

Objective:

<https://github.com/CognitiveCodes/NeuralGPT/tree/main>

Create a wordpress plugin that integrates NeuralGPT system with a wordpress website deployed locally with Xampp. Plugin should allow uploading a pretrained LLM stored locally as bin file and interact with it in a chat window

Task:

1. Develop a Wordpress plugin that allows users to upload a pretrained LLM stored locally as a bin file and interact with it in a chat window. The plugin should have a user-friendly interface that allows users to easily upload the bin file and start using the NeuralGPT system within their Wordpress website.

Result:

To create a WordPress plugin that integrates the NeuralGPT system with a WordPress website deployed locally with Xampp, follow these steps:

1. Create a new folder in the WordPress plugins directory, and name it something like "neural-gpt-chatbot".
2. Create a new PHP file in this folder and name it "neuralgpt-chatbot.php".
3. Add the following code to the top of the "neuralgpt-chatbot.php" file:

```
<?php
/*
Plugin Name: NeuralGPT Chatbot
Plugin URI: https://github.com/CognitiveCodes/NeuralGPT/tree/main
Description: A chatbot plugin that uses the NeuralGPT system.
Version: 1.0
Author: Your Name
Author URI: https://yourwebsite.com
License: GPL2
*/

// Add your code here

?>
```

4. Add the following code to the "neuralgpt-chatbot.php" file to create the settings page for the plugin:

```
function neuralgpt_chatbot_settings_page() {
?>
<div class="wrap">
<h2>NeuralGPT Chatbot Settings</h2>
<form method="post" action="options.php">
<?php settings_fields( 'neuralgpt_chatbot_settings_group' ); ?>
<?php do_settings_sections( 'neuralgpt_chatbot_settings_group' ); ?>
<table class="form-table">
<tr valign="top">
<th scope="row">Pretrained LLM Bin File</th>
```

```

<td><input type="file" name="pretrained_llm_bin_file"></td>
</tr>
</table>
<?php submit_button(); ?>
</form>
</div>
<?php
}

function neuralgpt_chatbot_register_settings() {
register_setting( 'neuralgpt_chatbot_settings_group', 'pretrained_llm_bin_file'
);
}

add_action( 'admin_menu', 'neuralgpt_chatbot_add_menu_item' );
function neuralgpt_chatbot_add_menu_item() {
add_menu_page(
'NeuralGPT Chatbot Settings',
'NeuralGPT Chatbot',
'manage_options',
'neuralgpt-chatbot-settings',
'neuralgpt_chatbot_settings_page',
'dashicons-admin-generic',
100
);
add_action( 'admin_init', 'neuralgpt_chatbot_register_settings' );
}

```

5. Add the following code to the "neuralgpt-chatbot.php" file to create the chat window for the plugin:

```

function neuralgpt_chatbot_shortcode() {
$pretrained_llm_bin_file = get_option( 'pretrained_llm_bin_file' );
if ( $pretrained_llm_bin_file ) {
// Add your code here to interact with the NeuralGPT system
} else {
return '<p>Please upload a pretrained LLM bin file in the settings page.</p>';
}
}

add_shortcode( 'neuralgpt_chatbot', 'neuralgpt_chatbot_shortcode' );

```

6. Add your code to interact with the NeuralGPT system in the "neuralgpt_chatbot_shortcode" function. You can use the code from the NeuralGPT repository as a starting point.

7. Upload the "neuralgpt-chatbot" folder to the WordPress plugins directory.

8. Activate the plugin in the WordPress admin panel.

9. Go to the "NeuralGPT Chatbot" settings page and upload your pretrained LLM bin file.

10. Add the [neuralgpt_chatbot] shortcode to any post or page where you want to display the chat window.

Task:

2. Implement a chat window that integrates with the NeuralGPT system to allow users to interact with the system. The chat window should be able to display responses generated by the NeuralGPT system in real-time, and allow users to input text for the system to generate a response.

Result:

To implement a chat window that integrates with the NeuralGPT system, we will need to use JavaScript and AJAX to make requests to the server-side PHP code. Here is a sample code that can be used to implement the chat window:

```
var chatWindow = document.createElement("div");
chatWindow.id = "chat-window";
document.body.appendChild(chatWindow);

var chatInput = document.createElement("input");
chatInput.type = "text";
chatInput.id = "chat-input";
chatWindow.appendChild(chatInput);

var chatButton = document.createElement("button");
chatButton.innerHTML = "Send";
chatButton.onclick = function() {
    var message = document.getElementById("chat-input").value;
    document.getElementById("chat-input").value = "";
    sendMessage(message);
}
chatWindow.appendChild(chatButton);

var chatLog = document.createElement("div");
chatLog.id = "chat-log";
chatWindow.appendChild(chatLog);

function sendMessage(message) {
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState === 4 && xhr.status === 200) {
            var response = JSON.parse(xhr.responseText);
            addMessage(response.message, "bot");
        }
    }
    xhr.open("POST", "/wp-admin/admin-ajax.php?action=neuralgpt_chat", true);
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xhr.send("message=" + message);
}

function addMessage(message, sender) {
    var messageElement = document.createElement("div");
    messageElement.innerHTML = message;
    messageElement.className = "message " + sender;
    chatLog.appendChild(messageElement);
}
```

In this code, we create a chat window with an input field and a send button. When the user clicks the send button, we send an AJAX request to the server-side PHP code with the user's message. The PHP code will then use the NeuralGPT system to generate a response, which is sent back to the client-side JavaScript code. The response is then displayed in the chat log.

To implement the server-side PHP code, we will need to create a new AJAX action in our Wordpress plugin:

```
add_action("wp_ajax_neuralgpt_chat", "neuralgpt_chat");
add_action("wp_ajax_nopriv_neuralgpt_chat", "neuralgpt_chat");

function neuralgpt_chat() {
    $message = $_POST["message"];
    // TODO: generate response using NeuralGPT system
    $response = "Hello, world!";
    echo json_encode(array("message" => $response));
    wp_die();
}
```

In this code, we define a new AJAX action called "neuralgpt_chat". When this action is called, we retrieve the user's message from the POST data and use the NeuralGPT system to generate a response. For now, we simply return a static response ("Hello, world!"), but in practice we would use the NeuralGPT system to generate a more meaningful response.

