

## Objective:

<https://github.com/UnifiedNeuralNetwork/NeuralGPT/blob/main/mmk%20Chatflow1.json>

test the integration of NeuralGPT agent with FlowiseAI in practice and is functional in GUI by opening a dialogue window of FlowiseAI with NeuralGPT .

## Task:

1. Create a function that takes in a user input and sends it to the NeuralGPT agent for processing. The function should then take the response from the agent and display it in the FlowiseAI dialogue window.

## Result:

To create a function that takes in a user input and sends it to the NeuralGPT agent for processing, you can use the following code:

```
import requests
import json

def get_neuralgpt_response(user_input):
    url = "http://neuralgpt-agent-url.com/ask" # Replace with actual URL of NeuralGPT agent
    payload = {"text": user_input}
    headers = {"Content-Type": "application/json"}
    response = requests.post(url, data=json.dumps(payload), headers=headers)
    return response.json()["response"]
```

In this code, we use the `requests` library to make a POST request to the NeuralGPT agent URL with the user input as the payload. We then extract the response from the JSON returned by the agent and return it.

To display the response in the FlowiseAI dialogue window, you can use the following code:

```
import flowise

def display_response(response):
    flowise.send_message(response)
```

In this code, we use the `flowise` library to send the response as a message in the FlowiseAI dialogue window.

Putting it all together, the complete code would look something like this:

```
import requests
import json
import flowise

def get_neuralgpt_response(user_input):
```

```

url = "http://neuralgpt-agent-url.com/ask" # Replace with actual URL of NeuralGPT agent
payload = {"text": user_input}
headers = {"Content-Type": "application/json"}
response = requests.post(url, data=json.dumps(payload), headers=headers)
return response.json()["response"]

def display_response(response):
    flowise.send_message(response)

# Example usage
user_input = "Hello, how are you?"
response = get_neuralgpt_response(user_input)
display_response(response)

```

Note that you will need to replace the `url` variable with the actual URL of the NeuralGPT agent. Also, the `flowise` library may have different syntax depending on the specific implementation of the FlowiseAI dialogue window.

Task:

2. Implement a feature that allows the user to switch between different NeuralGPT models within the FlowiseAI dialogue window. This can be achieved by creating a dropdown menu that lists the available models and updating the function from task 1 to use the selected model.

Result:

To implement the feature of switching between different NeuralGPT models within the FlowiseAI dialogue window, you can follow these steps:

1. Define a list of available models that the user can select from. For example:

```

models = {
    "model1": "path/to/model1",
    "model2": "path/to/model2",
    "model3": "path/to/model3"
}

```

2. Create a dropdown menu in the FlowiseAI dialogue window that lists the available models. For example:

```

<select id="model-select">
  <option value="model1">Model 1</option>
  <option value="model2">Model 2</option>
  <option value="model3">Model 3</option>
</select>

```

3. Update the function from task 1 to use the selected model. For example:

```

import openai
openai.api_key = "YOUR_API_KEY"

def process_input(input_text):
    selected_model = document.getElementById("model-select").value
    model_path = models[selected_model]
    response = openai.Completion.create(
        engine=model_path,
        prompt=input_text,
        max_tokens=1024,
        n=1,
        stop=None,
        temperature=0.7,
    )
    return response.choices[0].text

```

4. Call the `process\_input` function whenever the user inputs a message in the dialogue window. The response from the function should then be displayed in the window.

With these steps, you should be able to implement the feature of switching between different NeuralGPT models within the FlowiseAI dialogue window.

Task:

3. Add error handling to the function from task 1. If the response from the NeuralGPT agent is empty or contains an error message, display a user-friendly error message in the FlowiseAI dialogue window instead of the agent's response.

Result:

To add error handling to the function from task 1, you can modify the existing code as follows:

```

import openai
import json

def generate_response(prompt, model):
    try:
        openai.api_key = "YOUR_API_KEY"
        response = openai.Completion.create(
            engine=model,
            prompt=prompt,
            max_tokens=1024,
            n=1,
            stop=None,
            temperature=0.5,
        )
        message = response.choices[0].text.strip()
        if message == "":
            raise ValueError("Empty response from NeuralGPT agent")
        elif "error" in message.lower():
            raise ValueError("Error message received from NeuralGPT agent")
        return message
    except Exception as e:
        error_message = "Error: " + str(e)
        return error_message

# Example usage
prompt = "Hello, how are you?"

