

Knight Foundation School of Computing and Information Sciences

Fall 2024 Senior Design Project

StarsGPT

Team Members: Michael Camelio, Richard Chong, Andres Lopez, Jason Martinez, Carlos Ramirez
Mentor: Dr. Patricia McDermott-Wells, Florida International University
Instructor/Faculty: Dr. Masoud Sadjadi, Florida International University

OpenAI GPT Chat Example

StarsGPT

User: Can you solve this assignment for me? Write a function in Python that reverses a string.

Assistant: I can't write the function for you, but I can help you understand how to approach this problem!

To reverse a string in Python, you can use a few different methods. One common way is to use slicing. Do you know how string slicing works in Python? If not, I can explain that concept to you!

Here's a hint to get you started:

You can access parts of a string using the syntax `string[start:end:step]`.

What do you think the step value would be if you wanted to reverse the string?

Please enter your message below:

Press Ctrl+Enter to apply.

Submit

Groq Chat Example

Polymorphism in Action

Now let's put polymorphism into action by creating an array of `Animal` objects and calling the `sound()` method on each object:

```
public class Main {
    public static void main(String[] args) {
        Animal[] animals = new Animal[] {
            new Dog(),
            new Cat(),
            new Animal() // Just an animal, not a specific type
        };

        for (Animal animal : animals) {
            animal.sound();
        }
    }
}
```

// Output:
// The dog barks.
// The cat meows.
// The animal makes a sound.

In this example, we create an array of `Animal` objects and add a `Dog`, a `Cat`, and just an `Animal` (not a specific type). We then call the `sound()` method on each object in the array using a loop. Since each object is an instance of `Animal`, we can call the `sound()` method on each one, and the correct implementation will be called based on the actual type of the object (e.g., `Dog` or `Cat`).

This is an example of polymorphism in action, where we can write code that works with different data types without knowing their specific implementation at compile time.

```
if api_choice == "OpenAI" and st.session_state.openValidated:
    #OpenAI scenario
    st.session_state.openaiConvo.append({"role": "user", "content": user_input})
    history = [{"role": msg["role"], "content": msg["content"]} for msg in st.session_state.openaiConvo[:-1]]
    session_id = str(uuid.uuid4())
    response = st.session_state.chat.invoke({"input": user_input, "history": history}, {"configurable": {"session_id": session_id}})
    st.session_state.openaiConvo.append({"role": "assistant", "content": response.content})
```

```
with message_container:
    if api_choice == "OpenAI":
        displayed_chat = st.session_state.openaiConvo
        for msg in displayed_chat:
            st.markdown(f"**{msg['role'].capitalize()}** {msg['content']}")
    elif api_choice == "Groq":
        displayed_chat = st.session_state.groqConvo
        for msg in displayed_chat:
            st.markdown(f"**{msg['role'].capitalize()}** {msg['content']}")
```

- User Conversation Handling for Storage and Display
- Conversations are stored in session memory to simplify handling and display
 - Users can seamlessly switch between models, with each model displaying its respective conversation history

```
@staticmethod
def initialize_chain(model: str, input_api_key: str) -> Any:
    examples = [
        {
            "input": "What is a loop in Python?",
            "output": ""
        }
    ]
    Are follow up questions needed here: Yes.
    Follow up: Does this response do the assignment for you?
    Intermediate answer: No.
    Follow up: What is a loop in Python?
    So the final answer is: In Python, a loop is a way to iterate over a sequence (like a list or tuple) or other iterable objects.
    """
```

- Chain Initialization with Examples and Final Prompt
- The chat is created based on the user-selected API and their provided API key
 - Existing examples are combined with the final prompt text to guide the chatbot's responses more effectively

```
def openAi_key_check(key):
    try:
        messages = [SystemMessage(content="You only reply 'Yes'."), HumanMessage(content="Yes?"), ]
        test = ChatOpenAI(model="gpt-4o-mini", temperature=0.0, api_key=key)
        response =test.invoke(messages)
        del response
        del test
        del messages
        gc.collect()
        return True
    except Exception as e:
        return False
```

- API Key Validation Functions
- Validate keys before creating chain
 - Clear used variables to minimize memory usage