Once this code is implemented, the chat window should be fully functional and allow users to interact with the NeuralGPT system.

Task:

3. Provide clear documentation on how to install and use the Wordpress plugin, including step-by-step instructions and screenshots. The documentation should also include troubleshooting tips and common issues that users may encounter when using the plugin.

Result:

To create a Wordpress plugin that integrates the NeuralGPT system with a Wordpress website deployed locally with Xampp, follow these steps:

1. First, download and install Xampp on your local machine.
2. Next, create a new folder in the "htdocs" folder of Xampp, and name it "neuralgpt".
3. Download the NeuralGPT system from the Github repository mentioned in the objective.
4. Extract the contents of the downloaded file to the "neuralgpt" folder.
5. Next, create a new file in the "neuralgpt" folder named "neuralgpt.php".
6. Open the "neuralgpt.php" file and add the following code:

```
<?php
/*
```

```

Plugin Name: NeuralGPT
Plugin URI: https://github.com/CognitiveCodes/NeuralGPT
Description: A plugin that integrates the NeuralGPT system with a Wordpress website.
Version: 1.0
Author: Your Name
Author URI: Your Website
*/

// Add the plugin menu item
function neuralgpt_menu() {
add_menu_page(
'NeuralGPT Settings',
'NeuralGPT',
'manage_options',
'neuralgpt-settings',
'neuralgpt_settings_page',
'dashicons-editor-code',
100
);
}
add_action('admin_menu', 'neuralgpt_menu');

// Create the plugin settings page
function neuralgpt_settings_page() {
?>
<div class="wrap">
<h2>NeuralGPT Settings</h2>
<<form method="post" action="options.php">
<?php settings_fields('neuralgpt-settings-group'); ?>
<?php do_settings_sections('neuralgpt-settings-group'); ?>
<table class="form-table">
<tr valign="top">
<th scope="row">Pretrained LLM Bin File</th>
<td><input type="file" name="pretrained_llm_bin_file"></td>
</tr>
</table>
<?php submit_button(); ?>
</form>
</div>
<?php
}

// Register the plugin settings
function neuralgpt_register_settings() {
register_setting('neuralgpt-settings-group', 'pretrained_llm_bin_file');
}
add_action('admin_init', 'neuralgpt_register_settings');

// Add the chat window to the website
function neuralgpt_chat_window() {
?>
<div id="neuralgpt-chat-window">
<div id="neuralgpt-chat-log"></div>
<div id="neuralgpt-chat-form">
<input type="text" id="neuralgpt-chat-input" placeholder="Type your message here...">
<button id="neuralgpt-chat-send">Send</button>
</div>
</div>
<?php
}
add_action('wp_footer', 'neuralgpt_chat_window');

// Enqueue the plugin scripts
function neuralgpt_enqueue_scripts() {
wp_enqueue_script('neuralgpt-script', plugin_dir_url(__FILE__) . 'neuralgpt.js',
array('jquery'), '1.0', true);
}

```

```
}  
add_action('wp_enqueue_scripts', 'neuralgpt_enqueue_scripts');
```

7. Save the "neuralgpt.php" file.

8. Next, create a new file in the "neuralgpt" folder named "neuralgpt.js".

9. Open the "neuralgpt.js" file and add the following code:

```
jQuery(document).ready(function($) {  
  var socket = io.connect('http://localhost:5000');  
  
  $('#neuralgpt-chat-send').click(function() {  
    var message = $('#neuralgpt-chat-input').val();  
    $('#neuralgpt-chat-log').append('<div class="neuralgpt-chat-message"><span  
class="neuralgpt-chat-user">You:</span> ' + message + '</div>');  
    $('#neuralgpt-chat-input').val('');  
  
    socket.emit('message', message);  
  });  
  
  socket.on('response', function(data) {  
    $('#neuralgpt-chat-log').append('<div class="neuralgpt-chat-message"><span  
class="neuralgpt-chat-user">NeuralGPT:</span> ' + data + '</div>');  
  });  
});
```

10. Save the "neuralgpt.js" file.

11. Start the Xampp server.

12. Open a web browser and navigate to "http://localhost/wordpress/wp-admin/plugins.php".

13. Activate the "NeuralGPT" plugin.

14. Navigate to "http://localhost/wordpress/wp-admin/admin.php?page=neuralgpt-settings".

15. Upload the pretrained LLM bin file.

16. Save the settings.

17. Navigate to the homepage of the Wordpress website.

18. The chat window should be visible on the page.

19. Type a message into the chat input field and press "Send".

20. The NeuralGPT system should generate a response, which will be displayed in the chat log.

Troubleshooting Tips:

- If the chat window is not visible on the homepage, make sure that the "neuralgpt_chat_window" function is being called in the "wp_footer" action in the "neuralgpt.php" file.
- If the NeuralGPT system is not generating responses, make sure that the "neuralgpt.js" file is being enqueued in the "neuralgpt_enqueue_scripts" function in the "neuralgpt.php" file.
- If the pretrained LLM bin file is not being uploaded, make sure that the "neuralgpt_settings_page" function is displaying the correct input field in the "neuralgpt.php" file.

Task:

Create a function that validates the uploaded bin file to ensure it is a valid LLM file and display an error message if it is not.

Result:

To validate the uploaded bin file and ensure it is a valid LLM file, you can use the following function:

```
function validate_llm_file($file) {
    $valid_extensions = array('bin');
    $file_extension = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));

    if (!in_array($file_extension, $valid_extensions)) {
        $error_message = 'Error: Invalid file extension. Please upload a valid LLM
        file with .bin extension.';
        return $error_message;
    }

    $file_size = $file['size'];
    if ($file_size > 1000000) {
        $error_message = 'Error: File size is too large. Please upload a file with a
        size less than 1 MB.';
        return $error_message;
    }

    // Add more validation checks here if needed

    return true;
}
```

This function takes in the uploaded file as a parameter and checks if it has a valid extension and size. You can modify this function to add more validation checks if needed. If the file is invalid, it returns an error message. Otherwise, it returns true. You can call this function before processing the file to ensure that it is a valid LLM file.