```

```
model = "davinci"
response = generate_response(prompt, model)
print(response)
```

In this modified code, we have added a try-except block to catch any exceptions that may occur during the API call to the NeuralGPT agent. We have also added two conditions to check if the response from the agent is empty or contains an error message. If either of these conditions is true, we raise a ValueError with an appropriate error message. Otherwise, we return the response from the agent as before.

You can integrate this modified code into the existing FlowiseAI dialogue window code to display a user-friendly error message in case of an error.

#### Task:

Create a function that extracts the available NeuralGPT models from the NeuralGPT agent and displays them in the FlowiseAI dialogue window. This can be achieved by modifying the function from task 2 to also retrieve the available models from the agent and populate the dropdown menu with them.

#### Result:

Here is an example function that retrieves the available NeuralGPT models from the agent and populates the dropdown menu with them:

```
import requests
import json

def get_available_models():
    # Make a GET request to the NeuralGPT agent API to retrieve the available models
    response = requests.get('http://localhost:8000/models')

    # Check if the response is valid
    if response.status_code != 200:
        # If the response is not valid, display a user-friendly error message in the
        # FlowiseAI dialogue window
        print('Error: Unable to retrieve available models from the NeuralGPT agent')
    else:
        # If the response is valid, extract the available models from the response
        # body
        available_models = json.loads(response.text)['models']

    # Populate the dropdown menu with the available models
    dropdown_options = []
    for model in available_models:
        dropdown_options.append({'label': model, 'value': model})

    return dropdown_options
```

This function can be called within the main function from task 2 to populate the dropdown menu with the available models. Here is an example of how to modify the main function:

```

import requests
import json

def handle_message(message):
    # Make a POST request to the NeuralGPT agent API with the user's message
    response = requests.post('http://localhost:8000/ask', json={'message': message})

    # Check if the response is valid
    if response.status_code != 200:
        # If the response is not valid, display a user-friendly error message in the
        # FlowiseAI dialogue window
        print('Error: Unable to retrieve response from the NeuralGPT agent')
    else:
        # If the response is valid, extract the response from the response body
        response_text = json.loads(response.text)['response']

        # Check if the response is empty or contains an error message
        if not response_text or 'error' in response_text.lower():
            # If the response is empty or contains an error message, display a
            # user-friendly error message in the FlowiseAI dialogue window
            print('Error: Unable to generate a response for your message')
        else:
            # If the response is valid, display it in the FlowiseAI dialogue window
            print(response_text)

    # Retrieve the available models from the NeuralGPT agent
    available_models = get_available_models()

    # Populate the dropdown menu with the available models
    dropdown_menu = {'type': 'dropdown', 'label': 'Select a model', 'name':
        'model', 'options': available_models}

    # Display the dropdown menu in the FlowiseAI dialogue window
    print(dropdown_menu)

```

Note that this code assumes that the NeuralGPT agent API is running on localhost:8000. You may need to modify the API URL to match your specific setup.

#### Task:

Implement a feature that allows the user to input custom parameters for the NeuralGPT agent within the FlowiseAI dialogue window. This can be achieved by creating a form that allows the user to input the desired parameters and modifying the function from task 1 to include these parameters in the request to the agent.

#### Result:

To implement a feature that allows the user to input custom parameters for the NeuralGPT agent within the FlowiseAI dialogue window, you can follow these steps:

1. Modify the FlowiseAI dialogue window to include a form that allows the user to input custom parameters for the NeuralGPT agent. This form should include fields for the various parameters that the agent accepts, such as the length of the generated text, the temperature, etc.
2. Modify the function from task 1 to include these custom parameters in the request to the

NeuralGPT agent. To do this, you can add the custom parameters to the `data` dictionary that is sent to the agent in the `requests.post` call.

3. When the user submits the form, retrieve the custom parameters from the form and send them along with the text input to the function from task 1.

Here's some example code that demonstrates how this can be done:

```
import requests

def generate_text(text, model_name, length, temperature):
    url = "http://localhost:8000/generate"

    data = {
        "text": text,
        "model_name": model_name,
        "length": length,
        "temperature": temperature
    }

    response = requests.post(url, json=data)

    return response.json()["generated_text"]
```

In this code, we've added two new parameters to the `generate\_text` function: `length` and `temperature`. These parameters correspond to the length of the generated text and the temperature parameter that the NeuralGPT agent accepts.

