE IN ENGLISH

BACHELOR THESIS

SUPERVISOR:
Conf. Dr. Cosmin Bonchis

GRADUATE:
Codrin Rață

TIMIŞOARA
2025

WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
BACHELOR STUDY PROGRAM: COMPUTER SCIENCE IN ENGLISH

ReciPal - Integration of LLM in conversational cooking recipe generation

SUPERVISOR:
Conf. Dr. Cosmin Bonchis

GRADUATE:
Codrin Rață

TIMIŞOARA
2025

Abstract

In this thesis, the author explores the cooking recipe generation using Large Language Models. The main objective is to shorten the search for a recipe that uses specific ingredients. The novelty of this thesis lies in using Meta's LLaMA 3, a large language model, into recipe generation using natural language input of ingredients. This approach aims to offer a very wide range of possible recipes to choose from. It also integrates accessibility features (high contrast, text-to-speech and microphone input).

The LLaMA 3 model has been adapted to the culinary field by fine-tuning on a specialized dataset, composed of recipes, ingredient combinations, cooking techniques and other relevant information , in order to generate coherent, relevant and high-quality recipes. This model is integrated into a friendly web interface, where users can freely enter ingredients and receive personalized recipes in return.

This paper makes a relevant contribution to the ever-expanding field of applications based on linguistic models, demonstrating how open-weight models, such as LLaMA 3, can be customized and adapted for concrete applications specific to practical domains.

Contents

Chapter 1

Introduction

1.1 Motivation and problem statement

In the last years, along with it's increase in popularity, artificial intelligence has also made significant progress in natural language understanding, opening more possibilities for building intelligent assistants that can interact with users in more human-like ways. At the same time the food and nutrition domain has gained attention. It has also gained attention in the tech industry, particularly through applications that helps users plan meals, track nutritional values of foods or discover food recipes. Despite this, many applications or services rely on predefined existing databases, rigid interfaces or predefined filters. All that can be hard to learn at first and can become limiting. The motivation behind this thesis is to explore how modern large language models, specifically a fine-tuned instance of Meta's LLaMA 3 [1], can be used to provide a conversational and flexible food recipe generation experience, allowing users to just input the ingredients they have and immediately receive food recipes that use those ingredients.

Most recipe websites today work in a standarized way. Users usually have to select filters, choose from categories, or type in specific words to find a recipe. These systems do not work well with natural, everyday language and often cannot understand when someone lists random ingredients. They also do not remember what the user said before, which makes the experience feel less personal and harder to use. This thesis looks at that problem and tries to solve it by building a system that can create full recipes from what ingredients the user inputs. The primary challenge is to combine a strong language model with a web application that enables users to have a smooth and useful conversation.

1.2 Project objectives and proposed solution

The main objective of this thesis is to design and implement a web-based application that uses a conversational user interface combined with a large language model to generate cooking recipes from natural language input. The key goals are developing the conversational interface, where the users can freely type ingredients, optimizing an instance of Meta's LLaMA 3 to work better with recipe generation, ensuring the model gives out coherent and relevant recipes and building a backend that can connect all the needed parts in order to have the full application running.

The solution developed in this thesis consists of a full-stack web application featuring a chat-like frontend and a backend service that communicates with a fine-tuned LLaMA 3.2 model. When a user enters a list of available ingredients, the frontend sends this input to the backend. The backend then constructs an appropriate prompt and sends it to the language model, which generates a complete recipe including a title, ingredients list, step-by-step instructions and other relevant details. The response is then returned to the user in a conversational format. This architecture allows for flexible, real-time interaction with the model and supports free text input, making the overall experience more natural and useful than conventional recipe apps.

1.3 Author's contribution and thesis structure

The majority of the project tasks (analysis of the problem, design of the system, implementation and testing) were done by the author. On the backend side, the author fine tuned a cooking recipes database with the Meta's LLaMA 3.2 model and implemented server-side logic for the model. The data set used to fine tune the model was obtained from the Huggingface platform and was curated, filtered and transformed to fit the desired format. At frontend a live chat-like interface was constructed using web-technologies with the focus being on the ease of use. The author also developed appropriate prompts and inspected model output to ensure the quality of generated recipes. Every detail from the beginning was thought to be scalable and easy to use, resulting in a complete application, with artificial intelligence-based features and an intuitive and user-friendly interface.

This bachelor's thesis is strucured in multiple chapters. Chapter 2 describes the formal statement of the problem, the user requirements and the main challenges highlighted. Chapter 3 reviews literature in artificial intelligence and large language models. Chapter 2 discusses the general system architecture, as well as design of the business-logic and the data model with the services used. In Chapter 5, the application features, user interaction, user experience and possible scenarios of demonstration are introduced. Summary of Contributions The results are summarized in Chapter 6, the limitations of the developed scenarios are discussed and possible improvements are suggested as future work. Finally, the paper contains a bibliography with the consulted sources and the annexes with the code snippets and the implementation details.

Chapter 2

Problem description

2.1 Formal problem definition

In this thesis, we address the task of generating cooking recipes from natural language ingredient lists provided by the user using a large language model. The goal is to create a function

$$f : \mathcal{I} \to \mathcal{R}$$

where $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ represents a set of ingredients given by the user, written in natural language, and $\mathcal{R}$ is a structured recipe representation consisting of a recipe title T, metadata M (cuisine type, dietary considerations, and preparation and cooking times), a detailed ingredients list L expanded from the user's input and a sequence of clear, actionable cooking instructions S.

This is a conditioned natural language generation problem under real-time constraints where the model is required to emit coherent, relevant and executable outputs. The recipe's ingredients need to be almost (common ingredients can be added e.g., salt, pepper, oil) the same with the provided ones, the resulting recipes need to be safe, readable and returned in an amount of time that's reliable. Further, the problem is constrained by limited conversational memory because the model should be able to return the cooking recipe in one instruction, without the need of multiple questions and answers, and be able to produce outputs that are easy to follow and safe to cook.