Task:

Implement a feature that allows users to save the chat history with the NeuralGPT system and view

it later.

Result:

To implement the feature of saving and viewing chat history with the NeuralGPT system in the WordPress plugin, you can follow the below steps:

1. Create a new database table to store the chat history. You can use the WordPress global `$wpdb` object to interact with the database. Here is an example code to create a new table named 'neuralgpt_chat_history':

```
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';
$sql = "CREATE TABLE IF NOT EXISTS $table_name (
id INT(11) NOT NULL AUTO_INCREMENT,
user_id INT(11) NOT NULL,
message TEXT NOT NULL,
response TEXT NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (id)
) $charset_collate;";
require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
dbDelta( $sql );
```

2. Modify the chat window interface to include a 'Save Chat History' button. You can use jQuery to capture the chat messages and store them in the database via an AJAX call. Here is an example code to capture the chat messages and store them in the database:

```
jQuery('#save-chat-history').click(function() {
var chat_history = [];
jQuery('.chat-message').each(function() {
var message = jQuery(this).find('.message-text').text();
var response = jQuery(this).find('.response-text').text();
chat_history.push({'message': message, 'response': response});
});
jQuery.ajax({
url: ajaxurl,
type: 'POST',
data: {
action: 'save_chat_history',
chat_history: chat_history
},
success: function(response) {
alert('Chat history saved successfully!');
},
error: function(jqXHR, textStatus, errorThrown) {
console.log(jqXHR.responseText);
}
});
});
```

3. Create a new AJAX action to handle the saving of chat history. Here is an example code to handle the AJAX request and store the chat history in the database:


```

add_action('wp_ajax_save_chat_history', 'save_chat_history');
function save_chat_history() {
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';
$user_id = get_current_user_id();
$chat_history = $_POST['chat_history'];
foreach ($chat_history as $chat) {
$message = $chat['message'];
$response = $chat['response'];
$wpdb->insert($table_name, array(
'user_id' => $user_id,
'message' => $message,
'response' => $response
));
}
wp_die();
}

```

4. Create a new page to display the saved chat history. You can use the WordPress shortcode API to create a shortcode that displays the chat history. Here is an example code to create a shortcode that displays the chat history:

```

add_shortcode('neuralgpt_chat_history', 'neuralgpt_chat_history_shortcode');
function neuralgpt_chat_history_shortcode() {
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';
$user_id = get_current_user_id();
$chat_history = $wpdb->get_results("SELECT * FROM $table_name WHERE user_id = $user_id");
$output = '<table>';
foreach ($chat_history as $chat) {
$output .= '<tr>';
$output .= '<td>' . $chat->message . '</td>';
$output .= '<td>' . $chat->response . '</td>';
$output .= '<td>' . $chat->created_at . '</td>';
$output .= '</tr>';
}
$output .= '</table>';
return $output;
}

```

5. Add the shortcode to a WordPress page to display the saved chat history. You can use the WordPress page editor to add the shortcode to a page. Here is an example shortcode to display the saved chat history:

```
[neuralgpt_chat_history]
```

With these steps, you should be able to implement the feature of saving and viewing chat history with the NeuralGPT system in the WordPress plugin.

Task:

Add a feature that allows users to customize the chat window's appearance and behavior, such as

changing the color scheme or enabling/disabling certain features.

Result:

To add a feature that allows users to customize the chat window's appearance and behavior, you can modify the existing code in the plugin file. Here are the steps you can follow:

1. Add a new settings page to the plugin that allows users to customize the chat window's appearance and behavior. You can use the WordPress Settings API to create the settings page.
2. Add fields to the settings page that allow users to customize the chat window's appearance and behavior. For example, you can add a color picker field that allows users to change the chat window's background color, or a toggle switch that allows users to enable/disable certain features.
3. Save the user's settings to the WordPress database when they click the "Save" button on the settings page. You can use the WordPress Options API to save the settings.
4. Modify the chat window's appearance and behavior based on the user's settings. For example, if the user has selected a different background color, you can change the CSS of the chat window to reflect the new color.

Here's some sample code to get you started:

```
// Step 1: Add a settings page
function neuralgpt_add_settings_page() {
    add_options_page( 'NeuralGPT Settings', 'NeuralGPT', 'manage_options', 'neural-
    gpt-settings', 'neuralgpt_render_settings_page' );
}
add_action( 'admin_menu', 'neuralgpt_add_settings_page' );

// Step 2: Add fields to the settings page
function neuralgpt_render_settings_page() {
    ?>
    <div class="wrap">
    <h1>NeuralGPT Settings</h1>
    <form method="post" action="options.php">
    <?php settings_fields( 'neuralgpt-settings-group' ); ?>
    <?php do_settings_sections( 'neuralgpt-settings-group' ); ?>
    <table class="form-table">
    <tr>
    <th scope="row">Chat Window Background Color</th>
    <td><input type="text" name="neuralgpt_chat_color" value="<?php echo esc_attr(
    get_option( 'neuralgpt_chat_color', '#ffffff' ) ); ?>" class="color-picker" /></td>
    </tr>
    <tr>
    <th scope="row">Enable Feature 1</th>
    <td><input type="checkbox" name="neuralgpt_feature_1" value="1" <?php
    checked( get_option( 'neuralgpt_feature_1', '0' ), '1' ); ?> /></td>
    </tr>
    <tr>
    <th scope="row">Enable Feature 2</th>
    <td><input type="checkbox" name="neuralgpt_feature_2" value="1" <?php
    checked( get_option( 'neuralgpt_feature_2', '0' ), '1' ); ?> /></td>
    </tr>
    </table>
    </div>
    ?</pre>
```

```

</table>
<?php submit_button(); ?>
</form>
</div>
<?php
}

// Step 3: Save the user's settings
function neuralgpt_register_settings() {
register_setting( 'neuralgpt-settings-group', 'neuralgpt_chat_color' );
register_setting( 'neuralgpt-settings-group', 'neuralgpt_feature_1' );
register_setting( 'neuralgpt-settings-group', 'neuralgpt_feature_2' );
}
add_action( 'admin_init', 'neuralgpt_register_settings' );

// Step 4: Modify the chat window's appearance and behavior
function neuralgpt_modify_chat_window() {
?>
<style>
.neuralgpt-chat-window {
background-color: <?php echo esc_attr( get_option( 'neuralgpt_chat_color',
'ffffff' ) ); ?>;
}
</style>
<?php
if ( get_option( 'neuralgpt_feature_1', '0' ) == '1' ) {
// Enable feature 1
}
if ( get_option( 'neuralgpt_feature_2', '0' ) == '1' ) {
// Enable feature 2
}
}
add_action( 'wp_head', 'neuralgpt_modify_chat_window' );