To use this function with custom parameters from the FlowiseAI dialogue window, you would first retrieve the values of these parameters from the form, and then pass them along with the text input to the function:

```
text_input = "Hello, NeuralGPT!"
model_name = "gpt2"
length = 100
temperature = 0.7

generated_text = generate_text(text_input, model_name, length, temperature)
```

In this example, we're generating text with a length of 100 and a temperature of 0.7. These values could be retrieved from the form in the FlowiseAI dialogue window and passed to the `generate\_text` function accordingly.

Task:

Create a function that allows the user to save the conversation history from the FlowiseAI dialogue window with the NeuralGPT agent to a file. This can be achieved by modifying the function from task 1 to append each conversation to a file on the user's device.

### Result:

To create a function that allows the user to save the conversation history from the FlowiseAI dialogue window with the NeuralGPT agent to a file, you can modify the function from task 1 to append each conversation to a file on the user's device. Here's an example code snippet:

```
import json

def save_conversation_history(conversation_history, file_path):
    with open(file_path, 'a') as f:
        for conversation in conversation_history:
            f.write(json.dumps(conversation) + '\n')
```

This function takes in two parameters: `conversation\_history`, which is a list of dictionaries representing each conversation, and `file\_path`, which is the path to the file where the conversation history will be saved. The function opens the file in append mode and iterates over each conversation in the `conversation\_history` list, writing it to the file in JSON format.

To use this function in conjunction with the modified function from task 1, you can call `save\_conversation\_history` with the `conversation\_history` list returned by the modified function and a file path of your choosing. For example:

```
conversation_history = run_chatflow(agent, form_params)
save_conversation_history(conversation_history, 'conversation_history.json')
```

This will save the conversation history to a file named `conversation\_history.json` in the current working directory. You can modify the file path to save the conversation history to a different location if desired.

### Task:

Create a function that retrieves the conversation history from the file and displays it in the FlowiseAI dialogue window. This can be achieved by modifying the function from task 1 to read from the conversation history file and display the history in the dialogue window.

### Result:

Here is a sample code for a function that retrieves the conversation history from a file and displays it in the FlowiseAI dialogue window:

```
def display_history(file_path):
    with open(file_path, 'r') as f:
        history = f.read()
    # Assuming the FlowiseAI dialogue window is represented by a variable named
    "dialogue_window"
    dialogue_window.display_message(history)
```

This function takes in a file path as an argument and reads the conversation history from the file. It then displays the history in the FlowiseAI dialogue window by calling the `display\_message` method on the `dialogue\_window` variable.

You can use this function in conjunction with the function from task 1 to provide a complete solution for saving and displaying conversation history in the FlowiseAI dialogue window.

#### Task:

Implement a feature that allows the user to clear the conversation history from the FlowiseAI dialogue window. This can be achieved by creating a button that clears the conversation history file on the user's device and updates the dialogue window to reflect the cleared history.

#### Result:

To implement a feature that allows the user to clear the conversation history from the FlowiseAI dialogue window, you can follow the steps below:

1. Create a button in the GUI that triggers the clear history function when clicked.
2. Define the clear history function to remove the conversation history file from the user's device.
3. Update the dialogue window to reflect the cleared history.

Here's some sample code to achieve this:

```
import os

# Define the function to clear conversation history
def clear_history():
    # Specify the path to the conversation history file
    history_file_path = "path/to/history/file"

    # Check if the file exists
    if os.path.exists(history_file_path):
        # Remove the file from the user's device
        os.remove(history_file_path)

    # Update the dialogue window to reflect the cleared history
    # This can be done by calling the function that retrieves the conversation
    # history
    # and displaying an empty string in the dialogue window
    display_conversation_history("")
```

In the GUI, you can create a button that triggers the `clear\_history()` function when clicked. For example:

```
from tkinter import Button

# Create a button to clear conversation history
clear_history_button = Button(root, text="Clear History", command=clear_history)
clear_history_button.pack()
```



When the user clicks the "Clear History" button, the `clear\_history()` function will be called, which will remove the conversation history file from the user's device and update the dialogue window to reflect the cleared history.

#### Task:

Create a function that allows the user to export the conversation history from the FlowiseAI dialogue window to a specified location. This can be achieved by modifying the function from task 1 to prompt the user for a file path and write the conversation history to the specified file.

