Lucas Monroe

Locally Hosted DeepSeek ChatBot

03/11/2025

Project Overview

Ollama

Model Hosting

Ollama enables users to run open-source LLMs locally and offers libraries and APIs for interacting with the installed models.

DeepSeek R1:7b

The Model

DeepSeek R1 is an open-source LLM made with a focus on powerful reasoning such as coding problems. The model provides detailed explanations of how it approaches each prompt.

Python

Back-End

A python back-end manages data flow between the user and Ollama, while also storing chat history within conversations.

Gradio

Front-End

Gradio provides a Python library for easily building interactive web interfaces specifically for machine learning & AI applications.

New chat

Chatbot

Implement the Python "pop" function from scratch.

Okay, so I need to implement the Python 'pop' function from scratch. Hmm, where do I start? Let me think about what pop does. From what I remember, it's a list method that removes an item at a specified index and returns it. If no index is given, it defaults to removing the last element.

Wait, so how is this done in Python internally? I guess when you call pop on a list with an index, it finds the item at that position and then returns it after deleting it from the list. But since lists are mutable, modifying them can have side effects, right?

So, maybe I should write a function called pop that takes two arguments: the list itself and an optional index. If the index is not provided or is None, it defaults to -1, which removes the last element.

Let me outline the steps:

1. Define the function with 'lst' as the first parameter and 'index' as the second, defaulting to None.
2. Check if 'index' is None; if so, set it to -1.
3. Validate that 'index' is an integer because you can't have a non-integer index in a list.
4. Also, make sure the index is within the valid range. The valid indexes are from 0 to len(lst)-1. If the index is out of bounds, maybe raise an IndexError like Python's built-in function does.

Wait, what if someone passes a negative number for 'index'? Oh right, in Python, using a negative index counts from the end. So for example, -1 refers to the last element. But I'm not sure if we should handle that o st let it raise an error because maybe the user expects it to behave like built-in pop

New chat

Implement the Python "pop" function from...

Chatbot

plicitly provided indices correctly, ensuring that invalid indices raise appropriate exceptions.

Approach

1. **Default Index Handling**: The method should default to popping the last element if no index is provided.
2. **Type Check**: Ensure that the provided index is an integer; otherwise, a `TypeError` should be raised.
3. **Index Validity Check**: After converting or setting the index correctly, validate whether it falls within the bounds of the list. If not, raise an `IndexError`.
4. **Exception Handling**: Properly handle any exceptions raised during the popping operation to ensure consistent error messaging.

Solution Code

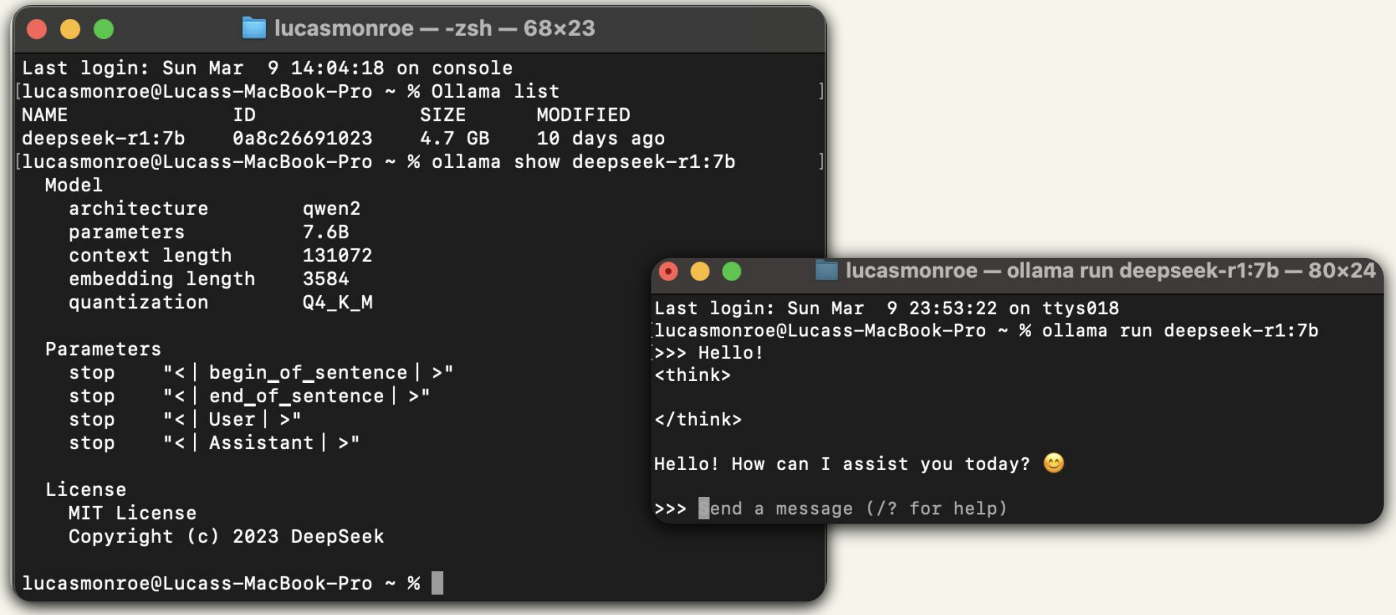
```
def custom_pop(lst, index=None):
    if index is None:
        index = -1
    else:
        if not isinstance(index, int):
            raise TypeError("Index must be an integer")

    try:
        return lst.pop(index)
    except IndexError as e:
        raise IndexError("list index out of range") from e
```