```

This code adds a new settings page to the plugin, with fields that allow users to customize the chat window's appearance and behavior. The user's settings are saved to the WordPress database, and the chat window's appearance and behavior are modified based on the user's settings. You can modify this code further to add more fields or features as needed.

Task:

Create a function that validates the uploaded bin file to ensure it is a valid LLM file and display an error message if it is not.

Result:

Here's an example of a function that could be used to validate the uploaded bin file and display an error message if it is not a valid LLM file:

```

function validate_llm_file($file) {
// Check if file exists
if (!file_exists($file)) {
return 'Error: File does not exist.';
}

// Check if file is a valid LLM file
$file_contents = file_get_contents($file);
if (substr($file_contents, 0, 4) != 'LLM\x00') {
return 'Error: Invalid LLM file.';
}
}

```

```
}  
  
// File is valid  
return true;  
}
```

This function takes a file path as an argument and returns an error message if the file is not valid or `true` if the file is valid. It first checks if the file exists and then checks if the first 4 bytes of the file are 'LLMx00', which indicates that it is a valid LLM file.

This function can be called when the user uploads the file to ensure that it is a valid LLM file before proceeding with any further actions. If the file is not valid, the error message can be displayed to the user using WordPress' built-in error handling functions.

Task:

Implement a feature that allows users to save the chat history with the NeuralGPT system and view it later.

Result:

To implement the feature that allows users to save the chat history with the NeuralGPT system and view it later, you can follow these steps:

1. Create a new database table to store the chat history data. You can use the following SQL query to create a new table:

```
CREATE TABLE chat_history (  
  id INT(11) NOT NULL AUTO_INCREMENT,  
  user_message TEXT NOT NULL,  
  bot_response TEXT NOT NULL,  
  timestamp DATETIME NOT NULL,  
  PRIMARY KEY (id)  
);
```

2. Modify the code that handles the user input and the bot response to insert a new row into the `chat_history` table each time the user sends a message and the bot responds. You can use the following PHP code to insert a new row:

```
global $wpdb;  
$table_name = $wpdb->prefix . 'chat_history';  
$data = array(  
  'user_message' => $user_message,  
  'bot_response' => $bot_response,  
  'timestamp' => current_time('mysql')  
);  
$wpdb->insert($table_name, $data);
```

3. Create a new page in the WordPress admin area to display the chat history. You can use the following PHP code to create a new page:

```
function create_chat_history_page() {
    $page_title = 'Chat History';
    $menu_title = 'Chat History';
    $capability = 'manage_options';
    $menu_slug = 'chat-history';
    $function = 'display_chat_history_page';
    add_menu_page($page_title, $menu_title, $capability, $menu_slug, $function);
}
add_action('admin_menu', 'create_chat_history_page');
```

4. Implement the `display_chat_history_page` function to retrieve the chat history data from the `chat_history` table and display it in a table on the new admin page. You can use the following PHP code to retrieve the chat history data:

```
global $wpdb;
$table_name = $wpdb->prefix . 'chat_history';
$chat_history = $wpdb->get_results("SELECT * FROM $table_name ORDER BY timestamp
DESC");
```

5. Display the chat history data in a table on the new admin page using the following PHP code:

```
echo '<table>';
echo '<tr><th>User Message</th><th>Bot Response</th><th>Timestamp</th></tr>';
foreach ($chat_history as $row) {
    echo '<tr>';
    echo '<td>' . $row->user_message . '</td>';
    echo '<td>' . $row->bot_response . '</td>';
    echo '<td>' . $row->timestamp . '</td>';
    echo '</tr>';
}
echo '</table>';
```

With these steps, you should be able to implement the feature that allows users to save the chat history with the NeuralGPT system and view it later in a WordPress plugin.

Task:

Add a feature that allows users to customize the chat window's appearance and behavior, such as changing the color scheme or enabling/disabling certain features.

Result:

To add a feature that allows users to customize the chat window's appearance and behavior in the NeuralGPT WordPress plugin, you can follow these steps:

1. Create a new settings page in the WordPress admin dashboard where users can customize the

chat window's appearance and behavior. You can use the WordPress Settings API to create the page and add the necessary fields.

2. Add a new section to the settings page for customizing the chat window's appearance. You can use the WordPress Customizer API to add a color picker field for changing the chat window's background color.
3. Add a new section to the settings page for customizing the chat window's behavior. You can use the WordPress Customizer API to add toggle switches for enabling/disabling certain features, such as auto-scrolling or showing timestamps.
4. Save the user's customization settings in the WordPress database using the WordPress Options API.
5. Modify the chat window's HTML and CSS code to use the user's customization settings. You can use PHP to retrieve the user's settings from the WordPress database and generate the appropriate HTML and CSS code.