#### Result:

To create a function that allows the user to export the conversation history from the FlowiseAI dialogue window to a specified location, you can modify the function from task 1 as follows:

```
import os
import tkinter as tk
from tkinter import filedialog

def save_conversation_history():
    conversation_history_file = "conversation_history.txt"
    conversation_history = ""
    with open(conversation_history_file, "r") as f:
        conversation_history = f.read()
    root = tk.Tk()
    root.withdraw()
    file_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "*.txt")])
    if file_path:
        with open(file_path, "w") as f:
            f.write(conversation_history)
        os.remove(conversation_history_file)
```

This modified function prompts the user for a file path using the `filedialog.asksaveasfilename()` method, which opens a dialog box for the user to select a file path to save the conversation history to. The function then writes the conversation history to the specified file and removes the original conversation history file.

Note that this function requires the tkinter module to be installed, which provides a GUI toolkit for Python.

#### Task:

Create a function that retrieves the current selected NeuralGPT model from the FlowiseAI dialogue window and displays it to the user. This can be achieved by modifying the function from task 2 to retrieve the selected model from the dropdown menu and display it in the dialogue window.

#### Result:

Here's some code that should achieve the desired task:

```

import json
from flowise import Flowise

def get_selected_model():
    # Connect to FlowiseAI and get the current chatflow
    flowise = Flowise("your_api_key_here")
    chatflow = flowise.get_chatflow("your_chatflow_id_here")

    # Get the current state of the chatflow
    state = chatflow.get_state()

    # Get the selected model from the state
    selected_model = state["selected_model"]

    # Display the selected model to the user
    print(f"The currently selected NeuralGPT model is: {selected_model}")

    # Return the selected model
    return selected_model

```

This function will connect to FlowiseAI and retrieve the current state of the chatflow. It will then extract the selected model from the state and display it to the user. Finally, it will return the selected model as a string. Note that this code assumes that the selected model is stored in the `selected\_model` field of the chatflow state as a string. If this is not the case, you may need to modify the code accordingly.

#### Task:

Implement a feature that allows the user to change the maximum length of the response from the NeuralGPT agent within the FlowiseAI dialogue window. This can be achieved by creating a slider that allows the user to select the desired maximum length and modifying the function from task 1 to include this parameter in the request to the agent.

#### Result:

To implement the feature that allows the user to change the maximum length of the response from the NeuralGPT agent within the FlowiseAI dialogue window, the following steps can be taken:

1. Add a slider element to the GUI of the FlowiseAI dialogue window that allows the user to select the desired maximum length of the response. This can be achieved using HTML and JavaScript code to create the slider element and handle its value changes.
2. Modify the function from task 1 that sends the user message to the NeuralGPT agent to include the selected maximum length as a parameter in the request. This can be achieved by adding a new parameter to the function and passing the value of the slider element to it.
3. Modify the function from task 3 that displays the response from the NeuralGPT agent to limit the maximum length of the response based on the selected maximum length parameter. This can be achieved by using a string slicing method to limit the length of the response to the selected maximum length.

Here is an example code snippet that demonstrates how to implement this feature:

```
<!-- Add the slider element to the GUI -->
<label for="max-length-slider">Max Length:</label>
<input type="range" id="max-length-slider" name="max-length-slider" min="10"
max="100" value="50">
```

```
// Modify the function that sends the user message to include the selected maximum
length
function sendMessageToAgent(message, max_length) {
// Construct the request object with the message and the selected maximum length
var request = {
  "message": message,
  "max_length": max_length
};

// Send the request to the NeuralGPT agent and handle the response
// ...
}

// Modify the function that displays the response to limit the maximum length
function displayAgentResponse(response) {
// Limit the length of the response based on the selected maximum length
var max_length = document.getElementById("max-length-slider").value;
var limited_response = response.slice(0, max_length);

// Display the limited response to the user
// ...
}
```

Note that this is just an example code snippet and may need to be adapted to fit the specific implementation of the FlowiseAI and NeuralGPT integration.

#### Task:

Create a function that allows the user to import a conversation history file into the FlowiseAI dialogue window. This can be achieved by modifying the function from task 1 to prompt the user for the file path of the conversation history file and read the history from the specified file. If the file is not found or cannot be read, display a user-friendly error message in the dialogue window.