2.2 Proposed solution

The first solution was fine-tuning a large language mode to come up with new recipes based on user input of ingredients and integrating it into an application of web interface to make it usable by everyone. The application was designed for the users who want to make a meal out of whatever they have at home. Rather than sifting through endless recipe databases, or a confusing series of filters, you just open up a web interface and start typing in a natural language (e.g.,"chicken, garlic, rice"). The system then output a full recipe from the ingredients provided (e.g., "Garlic Chicken Rice Bowl"), exact ingredient quantities, step by step instructions and other relevant data, like the fact that it's an Asian cuisine or that it takes 10 minutes to prepare and 20 minutes to cook.

All this takes place through a chat-like web interface, offering an intuitive, accessible and familiar experience even for users without much technical knowledge. The key participants in the generation process are the user, who inputs the ingredients and chooses dietary needs and course preferences, and the system that parses the inputs, formats it and sends it to a fine tuned LLaMA 3 model that generates the recipe and sends it back to the system. Ultimately the response is again formatted by the frontend part of the system and displayed to the user.

Chapter 3

Related work

3.1 Existing cooking recipe applications and platforms

The advancements made in the field of recipe generation using artificial intelligence or large language models include recipe generation from food images[2]. Some of the primary recipe generation models include Image-to-Recipe models [3], [4] and Text-to-Recipe models [5], [6]. Regarding Image-to-Recipe models, the development of large food datasets like Food-101[7], Recipe1M[8], RecipeNLG[9] or RecipeDB[10], along with initiatives such as the iFood Challenge, has played a key role in advancing visual food recognition. The recognition of food dishes and meals, and offering people detailed information on unknown or unfamiliar foods and dishes also plays an important role in other food related applications [11], [12]. Salvador *eu al.* [4]came up with InverseCooking, a system that uses an encoder-decoder framework to predict the title of a meal from a photo. Wang *et al.* [13] also worked on an Image-to-Recipe model but treated it like an image captioning task.

Even though food recognition is already a complex task, recipe generation is even more complex. It requires the models to know more about food composition, ingreients and cooking in general in order to solve the task well.

Some methods for recipe generation struggle to output personalized recipes when users have only partial knowledge of the ingredients used in specific cooking recipes. To address this limitation, Majmuder *et al.* [14] introduce a new task: personalized recipe generation. This involves taking a dish name and an incomplete list of ingredients and generating a full recipe in natural language that reflects the user's individual preferences. Their approach incorporates both cooking technique and recipe-level information drawn from the user's previously viewed or prepared recipes. The system uses past user data and blends it smartly with the current input. This helps the model create a personalized and accurate recipe. Experiments conducted on a large dataset of 180 000 recipes and 700 000 user interactions demonstrate that the proposed model generates more plausible and user-tailored recipes compared to models without personalization.

There are also more advanced systems that can recognize the contents of a specific meal and estmate the nutritional values of the meal (e.g., calories) [15], estimate food quantities [16], predict the list of ingredients used in a meal from a photo [17].

Another commercial grade service that uses artificial intelligence in the food and cooking field is Samsung's Smart things cooking. The service is integrated in the Health app and it tracks your consumed and burned calories and uses artificial intelligence in order to recommend healthy recipes that align with user's preferences and goals. Another feature of the app is the ability to join cooking communities and share recipes with others. The service also has IoT capabilities integrated as it lets you send the recipes to your oven for added convenience.

3.2 Conversational interfaces for recipe assistance

Conversational agents and chatbots increased in popularity across various fields. That has transformed user interaction by allowing a more natural communication. In culinary applications conversational interfaces enable users to communicate with recipe systems as if they were communicating with a human assistant, and this reduces the barrier for users not familiar with structured query formats. The arrival of large language models (LLMs), such as OpenAI's GPT series and Meta's LLaMA, has significantly improved the ability of chatbots to generate and understand human-like text, making it possible to generate coherent, contextually accurate responses in real-time.

The exploration and understanding of how humans interact, communicate and engage in conversational recipe recommandation is still an important task. Barko-Sherif *et al.* [18] conducted a Wizard-of-Oz study to explore how users engage in conversational recipe recommendation tasks using a prototype agent named Telefood. Their findings emphasized the diversity in user queries, the importance of natural and flexible dialogue management, and the distinct conversational patterns between voice and text-based interactions. This research provides foundational insights for building user-friendly, voice-enabled recipe assistants, and strongly supports the design goals of this thesis, which also aims to support real-time, chat-based culinary interactions.

Chapter 4

Application architecture

4.1 System overview and technical stack

The application has three main architectural components being the frontend, a browser-based chat interface, the backend, a server that handles input processing and model communication, and the model, a fine-tuned LLaMA 3 model hosted locally. The overall architecture is illustrated in Figure 4.1.

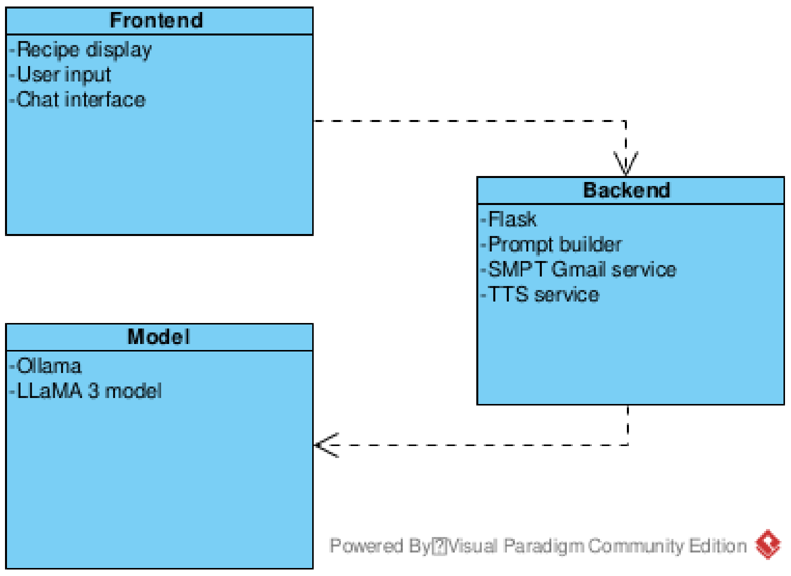


Figure 4.1: High-Level System Architecture

4.1.1 Frontend architecture

The frontend is a single-page application (SPA) built using HTML, CSS and Javascript, which provides a dynamic and responsive user interface. The main components of the frontend are the Chat UI component, that displays user inputs and model responses in a conversational manner and the user input field, that captures the user's input from keyboard or microphone. The architecture can be better seen and understood in Figure 4.2