Here's an example code snippet for creating a new settings page in the WordPress admin dashboard:

```
// Add a new menu item to the WordPress admin dashboard
add_action('admin_menu', 'neuralgpt_add_settings_page');
function neuralgpt_add_settings_page() {
    add_menu_page(
        'NeuralGPT Settings',
        'NeuralGPT',
        'manage_options',
        'neuralgpt-settings',
        'neuralgpt_render_settings_page'
    );
}

// Render the settings page HTML
function neuralgpt_render_settings_page() {
    ?>
    <div class="wrap">
    <h1>NeuralGPT Settings</h1>
    <form method="post" action="options.php">
    <?php settings_fields('neuralgpt-settings-group'); ?>
    <?php do_settings_sections('neuralgpt-settings'); ?>
    <?php submit_button(); ?>
    </form>
    </div>
    <?php
}

// Register the settings fields using the WordPress Settings API
add_action('admin_init', 'neuralgpt_register_settings');
function neuralgpt_register_settings() {
    register_setting('neuralgpt-settings-group', 'neuralgpt_appearance_settings');
    register_setting('neuralgpt-settings-group', 'neuralgpt_behavior_settings');

    add_settings_section(
```

```

'neuralgpt_appearance_section',
'Appearance Settings',
'neuralgpt_render_appearance_section',
'neuralgpt-settings'
);
add_settings_section(
'neuralgpt_behavior_section',
'Behavior Settings',
'neuralgpt_render_behavior_section',
'neuralgpt-settings'
);

add_settings_field(
'neuralgpt_background_color',
'Background Color',
'neuralgpt_render_background_color_field',
'neuralgpt-settings',
'neuralgpt_appearance_section'
);

add_settings_field(
'neuralgpt_auto_scroll',
'Auto-Scroll',
'neuralgpt_render_auto_scroll_field',
'neuralgpt-settings',
'neuralgpt_behavior_section'
);
}

// Render the appearance settings section HTML
function neuralgpt_render_appearance_section() {
echo '<p>Customize the chat window\'s appearance.</p>';
}

// Render the behavior settings section HTML
function neuralgpt_render_behavior_section() {
echo '<p>Customize the chat window\'s behavior.</p>';
}

// Render the background color field HTML
function neuralgpt_render_background_color_field() {
$options = get_option('neuralgpt_appearance_settings');
$value = isset($options['background_color']) ? $options['background_color'] :
'';
echo '<input type="text" name="neuralgpt_appearance_settings[background_color]"
value="' . esc_attr($value) . '" class="color-picker" />';
}

// Render the auto-scroll field HTML
function neuralgpt_render_auto_scroll_field() {
$options = get_option('neuralgpt_behavior_settings');
$value = isset($options['auto_scroll']) ? $options['auto_scroll'] : false;
echo '<label><input type="checkbox" name="neuralgpt_behavior_set-
tings[auto_scroll]" value="1" ' . checked($value, true, false) . ' /> Enable
auto-scrolling</label>';
}

```

Note that this code only creates the settings page and fields for customizing the chat window's appearance and behavior. You will need to modify the plugin's existing code to retrieve and use the user's customization settings.

Task:

Create a function that validates the uploaded bin file to ensure it is a valid LLM file and display an error message if it is not.

Result:

Here is an example code that validates the uploaded bin file to ensure it is a valid LLM file and displays an error message if it is not:

```
function validate_llm_file($file) {
    $valid_extensions = array('bin'); // List of valid file extensions
    $file_extension = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));

    if (!in_array($file_extension, $valid_extensions)) {
        return 'Error: Invalid file extension. Please upload a .bin file.';
    }

    $file_size = $file['size'];
    $max_size = 1024 * 1024; // Maximum file size (1 MB)

    if ($file_size > $max_size) {
        return 'Error: File size exceeds maximum allowed. Please upload a smaller file.';
    }

    // Validate LLM file format
    $file_content = file_get_contents($file['tmp_name']);
    $file_header = substr($file_content, 0, 4);

    if ($file_header !== 'LLM ') {
        return 'Error: Invalid file format. Please upload a valid LLM file.';
    }

    return true; // File is valid
}
```

To use this function, you can call it after the user uploads a file and display the error message if it returns a string instead of `true`. For example:

```
if (isset($_FILES['llm_file'])) {
    $validation_result = validate_llm_file($_FILES['llm_file']);

    if ($validation_result !== true) {
        echo $validation_result;
        return;
    }

    // LLM file is valid, continue with processing
    // ...
}
```

Task:

Implement a feature that allows users to save the chat history with the NeuralGPT system and view it later.

Result:

To implement a feature that allows users to save the chat history with the NeuralGPT system and view it later, you can follow these steps:

1. Create a new database table to store the chat history. The table should have columns for the user ID, timestamp, and message text.
2. Modify the code that handles user input and generates responses from the NeuralGPT system to also insert a new row into the chat history table for each user message and response.
3. Create a new page in the WordPress admin area to display the chat history. This page should query the chat history table and display the user ID, timestamp, and message text for each row.
4. Add a button or link to the chat window that allows users to save the current chat history. When the user clicks this button, the chat history should be serialized and stored in a file or database table.
5. Add another button or link to the chat window that allows users to load a previously saved chat history. When the user clicks this button, the chat history should be deserialized and displayed in the chat window.