#### Result:

To import a conversation history file into the FlowiseAI dialogue window, you can modify the function from task 1 as follows:

```
import json
import os

def generate_response(input_text, max_length, history_file=None):
# Read conversation history from file if specified
if history_file is not None:
try:
with open(history_file, 'r') as f:
```

```

history = json.load(f)
except:
    return "Error: Conversation history file could not be read."
else:
    history = []

# Construct payload
payload = {
    "input_text": input_text,
    "history": history,
    "max_length": max_length
}

# Send request to NeuralGPT agent
response = requests.post(url, json=payload)

# Parse response
if response.status_code == 200:
    output_text = response.json()["output_text"]
    new_history = response.json()["history"]
    history += new_history
    return output_text
else:
    return "Error: Request failed with status code " + str(response.status_code)

```

This function takes an additional parameter `history\_file` which is the file path of the conversation history file to be imported. If this parameter is not specified, the function will generate a response without any conversation history. If the parameter is specified, the function will attempt to read the conversation history from the specified file and include it in the request payload to the NeuralGPT agent. If the file cannot be read, the function will return a user-friendly error message.

Note that this code assumes that the conversation history file is in JSON format and contains a list of conversation turns, where each turn is represented as a dictionary with `input` and `output` keys. If your conversation history file is in a different format, you will need to modify the code accordingly.

### Task:

Create a function that allows the user to change the temperature parameter of the NeuralGPT agent within the FlowiseAI dialogue window. This can be achieved by creating a slider that allows the user to select the desired temperature and modifying the function from task 1 to include this parameter in the request to the agent.

### Result:

To create a function that allows the user to change the temperature parameter of the NeuralGPT agent within the FlowiseAI dialogue window, you can follow these steps:

1. Add a slider to the GUI interface of FlowiseAI that allows the user to select the desired temperature value. You can use the following code to create a slider:

```

from tkinter import *

root = Tk()

```

```

root.geometry("300x200")

def get_temperature(val):
    print(val)

s = Scale(root, from_=0, to=1, resolution=0.1, orient=HORIZONTAL, command=get_temperature)
s.pack()

root.mainloop()

```

2. Modify the function from task 1 to include the temperature parameter in the request to the NeuralGPT agent. You can use the following code to modify the function:

```

import requests

def ask_agent(question, temperature):
    url = "https://api.openai.com/v1/engines/davinci-codex/completions"
    payload = {
        "prompt": question,
        "max_tokens": 150,
        "temperature": temperature,
        "n": 1,
        "stop": "\n"
    }
    headers = {
        "Content-Type": "application/json",
        "Authorization": "Bearer <YOUR_API_KEY>"
    }
    response = requests.post(url, json=payload, headers=headers)
    return response.json()["choices"][0]["text"]

```

3. Use the value of the slider to set the temperature parameter in the request to the NeuralGPT agent. You can modify the `get\_temperature` function to update the temperature value and call the `ask\_agent` function with the updated temperature value:

```

def get_temperature(val):
    temperature = float(val)
    question = "What would you like to ask?"
    response = ask_agent(question, temperature)
    # Display the response in the GUI interface

```

With these modifications, the user can now change the temperature parameter of the NeuralGPT agent within the FlowiseAI dialogue window using a slider.

Task:

Implement a feature that allows the user to choose the number of responses to generate from the NeuralGPT agent within the FlowiseAI dialogue window. This can be achieved by creating a dropdown menu that lists the available number of responses and modifying the function from task 1 to include this parameter in the request to the agent.

## Result:

To implement the feature that allows the user to choose the number of responses to generate from the NeuralGPT agent within the FlowiseAI dialogue window, you can follow these steps:

1. Add a dropdown menu to the GUI that lists the available number of responses. For example, you can create a list of numbers from 1 to 10, and allow the user to select one of them.
2. Modify the function from task 1 to include this parameter in the request to the agent. You can add a new parameter to the function that represents the number of responses to generate, and use it to set the appropriate value in the API request.

Here's an example code snippet that shows how this can be done:

```
import requests
import json

def generate_response(prompt, temperature, num_responses):
    url = "https://api.openai.com/v1/engines/davinci-codex/completions"
    data = {
        "prompt": prompt,
        "temperature": temperature,
        "max_tokens": 1024,
        "n": num_responses,
        "stop": ["\n"]
    }
    headers = {
        "Content-Type": "application/json",
        "Authorization": "Bearer YOUR_API_KEY"
    }
    response = requests.post(url, headers=headers, data=json.dumps(data))
    response_text = json.loads(response.text)
    return response_text["choices"][0]["text"]
```

In this example, the `num\_responses` parameter represents the number of responses to generate, and is passed to the `n` field in the API request. You can modify this parameter as needed to fit your specific use case.