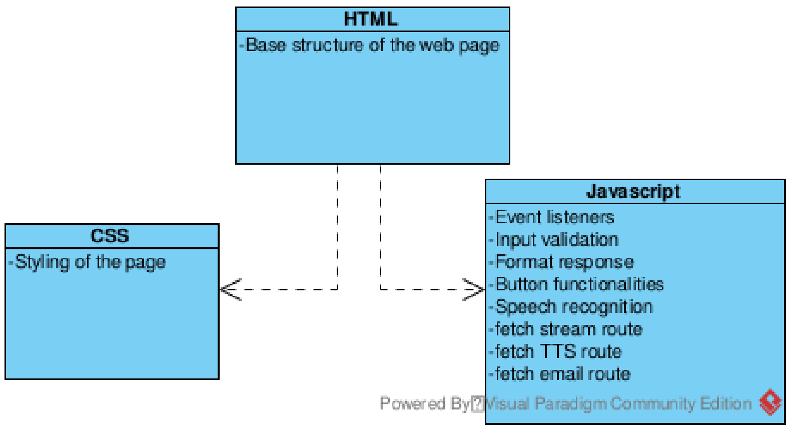


Figure 4.2: Frontend System Architecture

Others components of the frontend are additional buttons and fields such as the light and dark mode toggle, the high contrast toggle, the text-to-speech toggle and the hands-free mode toggle. Another field is located under the ingredient input field and is the email input field, where the user can input an email for all the recently generated recipes to be sent to.

The styling of the page is implemented using Bootstrap and local CSS and other functions are implemented using Javascript.

4.1.2 Backend architecture

The architecture can be better seen and understood in Figure 4.3

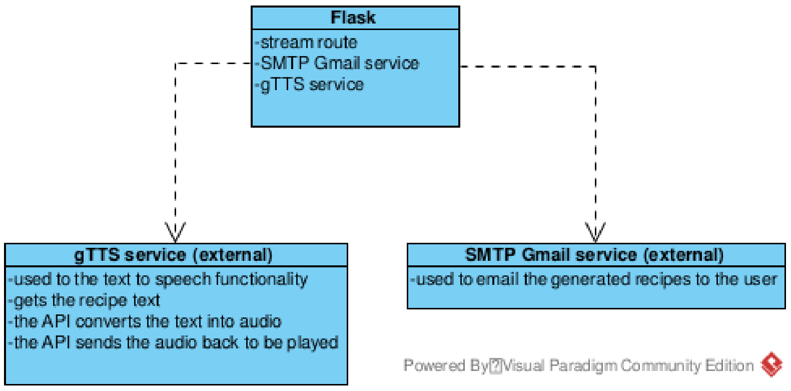

Flask
-stream route
-SMTP Gmail service
-gTTS service
gTTS service (external)
-used to the text to speech functionality
-gets the recipe text
-the API converts the text into audio
-the API sends the audio back to be played
SMTP Gmail service (external)
-used to email the generated recipes to the user
Powered By Visual Paradigm Community Edition

Figure 4.3: Backend System Architecture

The homepage route (/) is the main interface. It loads ”index.html” and acts as the landing page of the application.

The recipe generation route (/stream) is responsible for interacting with the model. It accepts a list of ingredients and filters (diet and course) from the frontend, streams the generated from the locally hosted LLaMA model. This route uses POST method and is also used to stream the model response line by line.

The email sending route (/send-recipe-email) is used to send the generated recipes via e-mail. It uses Gmail SMTP and is provided by the Flask-Mail library. It also uses Post method and, as input, it receives a JSON containing the recipient’s email, and recipe details. It returns a succes or error message.

The Text-to-Speech route (/api/speak) is used to play out loud the generated recipes through the Google TTS service. It takes the recipe content text, sends it to the gTTS API where it gets coverted into audio and sent back to be played to the user. It also uses POST method and, as input, it uses the extracted recipe content text from a JSON.

4.2 Model integration and training workflow

4.2.1 LLaMA 3.2 model integration and fine-tuning

The core of the system is a fine-tuned instance of Meta's LLaMA 3.2 3B Instruct language model. The model was fine-tuned using a curated dataset of 4000 structured cooking recipes (81 distinct cuisines).

The base of any language-based system lies in the quality of its training data. For this project, an open-access food recipe dataset available on Hugging Face was used.

The original dataset has 7101 entries, however, to ensure consistency and relevance, significant dataset processing was conducted. Cooking recipes containing non-English characters were removed and in order to reduce regional bias, the dataset was rebalanced by reducing the number of Indian and North Indian cuisine, which dominated the original collection. This adjustment ensured a more culturally diverse and regional balanced distribution of the cooking recipes. Unnecessary metadata (such as image URL and description) was also discarded. The dataset was then curated and normalized to focus on the list of input ingredients, the recipe title and step-by-step cooking instructions.

In order to fine-tune the language model efficiently, the food recipes were converted into a standardized prompt–completion format:

{"role": "system", "content": "You are an assistant"}
{"role": "user", "content": "What is 2+2?"}
{"role": "assistant", "content": "It's 4."}

This format emulates an instruction–response structure where the "instruction" is a user-provided list of ingredients and the "response" is a full cooking recipe generated by the model.

The formatting was made compatible with the Unsloth fine-tuning framework, enabling efficient training under constrained resources. By reducing the dataset to the most relevant fields, training time and memory usage were optimized for use on a free tier Google Colab environment.