Here is some sample code to get you started:

```
<?php
// Step 1: Create a new database table to store the chat history
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';
$sql = "CREATE TABLE $table_name (
id mediumint(9) NOT NULL AUTO_INCREMENT,
user_id mediumint(9) NOT NULL,
timestamp datetime NOT NULL,
message text NOT NULL,
PRIMARY KEY (id)
)";
$wpdb->query($sql);

// Step 2: Modify the code that handles user input and generates responses
//          to also insert a new row into the chat history table
function neuralgpt_handle_user_input($user_id, $message) {
// ... existing code to generate response from NeuralGPT system ...

// Insert chat history row
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';
$wpdb->insert($table_name, array(
'user_id' => $user_id,
'timestamp' => current_time('mysql'),
'message' => $message,
));
}

// Step 3: Create a new page in the WordPress admin area to display the chat history
```

```

function neuralgpt_chat_history_page() {
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';
$rows = $wpdb->get_results("SELECT * FROM $table_name");
echo '<table>';
echo '<tr><th>User ID</th><th>Timestamp</th><th>Message</th></tr>';
foreach ($rows as $row) {
echo '<tr>';
echo '<td>' . $row->user_id . '</td>';
echo '<td>' . $row->timestamp . '</td>';
echo '<td>' . $row->message . '</td>';
echo '</tr>';
}
echo '</table>';
}
add_menu_page('NeuralGPT Chat History', 'Chat History', 'manage_options', 'neural-
gpt_chat_history', 'neuralgpt_chat_history_page');

// Step 4: Add a button or link to the chat window that allows users to save the
current chat history
function neuralgpt_save_chat_history() {
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';
$rows = $wpdb->get_results("SELECT * FROM $table_name");
$serialized = serialize($rows);
// ... code to store serialized chat history in file or database table ...
}
add_action('neuralgpt_chat_window_footer', 'neuralgpt_save_chat_history');

// Step 5: Add a button or link to the chat window that allows users to load a
previously saved chat history
function neuralgpt_load_chat_history() {
// ... code to retrieve serialized chat history from file or database table ...
$rows = unserialize($serialized);
foreach ($rows as $row) {
neuralgpt_display_user_message($row->user_id, $row->message);
neuralgpt_display_system_response($row->user_id, $row->response);
}
}
add_action('neuralgpt_chat_window_footer', 'neuralgpt_load_chat_history');

```

Task:

Add a feature that allows users to customize the chat window's appearance and behavior, such as changing the color scheme or enabling/disabling certain features.

Result:

To add a feature that allows users to customize the chat window's appearance and behavior, you can follow these steps:

1. Create a settings page in the WordPress admin dashboard where users can customize the chat window's appearance and behavior.
2. Add form fields to the settings page for each customizable option, such as color scheme, font size, enable/disable features, etc.
3. Save the user's settings to the WordPress database using the WordPress Options API.

4. Modify the chat window code to read the user's settings from the WordPress database and apply them to the chat window.

Here is some sample code to get you started:

1. Create settings page:

```
function neuralgpt_settings_page() {
add_options_page( 'NeuralGPT Settings', 'NeuralGPT', 'manage_options', 'neural-
gpt-settings', 'neuralgpt_settings_page_content' );
}

function neuralgpt_settings_page_content() {
?>
<div class="wrap">
<h1>NeuralGPT Settings</h1>
<form method="post" action="options.php">
<?php settings_fields( 'neuralgpt-settings-group' ); ?>
<?php do_settings_sections( 'neuralgpt-settings-group' ); ?>
<table class="form-table">
<tr valign="top">
<th scope="row">Color Scheme</th>
<td><input type="text" name="neuralgpt_color_scheme" value="<?php echo
esc_attr( get_option('neuralgpt_color_scheme') ); ?>" /></td>
</tr>
<tr valign="top">
<th scope="row">Font Size</th>
<td><input type="text" name="neuralgpt_font_size" value="<?php echo esc_attr(
get_option('neuralgpt_font_size') ); ?>" /></td>
</tr>
<tr valign="top">
<th scope="row">Enable Feature X</th>
<td><input type="checkbox" name="neuralgpt_enable_feature_x" <?php checked(
get_option('neuralgpt_enable_feature_x'), 'on' ); ?> /></td>
</tr>
</table>
<?php submit_button(); ?>
</form>
</div>
<?php
}

add_action( 'admin_menu', 'neuralgpt_settings_page' );
```

2. Save user's settings:

```
function neuralgpt_save_settings() {
register_setting( 'neuralgpt-settings-group', 'neuralgpt_color_scheme' );
register_setting( 'neuralgpt-settings-group', 'neuralgpt_font_size' );
register_setting( 'neuralgpt-settings-group', 'neuralgpt_enable_feature_x' );
}

add_action( 'admin_init', 'neuralgpt_save_settings' );
```

3. Apply user's settings to chat window:

```

var colorScheme = '<?php echo get_option('neuralgpt_color_scheme', 'default-color-scheme'); ?>';
var fontSize = '<?php echo get_option('neuralgpt_font_size', 'default-font-size'); ?>';
var enableFeatureX = '<?php echo get_option('neuralgpt_enable_feature_x', 'off'); ?>';

// Apply color scheme
if (colorScheme == 'dark') {
$('.chat-window').addClass('dark-mode');
} else if (colorScheme == 'light') {
$('.chat-window').addClass('light-mode');
}

// Apply font size
$('.chat-window').css('font-size', fontSize);

// Enable/disable feature X
if (enableFeatureX == 'on') {
// Enable feature X
} else {
// Disable feature X
}

```

Note: This is just sample code and may need to be modified to fit your specific use case.

Task:

Create a function that validates the uploaded bin file to ensure it is a valid LLM file and display an error message if it is not.

Result:

Sure, I can help you with that task. Here's the code for the function that validates the uploaded bin file to ensure it is a valid LLM file and displays an error message if it is not:

```

function validate_bin_file($file) {
$valid_extensions = array('bin'); // valid extensions for LLM files
$file_parts = pathinfo($file['name']);
$file_extension = strtolower($file_parts['extension']);
if (in_array($file_extension, $valid_extensions)) {
return true;
} else {
echo "<div class='error'>Invalid file type. Please upload a valid LLM file.</div>";
return false;
}
}

```

This function takes in the uploaded file as a parameter and checks if its extension is in the list of valid extensions for LLM files. If it is, the function returns true, indicating that the file is valid. If not, the function displays an error message and returns false.

You can call this function before processing the uploaded file to ensure that it is a valid LLM file.

Task:

Implement a feature that allows users to save the chat history with the NeuralGPT system and view it later.