Explanation

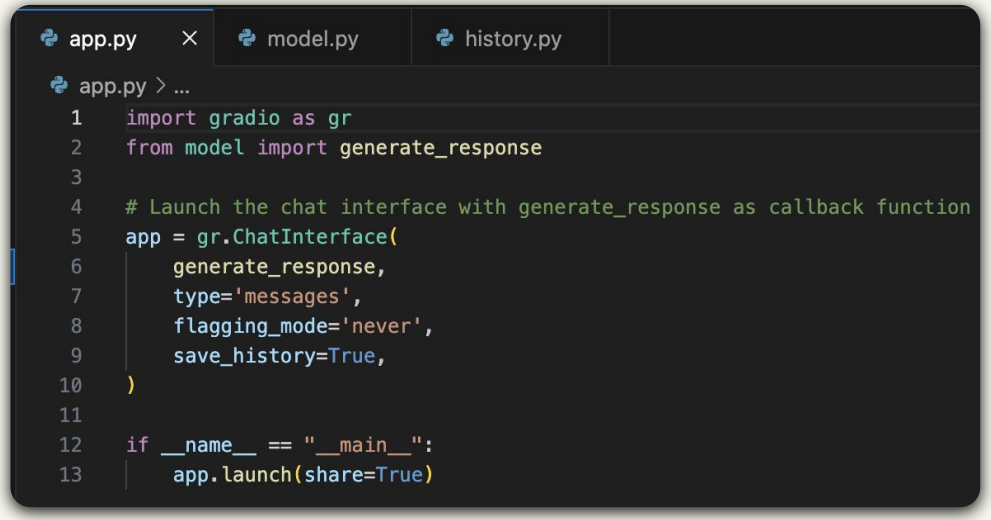
- **Default Index Handling**: If no `index` is provided (i.e., it's `None`), the method defaults to popping the last element using `-1`.
- **Type Check**: The else clause ensures that if an index is provided, it must be of type `int`. If not, a `TypeError` is raised immediately.
- **Index Validity Check**: The `pop` method inherently checks if the index is valid. Any exceptions caught by the `try-except` block are re-raised

Ollama & DeepSeek



app.py

This component serves as the "glue" of the application, seamlessly connecting the front end and back end. It achieves this by launching a custom Gradio interface, which leverages the generate_response function from model.py to facilitate back-end communication with DeepSeek through Ollama



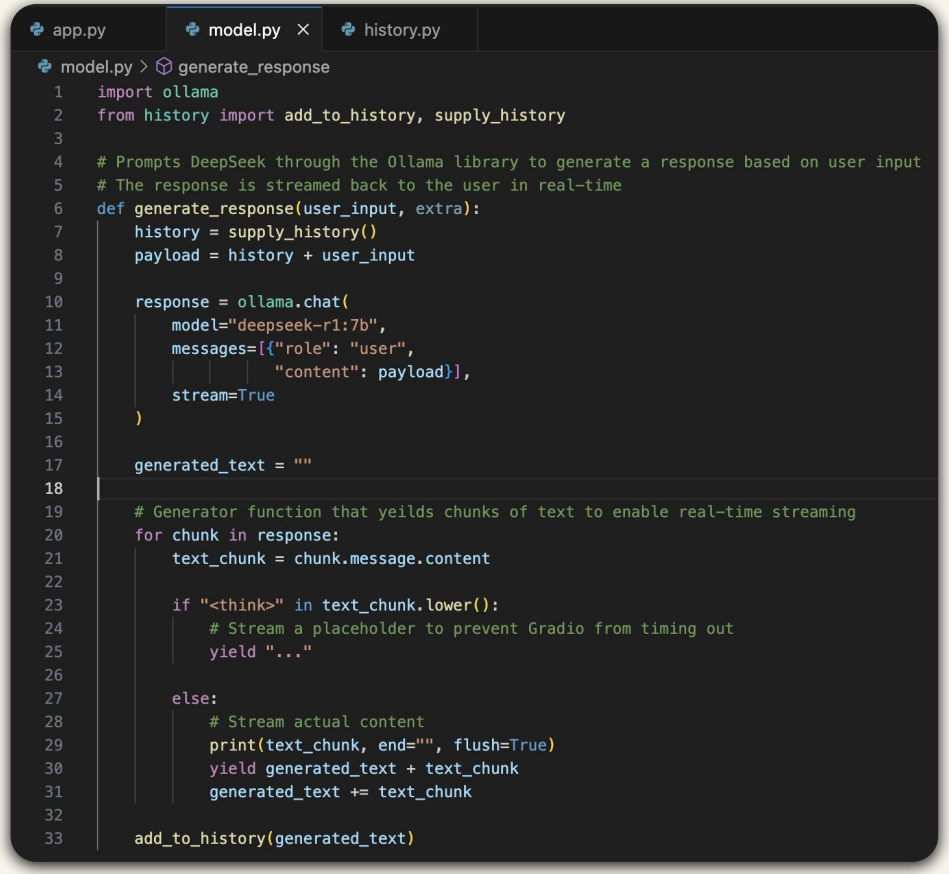
model.py

This is the backbone of the application, handling the flow of data between the user and DeepSeek. It passes the user's prompt to DeepSeek, retrieves its response, and delivers it to the front end.