After the formatting of the dataset, the number of entries became 4376. It was then reduced to 4000 recipes for training and 376 recipes reserved for testing and computing scores.

4.2.2 Training settings and methodology

The training process was carried out using a slightly modified version of the official Unsloth LLaMA 3 fine-tuning Jupyter notebook in Google Colab, on a Nvidia T4 GPU with 15GB VRAM, 12GB RAM and 112.7GB of storage using the Unsloth library in combination with PEFT (Parameter-Efficient Fine-Tuning) and LoRA (Low-Rank Adaptation).

The fine-tuning configuration included a batch size of 2 per device, with gradient accumulation step of 4 (simulating a batch size of 8), a learning rate of 2e-4 ($2*10^{-4}$ or 0.0002) with linear scheduling with the warmup over the first five steps, an AdamW 8 bit optimizer, no packing, 6 epochs and a mixed precision of fp16 (16-bit floating point).

Before the training starts, the loaded dataset is masked so the model only learns from the assistant's responses, not from the prompts/instructions provided by the user. This focuses the learning on the model's response and helps with reducing the computational costs.

4.2.3 Training statistics

After the training phase, some hardware statistics were discovered. Fine-tuning was done in 145.66 minutes (8739.6827 seconds), peak reserved memory was 2.16 GB, peak reserved memory for training was 0.529 GB, peak reserved memory percentage of max memory was 14.653 percent and the peak reserved memory for training percentage of maximum memory was 3.589 percentage. There also were plotted some statistics regarding model's performance. One way of measuring model performance is analyzing it's training loss [19]. Training loss is calculated during the training of the model and it reflects how well the model is fitting the training data. It should decrease over time as the model learns from the data, but lower values do not necessary mean better model. Very low values of training loss can also mean the model has overfit the data and learns the response and might not work well on new, unseen data. The training loss graph of the model during fine-tuning can be seen in Figure 4.4 with the average training loss highlighted.

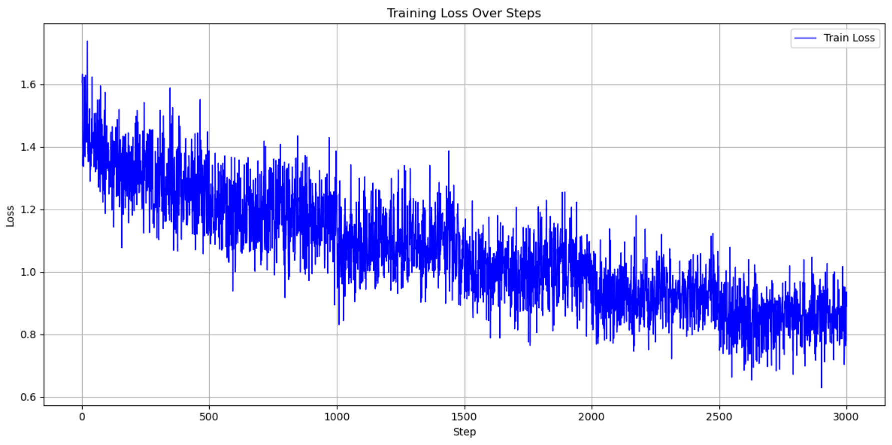


Figure 4.4: Training loss during the fine-tuning process

Average ROUGE-1 F1: 0.3366 Average ROUGE-L F1: 0.2166 Average Token-level F1: 0.2931

Afterwards the quality of the model's outputs was further evaluated using ROUGE [20] scores for both the original and the fine-tuned model. The ROUGE-1 score, which measures the overlap of unigrams between the generated and reference recipes, provides insight into the model's ability to replicate meaningful vocabulary. ROUGE-L, on the other hand, focuses on the longest common subsequence between reference and predicted texts, offering a more structural perspective. The scores obtained from the test set are shown in Table 4.1, reflecting the alignment between human-written and generated recipes. The ROUGE-1 score was also calculated for the two models. The original LLaMA 3.2 model had a ROUGE-1 score of 0.37 while the fine-tuned version of it resulted a score of 0.36.

Model	**ROUGE-L**
Trained from scratch (124M) [6]	0.37
Fine-tuned GPT-2 (124M) [6]	0.36
Fine-tuned GPT-2 (355M) [6]	0.37
ReciPal	0.18
Llama 3.2 1B Instruct	0.22

Table 4.1: Comparison of models on ROUGE-L score. Bold represents ReciPal model.

In addition to ROUGE, the F1 score [21] was also computed for the fine-tuned and the original model, using the same set of data, in order to evaluate the model's balance between precision and recall when generating recipe content. This score considers both false positives and false negatives in a token-level comparison between predicted and reference outputs. A higher F1 score indicates better token-level similarity and more accurate completions. The values obtained suggest that the model successfully captured essential information while maintaining fluency and structure, as shown in Table 4.2.

Model	**F1**
Trained from scratch (124M) [6]	0.75
Fine-tuned GPT-2 (124M) [6]	0.76
Fine-tuned GPT-2 (355M) [6]	0.77
ReciPal	0.36
Llama 3.2 1B Instruct	0.31

Table 4.2: Comparison of models on F1 score. Bold represents ReciPal model.

These results show that the fine-tuned model performs better at capturing relevant individual words and has an improved precision/recall of exact tokens. On the other hand, it struggles more with maintaining coherent or structured output.

4.3 Data communication and application flow

Regarding the interaction flow, the user first selects the necessary diets, cuisines (both optional) and submits a message (ingredient list) via the chat interface, then the frontend sends a POST request to the backend, the backend constructs a prompt and forwards it to the model which generates a cooking recipe and returns it to the backend, the backend formats the response and sends it to the frontend where the recipe is displayed in the chat. If the TTS is enabled, the recipe will also be sent to gTTS API, processed there and sent back to be played to the user. The whole process can be visualized in Figure 4.5.