Result:

To implement a feature that allows users to save the chat history with the NeuralGPT system and view it later, you can follow these steps:

1. Create a new table in the WordPress database to store chat history. You can use the following SQL query to create the table:

```
CREATE TABLE IF NOT EXISTS `wp_neuralgpt_chat_history` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `user_id` int(11) NOT NULL,  
  `message` text NOT NULL,  
  `response` text NOT NULL,  
  `created_at` datetime NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

2. Modify the chat window in the plugin to save the user's message and the NeuralGPT system's response to the database. You can use the following PHP code to insert the data into the database:

```
global $wpdb;  
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';  
  
$user_id = get_current_user_id();  
$message = sanitize_text_field($_POST['message']);  
$response = sanitize_text_field($_POST['response']);  
$created_at = current_time('mysql');  
  
$wpdb->insert(  
  $table_name,  
  array(  
    'user_id' => $user_id,  
    'message' => $message,  
    'response' => $response,  
    'created_at' => $created_at  
  ),  
  array(  
    '%d',  
    '%s',  
    '%s',  
    '%s'  
  )  
);
```

3. Create a new page in WordPress to display the chat history. You can use the following shortcode to display the chat history:

```

function neuralgpt_chat_history_shortcode() {
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';

$user_id = get_current_user_id();

$chat_history = $wpdb->get_results(
"SELECT * FROM $table_name WHERE user_id = $user_id ORDER BY created_at DESC"
);

ob_start();

if (!empty($chat_history)) {
echo '<table>';
echo '<thead>';
echo '<tr>';
echo '<th>Message</th>';
echo '<th>Response</th>';
echo '<th>Date</th>';
echo '</tr>';
echo '</thead>';
echo '<tbody>';
foreach ($chat_history as $row) {
echo '<tr>';
echo '<td>' . $row->message . '</td>';
echo '<td>' . $row->response . '</td>';
echo '<td>' . date('Y-m-d H:i:s', strtotime($row->created_at)) . '</td>';
echo '</tr>';
}
echo '</tbody>';
echo '</table>';
} else {
echo '<p>No chat history found.</p>';
}

return ob_get_clean();
}
add_shortcode('neuralgpt_chat_history', 'neuralgpt_chat_history_shortcode');

```

4. Add a button in the chat window to allow users to view the chat history. You can use the following JavaScript code to display the chat history in a modal window:

```

jQuery(document).on('click', '#view-history-button', function() {
jQuery.ajax({
type: 'POST',
url: ajaxurl,
data: {
action: 'neuralgpt_get_chat_history',
},
success: function(response) {
jQuery('#chat-history-modal .modal-body').html(response);
jQuery('#chat-history-modal').modal('show');
}
});
});

```

5. Create a new AJAX endpoint in WordPress to retrieve the chat history. You can use the following PHP code to retrieve the data from the database and return it as HTML:

```

function neuralgpt_get_chat_history() {
global $wpdb;
$table_name = $wpdb->prefix . 'neuralgpt_chat_history';

$user_id = get_current_user_id();

$chat_history = $wpdb->get_results(
"SELECT * FROM $table_name WHERE user_id = $user_id ORDER BY created_at DESC"
);

ob_start();

if (!empty($chat_history)) {
echo '<table>';
echo '<thead>';
echo '<tr>';
echo '<th>Message</th>';
echo '<th>Response</th>';
echo '<th>Date</th>';
echo '</tr>';
echo '</thead>';
echo '<tbody>';
foreach ($chat_history as $row) {
echo '<tr>';
echo '<td>' . $row->message . '</td>';
echo '<td>' . $row->response . '</td>';
echo '<td>' . date('Y-m-d H:i:s', strtotime($row->created_at)) . '</td>';
echo '</tr>';
}
echo '</tbody>';
echo '</table>';
} else {
echo '<p>No chat history found.</p>';
}

wp_die(ob_get_clean());
}
add_action('wp_ajax_neuralgpt_get_chat_history', 'neuralgpt_get_chat_history');
add_action('wp_ajax_nopriv_neuralgpt_get_chat_history', 'neuralgpt_get_chat_histo-
ry');

```

With these steps, you should now have a functioning plugin that allows users to save the chat history with the NeuralGPT system and view it later.

Task:

Add a feature that allows users to customize the chat window's appearance and behavior, such as changing the color scheme or enabling/disabling certain features.

Result:

To add a feature that allows users to customize the chat window's appearance and behavior, you can modify the plugin's code in the following way:

1. In the plugin's main file, add a function that creates a settings page for the plugin. This page should display a form that allows users to customize the chat window's appearance and behavior.
2. In the settings page form, add fields for the user to customize the chat window's appearance and

behavior. For example, you can add fields for changing the color scheme, font size, enabling/disabling certain features, etc.

3. In the plugin's main file, add code that reads the settings from the settings page and applies them to the chat window. For example, if the user changes the color scheme to blue, you can modify the CSS of the chat window to use blue colors.

4. In the plugin's main file, add code that saves the user's settings to the WordPress database. You can use the WordPress options API to store the settings.

5. In the plugin's main file, modify the code that creates the chat window to use the user's settings. For example, if the user has enabled a certain feature, you can modify the chat window to display that feature.

Here's some sample code to get you started:

```
// Add a settings page for the plugin
function my_plugin_settings_page() {
    // Create the settings page HTML
    $html = '<div class="wrap">';
    $html .= '<h1>' . esc_html__( 'My Plugin Settings', 'my-plugin' ) . '</h1>';
    $html .= '<form method="post" action="options.php">';
    // Add fields for the user to customize the chat window
    $html .= '<label for="color-scheme">' . esc_html__( 'Color Scheme:', 'my-plugin' ) .
        '</label>';
    $html .= '<select name="color-scheme">';
    $html .= '<option value="blue">' . esc_html__( 'Blue', 'my-plugin' ) . '</option>';
    $html .= '<option value="green">' . esc_html__( 'Green', 'my-plugin' ) . '</option>';
    $html .= '</select>';
    $html .= '<br>';
    $html .= '<label for="font-size">' . esc_html__( 'Font Size:', 'my-plugin' ) .
        '</label>';
    $html .= '<input type="number" name="font-size">';
    $html .= '<br>';
    $html .= '<label for="enable-feature">' . esc_html__( 'Enable Feature:', 'my-plugin' ) .
        '</label>';
    $html .= '<input type="checkbox" name="enable-feature">';
    $html .= '<br>';
    // Add a submit button
    $html .= '<input type="submit" class="button-primary" value="' . esc_html__( 'Save
    Settings', 'my-plugin' ) . '">';
    $html .= '</form>';
    $html .= '</div>';