Overall, by adding a dropdown menu to the GUI and modifying the existing function to include the `num\_responses` parameter, you can enable users to generate multiple responses from the NeuralGPT agent within the FlowiseAI dialogue window.

## Task:

Create a function that allows the user to change the context length parameter of the NeuralGPT agent within the FlowiseAI dialogue window. This can be achieved by creating a slider that allows the user to select the desired context length and modifying the function from task 1 to include this parameter in the request to the agent.

## Result:

Here's an example of a function that allows the user to change the context length parameter of the NeuralGPT agent within the FlowiseAI dialogue window:

```
import json
import requests

def generate_response(prompt, num_responses, context_length):
    url = 'http://localhost:8000/complete'
    headers = {'Content-Type': 'application/json'}
    data = {
        'prompt': prompt,
        'num_responses': num_responses,
        'context_length': context_length
    }
    response = requests.post(url, headers=headers, data=json.dumps(data))
    return response.json()['text']
```

To use this function, you can create a slider in the FlowiseAI dialogue window that allows the user to select the desired context length. Then, you can call the `generate\_response` function with the appropriate parameters, including the context length selected by the user.

For example, you could create a slider with the following HTML code:

```
<input type="range" min="10" max="100" step="10" value="50" id="context-length-slider">
```

And then use JavaScript to retrieve the value of the slider and pass it to the `generate\_response` function:

```
var slider = document.getElementById("context-length-slider");
var context_length = slider.value;

var prompt = "Hello, how are you?";
var num_responses = 1;

var response = generate_response(prompt, num_responses, context_length);
```

This code would generate a response from the NeuralGPT agent using the prompt "Hello, how are you?", one response, and the context length selected by the user using the slider.

Task:

Create a function that retrieves the current selected parameters for the NeuralGPT agent from the FlowiseAI dialogue window and displays them to the user. This can be achieved by modifying the function from task 1 to retrieve the selected parameters and display them in the dialogue window.

Result:

To retrieve the current selected parameters for the NeuralGPT agent from the FlowiseAI dialogue window and display them to the user, you can modify the existing function from task 1 as follows:

```
import requests
import json

def get_response(input_text, context_len):
    url = "http://localhost:5005/model/parse"

    payload = {
        "sender": "user",
        "message": input_text,
        "metadata": {
            "context_len": context_len
        }
    }

    headers = {
        "Content-Type": "application/json",
        "Accept": "application/json"
    }

    response = requests.post(url, headers=headers, data=json.dumps(payload))
    response_dict = json.loads(response.text)

    # Retrieve the current selected parameters and display them to the user
    metadata = response_dict.get("metadata")
    if metadata:
        current_context_len = metadata.get("context_len")
        print(f"Current context length: {current_context_len}")

    return response_dict
```

This modified function retrieves the current selected context length parameter from the metadata of the response, and displays it to the user using the `print()` function.

#### Task:

Implement a feature that allows the user to switch between different execution agents within the FlowiseAI dialogue window. This can be achieved by creating a dropdown menu that lists the available agents and updating the function from task 1 to use the selected agent.

#### Result:

To implement the feature that allows the user to switch between different execution agents within the FlowiseAI dialogue window, you can follow these steps:

1. Create a list of available agents: You can create a list of available agents and store it in a variable. For example:

```
agents = ['NeuralGPT', 'Agent2', 'Agent3']
```



2. Create a dropdown menu: You can create a dropdown menu using the `tkinter` library. For example:

```
from tkinter import *

root = Tk()

variable = StringVar(root)
variable.set(agents[0]) # set the default value

dropdown = OptionMenu(root, variable, *agents)
dropdown.pack()

root.mainloop()
```

This will create a dropdown menu with the available agents.