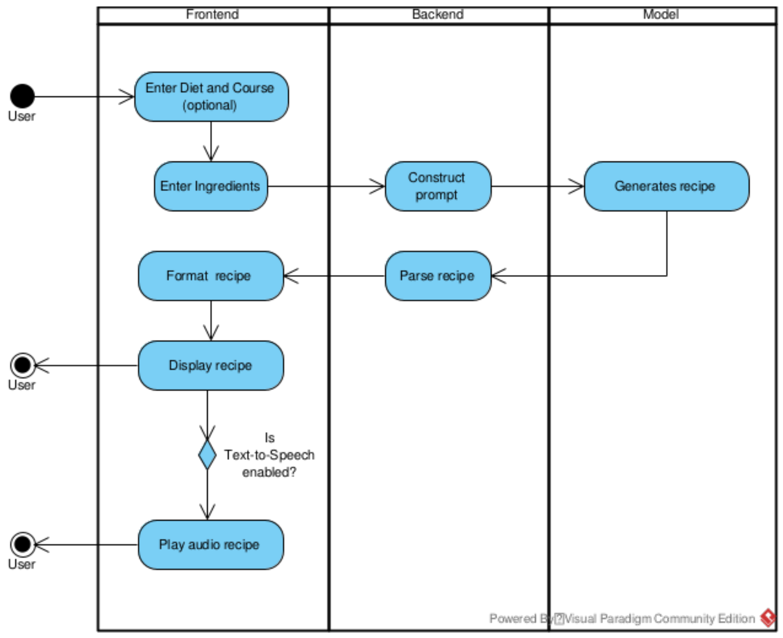


Figure 4.5: High-Level System Architecture

All data is currently processed and stored locally in the browser using the Web Storage API, specifically localStorage. This lets the application retain information such as recent inputs or responses across page refreshes during a session. However, this storage method is only client-side and offers no persistence across devices, browsers, or private browsing modes. Moreover, it is naturally limited in terms of capacity and security.

From a privacy standpoint, using local storage ensures that user data is not transmitted to any external server or service, which reduces exposure to unauthorized access data breaches or data leaks. However, it also introduces limitations: if the browser data is cleared or the user switches devices, all stored information is lost.

In terms of security, local storage is accessible via client-side JavaScript, which means it is vulnerable to cross-site scripting (XSS) attacks if the application is not properly secured. Since no user authentication or encryption mechanisms are implemented, sensitive data should not be stored in this format. For future versions of the system, introducing secure server-side storage and user authentication would allow for persistent sessions, better data control, and stronger guarantees regarding privacy and data protection.

4.4 Testing the app

This section provides step-by-step instructions for setting up, running, and interacting with the recipe generation web application developed as part of this thesis.

Requirements

The application consists of a backend, a frontend, and a locally hosted large language model served via Ollama. Below are the software and hardware requirements:

- Python 3.10 or higher
- Flask, Flask-Mail, requests, gTTS, and other required Python packages (can be done via ”*pip install -r requirements.txt*”)
- Ollama for model hosting
- Fine-tuned LLaMA 3.2 GGUF model
- A browser (Chrome, Firefox, etc.)

Installation and Setup

1. Clone the Repository

```
git clone https://gitlab.dev.info.uvt.ro/didactic/2025/licenta/ie/licentacodrinrata
cd licentacodrinrata2025/proiect
```

2. Create a virtual environment and install dependencies

This can be done either through IDE’s interface or through CLI.
bash:

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -r requirements.txt
```

3. Configure environment variables

Update the Flask-Mail settings in `app.py` or create a ‘.env‘ file containing:

```
MAIL_USERNAME=your_email@gmail.com
MAIL_PASSWORD=your_app_password
```

App passwords for Gmail can be generated from here.

4. Start the Ollama Server and Load the Model

```
ollama pull hf.co/codrin32/licenta2025-ReciPal-Q8_0-GGUF:Q8_0
ollama serve
```

5. Run the Flask application

```
python app.py
```

Using the Application

Once the app is running:

- Open a browser and go to `http://localhost:5000`
- Enter ingredients and dietary and cuisine preferences into the chat interface
- Wait for the recipe to be generated and displayed
- Use additional options like:
  - Sending the recipe via email
  - Listening to the recipe with the text-to-speech feature

Notes

- All data is stored locally in the browser via ‘localStorage‘
- No user accounts or cloud services are required
- The system works offline after the model is downloaded and installed
- Firefox might not support microphone input

Chapter 5

Application functionalities

5.1 User interaction: chat interface and input handling

The core user interaction occurs through a real-time chat interface that emulates a modern messaging application. Users can freely type ingredients (*"chicken, rice, carrots"*), select diet, course and select the number of cooking recipes to be generated. The system responds with the selected number of complete recipes, formatted to include a title, ingredients, preparation and cooking time and preparation steps. The key features of the chat are message history, as it preserves the conversation so users can see older generated cooking recipes, visual feedback for the model response wait time, as it indicates the instructions actually got to the model and it is processing the input.

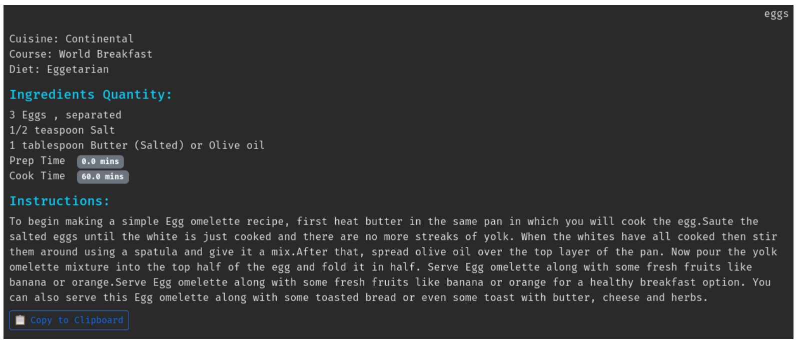
Users are encouraged to follow a simple but natural syntax (name of the ingredients separated by commas), as the model works best with it. The system accepts natural language expressions of ingredients, quantities and even contextual notes (e.g., *"leftover chicken from last night"*). This flexibility is enabled by the fine-tuned LLaMA 3.2 model, which processes the text input semantically rather than relying on predefined keywords. The diet input also has a custom field, where the user inputs his own diet, as his specific needs and requirements might not be found among the predefined ones.

The application also offers two forms for diet input (e.g., *vegetarian, vegan, low carb*) and course input (e.g., *breakfast, lunch, dinner, snack*). The diet form also has a custom field, where the user inputs his own diet, as his specific needs and requirements might not be found among the predefined ones. The use of the said forms is not mandatory as the system will not generate specific diet or course cooking recipes if none are selected.

To ensure robustness, the system includes basic error-handling mechanisms. One of those is invalid or empty input detection, where users are prompted to input something if the input field is empty or change the input if it is detected as invalid (e.g., only numbers with no characters). Timeout messages also represent another basic error-handling mechanism, as it tells the user that the model failed to respond within a reasonable time.

5.2 Cooking recipe generation and output presentation

At the core of the application is a fine-tuned instance of Meta's LLaMA 3.2 3B Instruct model. This model has been specifically adapted for the culinary domain using a curated dataset of food recipes. The model generates output in a structured way containing the title, the cuisine, the course and the diet of the cooking recipe, the preparation and the cooking time of the recipe, the ingredients list, containing the name and quantity of both the provided and other common ingredients (e.g., salt, oil, pepper), the instructions in order to prepare the meal.