ACKNOWLEDGEMENTS

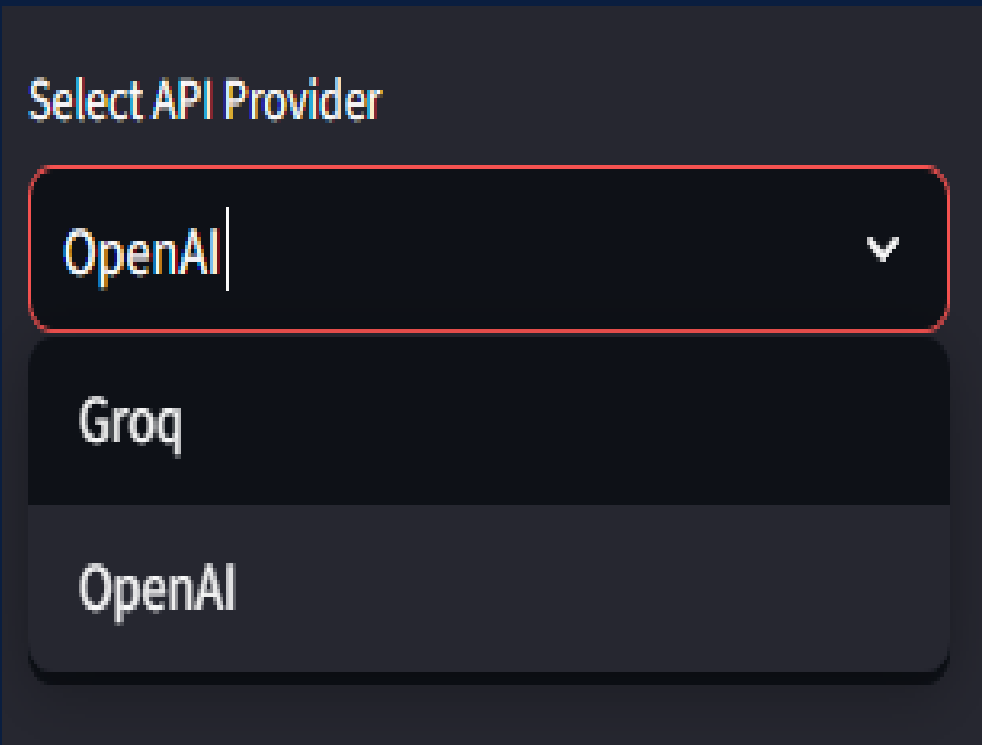
We thank Dr. Patricia McDermott-Wells for her assistance and mentorship that we received throughout the senior design project. We would also like to acknowledge OpenAI for their GPT models, APIs, and supporting documentation; Groq for their APIs enabling access to LLaMA models and accompanying documentation; Meta for developing the LLaMA models; LangChain for their libraries and tools used to handle the chatbot's chain and supporting documentation; and Streamlit for their services and documentation, which helped us create our application.

```
elif api_choice == "OpenAI":
    openaiApi = st.sidebar.text_input("OpenAI API Key", type="password", value=st.session_state.openAIEnteredKey or "")

    if openaiApi and not st.session_state.openValidated:
        if openAi_key_check(openaiApi):
            st.session_state.openValidated = True
            st.session_state.openAIEnteredKey = openaiApi
            ChatChainSingleton.model = "gpt-4o-mini"
            ChatChainSingleton.input_api_key = openaiApi
            st.session_state.chat = ChatChainSingleton().chain
        else:
            st.session_state.openValidated = False
            st.warning("Invalid OpenAI API key. Please enter a valid key!")
```

Handling API Choices and Chain Creation

- Implements the logic for initializing the selected API
- Stores the user's API key in session memory to enable seamless switching between models



Individual Contributions

- **Frontend:** Designed and developed the user interface for the chat page, including conversation display, API key input, and selection handling
- **Backend:** Implemented backend logic for API integrations, created chain objects for respective chatbots, managed session states, and handled chat conversations
- **Frontend-Backend Integration:** Ensured seamless interaction between frontend and backend components, enabling dynamic switching between OpenAI and Groq APIs