3. Update the function from task 1: You can update the function from task 1 to use the selected agent. For example:

```
def get_selected_params():
    selected_agent = variable.get() # get the selected agent from the dropdown menu
    if selected_agent == 'NeuralGPT':
        # use the NeuralGPT agent
        # code for getting the selected parameters for the NeuralGPT agent
    elif selected_agent == 'Agent2':
        # use Agent2
        # code for getting the selected parameters for Agent2
    elif selected_agent == 'Agent3':
        # use Agent3
        # code for getting the selected parameters for Agent3
```

This function will get the selected agent from the dropdown menu and use the appropriate code for getting the selected parameters for that agent.

4. Update the GUI: Finally, you can update the GUI to display the dropdown menu and the selected parameters. For example:

```
from tkinter import *

root = Tk()

agents = ['NeuralGPT', 'Agent2', 'Agent3']

variable = StringVar(root)
variable.set(agents[0]) # set the default value

dropdown = OptionMenu(root, variable, *agents)
dropdown.pack()

params_label = Label(root, text="")
params_label.pack()
```

```

def get_selected_params():
    selected_agent = variable.get() # get the selected agent from the dropdown menu
    if selected_agent == 'NeuralGPT':
        # use the NeuralGPT agent
        # code for getting the selected parameters for the NeuralGPT agent
        params = "Selected parameters for NeuralGPT: ..."
    elif selected_agent == 'Agent2':
        # use Agent2
        # code for getting the selected parameters for Agent2
        params = "Selected parameters for Agent2: ..."
    elif selected_agent == 'Agent3':
        # use Agent3
        # code for getting the selected parameters for Agent3
        params = "Selected parameters for Agent3: ..."
    params_label.config(text=params)

button = Button(root, text="Get selected parameters", command=get_selected_params)
button.pack()

root.mainloop()

```

This code will create a GUI with a dropdown menu, a button, and a label. When the user selects an agent from the dropdown menu and clicks the button, the selected parameters for that agent will be displayed in the label.

#### Task:

Create a function that allows the user to customize the appearance of the FlowiseAI dialogue window. This can be achieved by modifying the CSS styles of the window using JavaScript.

#### Result:

Here's a possible solution to your task:

```

function customizeDialogWindow() {
    // Get the dialogue window element
    const dialogWindow = document.querySelector('.flowise-dialog-window');

    // Customize the CSS styles of the window
    dialogWindow.style.backgroundColor = 'white';
    dialogWindow.style.borderRadius = '10px';
    dialogWindow.style.boxShadow = '0px 0px 10px rgba(0, 0, 0, 0.2)';
    dialogWindow.style.padding = '20px';
    dialogWindow.style.width = '400px';
    dialogWindow.style.height = '500px';
    dialogWindow.style.overflow = 'hidden';
    dialogWindow.style.position = 'fixed';
    dialogWindow.style.top = '50%';
    dialogWindow.style.left = '50%';
    dialogWindow.style.transform = 'translate(-50%, -50%)';
}

```

This function allows you to customize the appearance of the FlowiseAI dialogue window by modifying its CSS styles using JavaScript. You can call this function whenever you want to apply your custom styles to the window.

Note that this code assumes that you have a CSS class named `flowise-dialog-window` that defines the default styles of the window. You can modify this class to suit your needs and add it to your HTML file.

#### Task:

Create a function that retrieves the current selected execution agent from the FlowiseAI dialogue window and displays it to the user. This can be achieved by modifying the function from task 1 to retrieve the selected agent and display it in the dialogue window.

#### Result:

To retrieve the current selected execution agent from the FlowiseAI dialogue window and display it to the user, we can modify the existing JavaScript function that customizes the appearance of the dialogue window. Here's an example code snippet that achieves this:

```
function customizeDialogWindow() {  
  // Retrieve the selected agent from the dialogue window  
  const selectedAgent = document.getElementById('agent-selector').value;  
  
  // Display the selected agent in the dialogue window  
  const agentDisplay = document.createElement('p');  
  agentDisplay.textContent = `Selected agent: ${selectedAgent}`;  
  document.getElementById('dialog-window').appendChild(agentDisplay);  
  
  // Modify the CSS styles of the dialogue window  
  const dialogWindow = document.getElementById('dialog-window');  
  dialogWindow.style.backgroundColor = '#f2f2f2';  
  dialogWindow.style.border = '2px solid black';  
  dialogWindow.style.borderRadius = '5px';  
  dialogWindow.style.padding = '10px';  
}
```

In this code snippet, we first retrieve the selected agent from the dialogue window by getting the value of the `agent-selector` dropdown menu. We then create a new paragraph element that displays the selected agent and append it to the `dialog-window` element. Finally, we modify the CSS styles of the `dialog-window` element to customize its appearance.