```
                                                                     eggs
Cuisine: Continental
Course: World Breakfast
Diet: Eggetarian

Ingredients Quantity:
3 Eggs , separated
1/2 teaspoon Salt
1 tablespoon Butter (Salted) or Olive oil
Prep Time  0.0 mins
Cook Time  60.0 mins

Instructions:
To begin making a simple Egg omelette recipe, first heat butter in the same pan in which you will cook the egg.Saute the
salted eggs until the white is just cooked and there are no more streaks of yolk. When the whites have all cooked then stir
them around using a spatula and give it a mix.After that, spread olive oil over the top layer of the pan. Now pour the yolk
omelette mixture into the top half of the egg and fold it in half. Serve Egg omelette along with some fresh fruits like
banana or orange.Serve Egg omelette along with some fresh fruits like banana or orange for a healthy breakfast option. You
can also serve this Egg omelette along with some toasted bread or even some toast with butter, cheese and herbs.
```

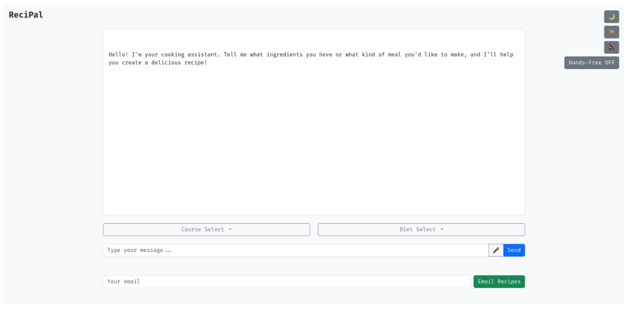
Figure 5.1: Generated recipe format

Once a food recipe is generated, it is displayed in a structured, readable format Figure 5.1. The raw text given by the model is formatted before it is displayed to be easier to read (e.g., title and text is highlighting, better spacing). The chat section also has a scroll-to-latest behavior to keep the latest generated recipe and text in view.

After the cooking recipe is done generating, it can also be copied to clipboard or sent through e-mail along with the other recipes generated in order to save it for future reference.

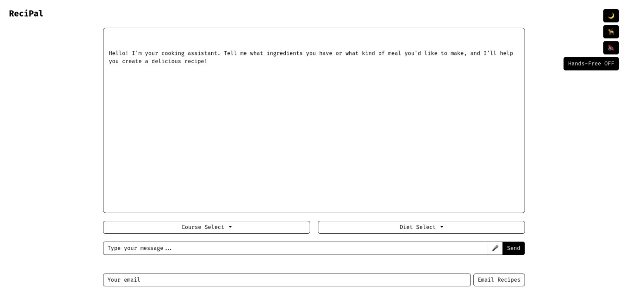
5.3 Accessibility features

While the app is not specifically designed for special needs people, it has some accessibility features. One of the features consists of a high contrast enabling button. It is designed to make everything easier to read improve and visibility for users with visual impairments or low vision. It works by increasing the color contrast between foreground elements (e.g., text, icons, buttons) and background elements. It works in both light and dark mode.



Hello! I'm your cooking assistant. Tell me what ingredients you have or what kind of meal you'd like to make, and I'll help you create a delicious recipe!

(a) Light mode, high contrast mode disabled

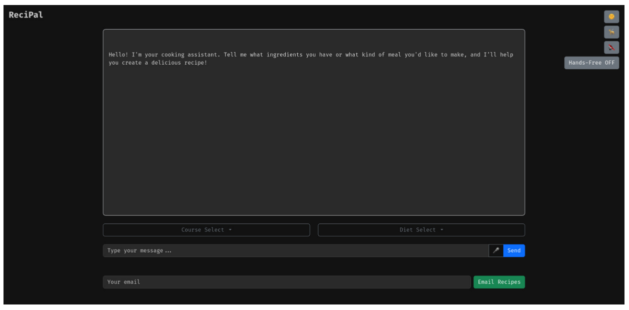


Hello! I'm your cooking assistant. Tell me what ingredients you have or what kind of meal you'd like to make, and I'll help you create a delicious recipe!

(b) Light mode, high contrast mode enabled

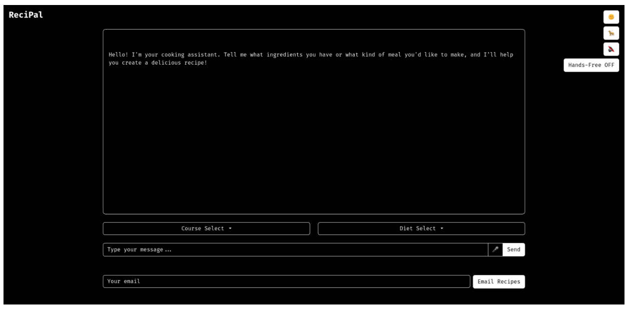
Figure 5.2: Comparison between light mode with high contrast turned disabled and enabled

The difference can be seen in Figure 5.2 for light mode and Figure 5.3 for dark mode.



Hello! I'm your cooking assistant. Tell me what ingredients you have or what kind of meal you'd like to make, and I'll help you create a delicious recipe!

(a) Dark mode, high contrast mode disabled



Hello! I'm your cooking assistant. Tell me what ingredients you have or what kind of meal you'd like to make, and I'll help you create a delicious recipe!

(b) Dark mode, high contrast mode enabled

Figure 5.3: Comparison between dark mode with high contrast turned disabled and enabled

Another feature is the ability to input the ingredients using the microphone.

This makes the whole process of ingredient input way easier without the need to use the keyboard.

The microphone input feature goes well with the text-to-speech feature, where the cooking recipe is read out loud after it finished generating.

The last two mentioned features can be both enabled by pressing a button, enabling a "hands-free" mode, where the only keyboard interaction needed is the press of any button in order to activate the listening of the microphone used for the input. After the listening has stopped, the input is automatically sent and the recipe is generated and played to the user.

Chapter 6

Conclusions and future work

6.1 Conclusions

The main objective of this thesis was to develop a web application capable of generating cooking recipes in response to ingredient lists as prompts, provided in natural language by the user. This was accomplished by fine-tuning the LLaMA 3 language model on a curated and filtered recipe dataset and integrating the model into a web application with a conversational frontend.

The author successfully carried out the curation and preprocessing a large dataset of international recipes to remove bias and ensure consistency, the formatting of the data into a prompt–completion structure compatible with instruction-based fine-tuning, the fine-tuning of the LLaMA 3 model using the Unsloth library and QLoRA optimization to enable efficient training on limited hardware, the deployment of the resulting model locally using the Ollama framework to ensure privacy, eliminate latency issues and reduce operational costs and the Development of a clean, responsive frontend using HTML, CSS and JavaScript, and the building of a Python-based Flask backend to serve user inputs and model responses.

