WeatherFrog Abstract by Joop Timm

1. Overview and Objective

The goal of the project was to design and implement a chatbot for the weather channel: “Weather-Frog”. The chatbot enables users to easily request weather information anywhere and at any time it is required. It therefore consolidates the weather channel’s audience, but also expands the target audience, by reaching new groups that tend to use more modern media.

2. Resources

The chatbot framework Rasa CALM (Bocklisch, Werkmeister, Varshnaya, & Nichol, 2024) is used to implement the chatbot. Rasa is a partly open-source framework with flexible implementation options and is also easily connected to several target applications such as websites, WhatsApp chats, Facebook chats and many more. The CALM framework from rasa enables the connection of state-of-the-art Large Language Models (LLMs) including ChatGPT (OpenAI, 2022), Llama (Meta Platforms Inc., 2024), Cohere (Cohere, 2024) and others to reach optimal language understanding- and language generation capabilities. The LLM used for this project is GPT-3.5-Turbo by OpenAI.

To achieve reliable and constantly updated weather forecast abilities for the chatbot, the Open WeatherMap API (OpenWeather, 2024) is used. Open WeatherMap API is free of charge in its two base formats while having the option to switch to several pay-models for a larger forecast range as well as more precise, hourly forecasts. The API covers predictions for all major cities in the world.

3. Contents and Implementation

The implementation of the chatbot revolves around Rasa’s core components: config, domain, flows, custom actions and endpoints:

Config (/config.yml): The first step is to implement the general configuration. The language is set to English and the pipeline for Rasa CALM is configured to enable the external LLM Command Generator GPT-3.5-turbo. Additionally, the personal API-key must be set in the environment configuration via the terminal. The temperature parameter indicates how “freely” the LLM should operate. The value 0.7 was tested to have a decent flexibility in language expression without any tendencies to hallucinate. For the flow policy Rasa’s standard policy is used.

Domain (/domain.yml): In the domain file all general information is defined: predefined responses, an overview of all actions, entities and slots for varying information from the user which is used further in the conversation and actions. The first slot is “location” for the requested weather. The second is “weather_time” to adjust the request for the queried weather. And the last is “request_type”: a categorical slot for special requests that is set accordingly. This includes cloudiness, feels like, humidity, rainfall, rainy, snowfall, snowy, sunshine, temperature and wind speed.

Flows (/data/flows.yml): The conversation logic in WeatherFrog is implemented via Rasa flows. Flows are learned by the language model through the description of the respective flow, so the length and structure of the description depends on the complexity of the task the flow must fulfil. They are activated by certain inputs via the learned description and then follow a step-by-step logic for the task at hand. The general weather flow activates when the user prompts a general weather request. If there is a time given the “weather_time” slot is set accordingly in the “collect: weather_time” step. As times are given in many different formats by the user, the collection step is complex, thus the step requires an extensive description with many different examples to function properly. Earlier versions with only a short description have proven insufficient. When no time is given, the slot is set to today. The next step is collecting the location for the request. When no location is given in the initial prompt,

the chatbot will specifically ask for it. The flow will then pass on the request to the custom action “action_get_weather”, which will make an API call and will then result in a response to the request. The answer is then rephrased by the LLM to provide a more natural user experience. For specific weather requests the respective flow is extended with the “collect: request_type”-step and the descriptions of the flow and “collect: time” steps are adjusted for the specific task.

Non-weather requests such as the bot challenge and information about the bot were initially thought to be implemented via Rasa rules to have a stricter structure which then redirects the user to weather requests. Testing showed that flows had much better results here as well, as the LLM command generator has better language understanding capabilities than the internal Rasa NLU packages. To avoid any deviances from the norm for some flows the rephrasing of the LLM is disabled here.

Custom Actions (/actions.py): The API call to the Open Weathermap API is formulated in the custom action “action_get_weather”. The custom action uses slots to extract the right information. First the “weather_time” slot is parsed into a time difference to the current date. For the API call the URL is composed from skeleton URL API-key and location. The response-json for the next four days is then filtered to provide daily information at noon. All of the necessary information is extracted for the requested time. From there an answer is composed depending on specific information and time requested and returned to the user.

Endpoints (/endpoints.yml): To bring it all together in endpoints, an action endpoint is established. The rephraser is defined for a better user experience. A temperature of 0.7 served well as a general specification, but this is disabled or adjusted for certain answers in the domain file.

4. Reflection and Outlook

The bot is capable of providing general weather information and also some specific requests as desired. The current user experience is good due to the LLM’s rephrasing providing a decent natural feel for the user. Thus the initial version of WeatherFrog is implemented successfully.

Nevertheless, while the chatbot achieves what it should, there is still a lot of room for improvement. While there is some variance in specific weather requests, these can be further extended to provide all of the information available. Additionally, the logic of the slot and custom action could be changed to be able to combine several specific requests in one flow. The forecast is currently also limited to four days and daily information. With some alterations and changing the WeatherMap API call this could be expanded to a 16-day forecast and hourly information. This however would require a payment plan for the API, as this functionality exceeds the free version.

The user experience so far is decent, with some variance in the responses. But GPT-3,5-turbo can already be seen as an outdated model, which also makes it cheaper to apply than state-of-the-art LLMs. Advancements in LLMs are constant and fast, so updating the language models accordingly would guarantee state-of-the-art user experience. However this also requires regular reiteration and high prices per prompt. A possible current improvement would be to upgrade to GPT-4o.

Currently the time parser is a function within the custom action. While working decently, it sometimes does not recognize all date formats within the users prompts. The initial idea was to use the LLM to extract the date directly to a categorical slot which could then be used within the function without further processing. But GPT-3.5-turbo was not able to learn to do this reliably. An upgrade to the LLM could enable a reliable assignment of the categorical “weather_time” slot and avoid the extra step in the custom action which currently, although necessary in this iteration, resembles a doubled step.

The general framework and code are built flexibly for all these changes and upgrades to be applied with relative ease, so the foundation is laid for the already successful chatbot to be further developed.

References

Bocklisch, T., Werkmeister, T., Varshnaya, D., & Nichol, A. (2024). Task-Oriented Dialogue with In-Context Learning. *arXiv*.

Cohere. (2024, 5 22). *Cohere*. Retrieved from https://cohere.com/

Meta Platforms Inc. (2024, 5 27). *Llama*. Retrieved from https://llama.meta.com/

OpenAI. (2022, 11 30). *ChatGPT*. Retrieved from openai.com: https://openai.com/index/chatgpt/

OpenWeather. (2024, 5 27). *openweathermap*. Retrieved from openweathermap.org/api: https://openweathermap.org/api