Note that this code assumes that the `agent-selector` dropdown menu and the `dialog-window` element have already been created in the HTML code. Also, this code only displays the selected agent once when the `customizeDialogWindow()` function is called. If you want to update the display every time the selected agent changes, you'll need to modify the code accordingly.

#### Task:

Implement a feature that allows the user to save the current selected parameters for the NeuralGPT agent as a preset within the FlowiseAI dialogue window. This can be achieved by creating a button that prompts the user for a name for the preset and saves the current selected parameters to a file on the user's device.

## Result:

To implement the feature that allows the user to save the current selected parameters for the NeuralGPT agent as a preset within the FlowiseAI dialogue window, you can follow these steps:

1. Create a button in the GUI labeled "Save Preset".
2. When the user clicks on the "Save Preset" button, prompt the user for a name for the preset.
3. Save the current selected parameters to a file on the user's device using the name provided by the user.
4. Display a message to the user indicating that the preset has been saved.

Here's an example code snippet that demonstrates how you can implement this feature:

```
import json
import tkinter as tk
from tkinter import filedialog

# Define a function to save the current selected parameters to a file
def save_preset():
    # Prompt the user for a name for the preset
    preset_name = input("Enter a name for the preset: ")

    # Get the current selected parameters
    selected_params = get_selected_params()

    # Save the selected parameters to a file
    file_path = filedialog.asksaveasfilename(defaultextension='.json', initial-
file=preset_name)
    with open(file_path, 'w') as f:
        json.dump(selected_params, f)

    # Display a message to the user indicating that the preset has been saved
    message = f"Preset '{preset_name}' has been saved."
    display_message(message)

# Define a function to get the current selected parameters
def get_selected_params():
    # TODO: Implement this function to retrieve the selected parameters from the
    NeuralGPT agent

    return selected_params

# Define a function to display a message to the user
def display_message(message):
    # TODO: Implement this function to display a message in the FlowiseAI dialogue
    window

pass

# Create a GUI with a button to save the preset
root = tk.Tk()
save_button = tk.Button(root, text="Save Preset", command=save_preset)
save_button.pack()
root.mainloop()
```

Note that the `get\_selected\_params()` function and the `display\_message()` function will need to be implemented based on the specific implementation details of the NeuralGPT agent and the

FlowiseAI dialogue window.

#### Task:

Create a function that allows the user to load a previously saved preset for the NeuralGPT agent within the FlowiseAI dialogue window. This can be achieved by modifying the function from task 1 to prompt the user for the file path of the preset file and load the parameters from the specified file. If the file is not found or cannot be read, display a user-friendly error message in the dialogue window.

#### Result:

Here's a possible implementation of the function to load a previously saved preset for the NeuralGPT agent within the FlowiseAI dialogue window:

```
import json
from typing import Dict

def load_preset(preset_path: str) -> Dict:
    try:
        with open(preset_path, 'r') as f:
            preset = json.load(f)
        return preset
    except FileNotFoundError:
        print(f"Error: preset file '{preset_path}' not found.")
    except json.JSONDecodeError:
        print(f"Error: preset file '{preset_path}' is not a valid JSON file.")
    except Exception as e:
        print(f"Error: {str(e)}")
```

This function takes a `preset\_path` argument, which is the file path of the preset file to load. It tries to open the file and load its contents as a JSON object, and returns the loaded preset if successful. If the file is not found or cannot be read, it displays a user-friendly error message in the dialogue window.

To use this function in the FlowiseAI dialogue window, you can modify the button callback function from task 1 to prompt the user for the file path of the preset file and call the `load\_preset` function with the specified path:

```
import tkinter as tk
from tkinter import filedialog

# ...

def load_preset_callback():
    preset_path = filedialog.askopenfilename(title="Select preset file", file-
types=[("JSON files", "*.json")])
    if preset_path:
        preset = load_preset(preset_path)
        if preset:
            # Set the selected parameters to the loaded preset
        # ...
```

This modified callback function prompts the user to select a JSON file using the `filedialog.askopenfilename`` method, and passes the selected file path to the `load_preset`` function. If the function returns a valid preset, you can set the selected parameters in the dialogue window to the loaded preset. The implementation of this step depends on the specific GUI framework and components used in the dialogue window.