generate_response() constructs a payload containing the user input and chat history, then uses Ollama's chat function to send it to DeepSeek.

As DeepSeek generates a response, a generator function streams chunks of it to the front end in real time.

Finally, the new response is appended to the chat history.



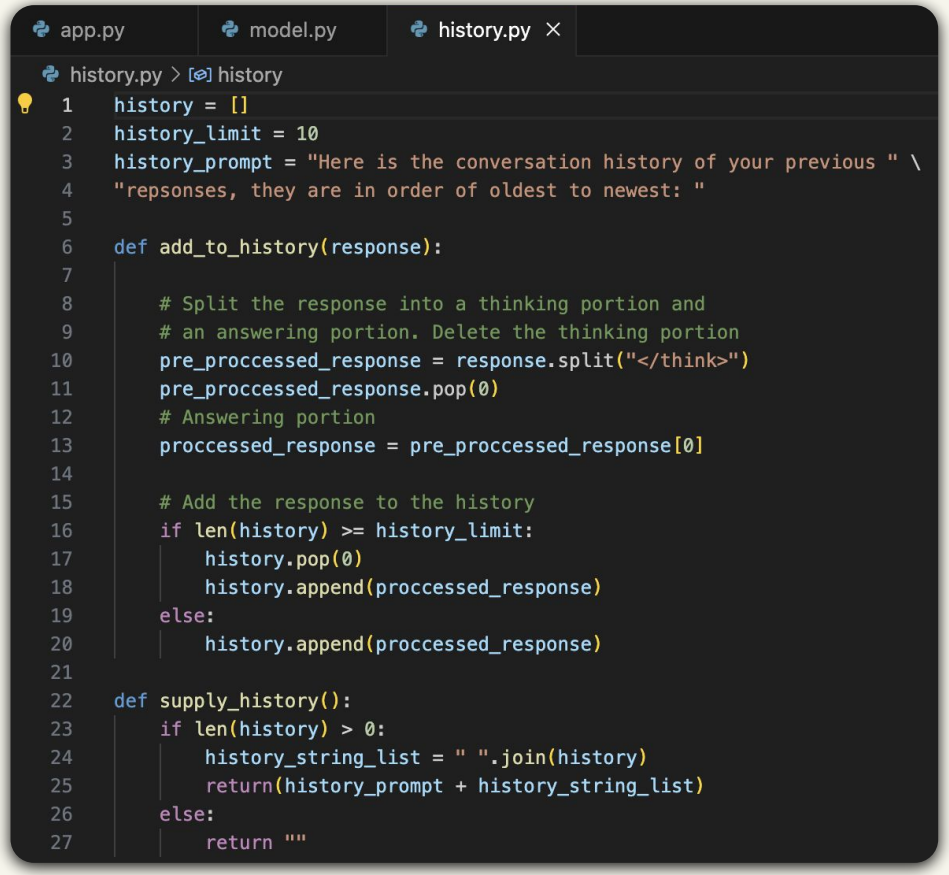
history.py

This class manages history collection and serves it to DeepSeek.

add_to_history() processes each DeepSeek response by splitting them into a "thinking" and "answering" portion. The thinking portion is discarded for efficiency, while the answering portion is appended to the history list.

To maintain relevance, only the 10 most recent responses are stored. Once the limit is reached, the oldest response is removed.

supply_history() restructures the history list to a string for Gradio, including a contextual prompt for DeepSeek. If no history exists, nothing is served.



Changes & Challenges

- **Initial implementation of model.py didn’t support real-time streaming**
  - Responses were displayed once they were complete, not as they were being made
  - User friendly response streaming was implemented using a generator function

- **Initial implementation of model.py used an API**
  - Fun to implement, but unnecessary and over complicated
  - API returned data in a JSON structure which played a part in the issue below

- **Initial implementation of model.py received responses from DeepSeek in a JSON structure**
  - An intermittent issue caused responses to appear in the terminal but not in the front end
  - After extensive debugging, reading documentation, adding try/catch statements, and rewriting code, no clear error or solution emerged
  - Seeking help on the Ollama subreddit, a user pointed out that DeepSeek's JSON structure is unpredictable and not recommended
  - Switching to the Ollama Python library for model communication eliminated reliance on JSON data
  - Additionally, displaying placeholder text prevents Gradio from timing out while DeepSeek processes responses

Interactive Demo!