The end result is a working basic web application capable of delivering personalized, structured recipes from ingredient lists, with a user-friendly, chat-style interface.

Throughout the course of this project, several technical and conceptual challenges appeared.

Data imbalance was one of them, as the original dataset was heavily biased towards Indian and North Indian cuisine, which could have caused the model to generate recipes mainly from that region. That was solved by curating and filtering the dataset and removing a good part of the Indian and North Indian recipes.

Another challenge was limited fine-tuning resources. Even though the model was not trained from scratch, the fine-tuning of it also took time and resources. To partially overcome this, the model was fine-tuned on a free-tier Google Colab Jupyter Notebook utilizing the Unsloth library, which drastically reduced the needed time and resources, but it still only allowed around 4 hours of running time per day.

The sensitivity of the prompt format was also a concern and a challenge. Ensuring the model responded well to user input required constant and careful refinement and formatting of the prompts. This was addressed through iterative testing and prompt redesign.

Another initial problem was integrating and hosting the model. An efficient and cost-effective way to do this was needed. This was solved by using Ollama, which allowed the model to run locally with almost no configuration required.

These challenges provided valuable learning opportunities and influenced several design decisions that ultimately strengthened the robustness and performance of the application.

While the system meets its core objectives, it is important to acknowledge its limitations.

The current implementation does not support multi-turn dialogue, each user input being treated independently. This limits the depth and continuity of conversations. Also because of this, the model does not store the user's preferences, dietary restrictions or previous interactions, all prompts being stateless and generic.

Language support is another current limitation. Both input and output are at this time only supported in English. This excludes non-English speaking people from benefiting from the application.

Another important limitation is recipe verification and feasibility. The generated recipes are based solely on textual inference and are not validated for nutritional accuracy, cooking and preparation time accuracy or safety.

The text to speech feature also has it's limitations. Currently the delay between the generation of the cooking recipe and the audio feedback is considerable. In order for the audio to be played, Google Text-To-Speech has to first fetch the recipe text, send it to the API for conversion into natural sounding voice, then the API sends the audio back and it is played to the user.

Currently the application has been only developed for desktop users, giving mobile users a poor experience using it.

6.2 Future work

During this work, several directions for further development and improvement were identified.

Expanding the model to understand and generate content in multiple languages would increase its accessibility and make it more useful to an international audience.

Multi-Turn Dialogue would also be a good improvement. Introducing memory and contextual awareness would allow users to ask follow-up questions, request substitutions, or adjust recipe parameters over the course of a session. This would also mean better fine-tuning using a whole conversation instead of a single request with a single response.

Also designing a responsive application for mobile or adapting the interface for mobile users is important. This would significantly improve usability and mobile user engagement.

Another possible improvement is user profiles and preferences. Implementing accounts and allowing users to manage their preferences would enable the system to always generate appropriate recipes without much user input when it comes to allergies, diet or course.

Adding a post-processing layer to validate and rate generated recipes based on known culinary rules or expert databases would increase trust in the application's output. This would also work by letting users give feedback to the recipes or allowing them to rate the recipes.

The text-to-speech delay issue can also be solved by using microservices. The TTS functionality could be separated from the main application logic by deploying it as an independent microservice. This architecture would allow asynchronous processing of the speech synthesis tasks and better scalability

Implementing these features would transform the current prototype into a fully-fledged intelligent cooking assistant.

Bibliography

[1] A. Grattafiori *et al.*, *The llama 3 herd of models*, 2024. arXiv: `2407.21783 [cs.AI]`. [Online]. Available: `https://arxiv.org/abs/2407.21783`.

[2] M. Liang and X. Hu, “Recurrent convolutional neural network for object recognition”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[3] P. Chhikara, D. Chaurasia, Y. Jiang, O. Masur, and F. Ilievski, “Fire: Food image to recipe generation”, in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Jan. 2024, pp. 8184–8194.

[4] A. Salvador, M. Drozdzal, X. Giro-i-Nieto, and A. Romero, “Inverse cooking: Recipe generation from food images”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[5] M. Goel, P. Chakraborty, V. Ponnaganti, M. Khan, S. Tatipamala, A. Saini, and G. Bagler, “Ratatouille: A tool for novel recipe generation”, in *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*, 2022, pp. 107–110. DOI: `10.1109/ICDEW55742.2022.00022`.

[6] H. H. Lee, K. Shu, P. Achananuparp, P. K. Prasetyo, Y. Liu, E.-P. Lim, and L. R. Varshney, “Recipegpt: Generative pre-training based cooking recipe generation and evaluation system”, in *Companion Proceedings of the Web Conference 2020*, ser. WWW ’20, Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 181–184, ISBN: 9781450370240. DOI: `10.1145/3366424.3383536`. [Online]. Available: `https://doi.org/10.1145/3366424.3383536`.

[7] L. Bossard, M. Guillaumin, and L. Van Gool, “Food-101 – mining discriminative components with random forests”, in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 446–461, ISBN: 978-3-319-10599-4.

[8] J. Marín, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba, “Recipe1m: A dataset for learning cross-modal embeddings for cooking recipes and food images”, *CoRR*, vol. abs/1810.06553, 2018.

[9] M. Bień, M. Gilski, M. Maciejewska, W. Taisner, D. Wisniewski, and A. Lawrynowicz, “RecipeNLG: A cooking recipes dataset for semi-structured text generation”, in *Proceedings of the 13th International Conference on Natural Language Generation*, B. Davis, Y. Graham, J. Kelleher, and Y. Sripada, Eds., Dublin, Ireland: Association for Computational Linguistics, Dec. 2020, pp. 22–28. DOI: `10.18653/v1/2020.inlg-1.4`. [Online]. Available: `https://aclanthology.org/2020.inlg-1.4/`.

[10] D. Batra, N. Diwan, U. Upadhyay, J. S. Kalra, T. Sharma, A. K. Sharma, D. Khanna, J. S. Marwah, S. Kalathil, N. Singh, R. Tuwani, and G. Bagler, “Recipedb: A resource for exploring recipes”, *Database*, vol. 2020, baaa077, Nov. 2020, ISSN: 1758-0463. DOI: `10.1093/database/baaa077`. eprint: `https://academic.oup.com/database/article-pdf/doi/10.1093/database/baaa077/34526636/baaa077.pdf`. [Online]. Available: `https://doi.org/10.1093/database/baaa077`.

[11] E. Aguilar, B. Remeseiro, M. Bolaños, and P. Radeva, “Grab, pay, and eat: Semantic food detection for smart restaurants”, *IEEE Transactions on Multimedia*, vol. 20, no. 12, pp. 3266–3275, 2018. DOI: `10.1109/TMM.2018.2831627`.

[12] P. Pouladzadeh and S. Shirmohammadi, “Mobile multi-food recognition using deep learning”, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 13, no. 3s, pp. 1–21, 2017.

[13] H. Wang, G. Lin, S. C. Hoi, and C. Miao, “Structure-aware generation network for recipe generation from images”, in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII 16*, Springer, 2020, pp. 359–374.

[14] B. P. Majumder, S. Li, J. Ni, and J. J. McAuley, “Generating personalized recipes from historical user preferences”, *CoRR*, vol. abs/1909.00105, 2019. arXiv: `1909.00105`. [Online]. Available: `http://arxiv.org/abs/1909.00105`.

[15] A. Meyers, N. Johnston, V. Rathod, A. Korattikara, A. Gorban, N. Silberman, S. Guadarrama, G. Papandreou, J. Huang, and K. P. Murphy, “Im2calories: Towards an automated mobile vision food diary”, in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[16] M.-Y. Chen, Y.-H. Yang, C.-J. Ho, S.-H. Wang, S.-M. Liu, E. Chang, C.-H. Yeh, and M. Ouhyoung, “Automatic chinese food identification and quantity estimation”, in *SIGGRAPH Asia 2012 Technical Briefs*, ser. SA ’12, Singapore, Singapore: Association for Computing Machinery, 2012, ISBN: 9781450319157. DOI: `10.1145/2407746.2407775`. [Online]. Available: `https://doi.org/10.1145/2407746.2407775`.

[17] J. Chen and C.-w. Ngo, “Deep-based ingredient recognition for cooking recipe retrieval”, in *Proceedings of the 24th ACM International Conference on Multimedia*, ser. MM ’16, Amsterdam, The Netherlands: Association for Computing Machinery, 2016, pp. 32–41, ISBN: 9781450336031. DOI: `10.1145/2964284.2964315`. [Online]. Available: `https://doi.org/10.1145/2964284.2964315`.

[18] S. Barko-Sherif, D. Elsweiler, and M. Harvey, “Conversational agents for recipe recommendation”, in *Proceedings of the 2020 Conference on Human Information Interaction and Retrieval*, ser. CHIIR ’20, Vancouver BC, Canada: Association for Computing Machinery, 2020, pp. 73–82, ISBN: 9781450368926. DOI: `10.1145/3343413.3377967`. [Online]. Available: `https://doi.org/10.1145/3343413.3377967`.

[19] T. Ishida, I. Yamane, T. Sakai, G. Niu, and M. Sugiyama, *Do we need zero training loss after achieving zero training error?*, 2021. arXiv: `2002.08709 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2002.08709`.

[20] C.-Y. Lin, “Rouge: A package for automatic evaluation of summaries”, in *Text summarization branches out*, 2004, pp. 74–81.

[21] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks”, *Information Processing Management*, vol. 45, no. 4, pp. 427–437, 2009, ISSN: 0306-4573. DOI: `https://doi.org/10.1016/j.ipm.2009.03.002`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0306457309000259`.