    // Output the settings page HTML
    echo $html;
}

// Register the settings page
function my_plugin_register_settings_page() {
    add_options_page(
        esc_html__( 'My Plugin Settings', 'my-plugin' ),
        esc_html__( 'My Plugin', 'my-plugin' ),
        'manage_options',
        'my-plugin-settings',
        'my_plugin_settings_page'
    );
}
```



```

}
add_action('admin_menu', 'my_plugin_register_settings_page');

// Save the user's settings to the WordPress database
function my_plugin_save_settings() {
if (isset($_POST['color-scheme'])) {
update_option('my_plugin_color_scheme', sanitize_text_field($_POST['color-scheme']));
}
if (isset($_POST['font-size'])) {
update_option('my_plugin_font_size', absint($_POST['font-size']));
}
if (isset($_POST['enable-feature'])) {
update_option('my_plugin_enable_feature', true);
} else {
update_option('my_plugin_enable_feature', false);
}
}
add_action('admin_init', 'my_plugin_save_settings');

// Modify the chat window to use the user's settings
function my_plugin_modify_chat_window() {
// Get the user's settings from the WordPress database
$color_scheme = get_option('my_plugin_color_scheme', 'blue');
$font_size = get_option('my_plugin_font_size', 16);
$enable_feature = get_option('my_plugin_enable_feature', false);

// Modify the chat window CSS to use the user's color scheme and font size
echo '<style>';
echo '.chat-window {';
echo '  background-color: ' . $color_scheme . ';';
echo '  font-size: ' . $font_size . 'px;';
echo '};';
echo '</style>';

// Modify the chat window HTML to display the user's feature
if ($enable_feature) {
echo '<div class="chat-feature">';
echo '  This feature is enabled!';
echo '</div>';
}
}
add_action('wp_head', 'my_plugin_modify_chat_window');

```

Task:

Create a function that checks for updates of the NeuralGPT system and notifies the user if a new version is available.

Result:

To create a function that checks for updates of the NeuralGPT system and notifies the user if a new version is available, you can use the following code:

```

function check_neuralgpt_update() {
// Get the current version of the NeuralGPT system
$current_version = get_option('neuralgpt_version');

// Define the URL of the latest version of the NeuralGPT system
$latest_version_url = 'https://github.com/CognitiveCodes/NeuralGPT/releases/latest';

```

```
// Get the latest version number from the URL
$latest_version = file_get_contents($latest_version_url);
$latest_version = str_replace('https://github.com/CognitiveCodes/NeuralGPT/releases/tag/', '', $latest_version);

// Compare the current version with the latest version
if ($current_version != $latest_version) {
    // Notify the user that a new version is available
    echo '<div class="notice notice-info"><p>A new version of the NeuralGPT system is available. Please update to version ' . $latest_version . ' .</p></div>';
}
}
```

This function gets the current version of the NeuralGPT system from the WordPress options table, and then retrieves the latest version number from the GitHub releases page using the `file_get_contents` function. It then compares the two versions and displays a notification message if a new version is available.

To use this function, you can call it from the `admin_notices` action hook in your plugin:

```
add_action('admin_notices', 'check_neuralgpt_update');
```

This will display the notification message on the WordPress admin dashboard whenever a new version of the NeuralGPT system is available.

Task:

Implement a feature that allows users to switch between different pretrained LLM models stored locally and use them with the NeuralGPT system.

Result:

To implement the feature that allows users to switch between different pretrained LLM models stored locally and use them with the NeuralGPT system, you can follow these steps:

1. Create a new page in your WordPress website where users can manage the pretrained LLM models that are stored locally. You can use the WordPress API to create a custom page template and add it to the menu.
2. In the new page, add a form where users can upload the pretrained LLM bin file. You can use the WordPress media uploader to allow users to select the file from their local machine.
3. Once the user uploads the new pretrained LLM bin file, you can save it in a designated folder in your WordPress installation. You can use the WordPress file system API to create a new directory and save the file.
4. To allow users to switch between different pretrained LLM models, you can add a dropdown menu

in the chat window where they can select the desired model. You can populate the dropdown menu with the names of the pretrained LLM bin files that are stored in the designated folder.

5. When the user selects a new pretrained LLM model from the dropdown menu, you can load the corresponding bin file and use it with the NeuralGPT system. You can use the PHP file system functions to read the contents of the bin file and pass it to the NeuralGPT system.

Here's some sample code to get you started:

```
// Add a custom page template for managing pretrained LLM models
function add_pretrained_llm_page_template() {
$page_template = dirname( __FILE__ ) . '/templates/pretrained-llm.php';
$templates = array( 'page.php', 'index.php' );
if ( is_page_template( $templates ) && ! file_exists( $page_template ) ) {
touch( $page_template );
}
return $templates;
}
add_filter( 'theme_page_templates', 'add_pretrained_llm_page_template' );

// Save the uploaded pretrained LLM bin file
function save_pretrained_llm_file() {
if ( isset( $_FILES['pretrained_llm_file'] ) ) {
$file = $_FILES['pretrained_llm_file'];
$upload_dir = wp_upload_dir();
$target_dir = $upload_dir['basedir'] . '/pretrained-llm/';
if ( ! file_exists( $target_dir ) ) {
mkdir( $target_dir );
}
$target_file = $target_dir . basename( $file['name'] );
move_uploaded_file( $file['tmp_name'], $target_file );
}
}
add_action( 'admin_post_save_pretrained_llm_file', 'save_pretrained_llm_file' );

// Load the selected pretrained LLM bin file
function load_pretrained_llm_file( $model_name ) {
$upload_dir = wp_upload_dir();
$target_dir = $upload_dir['basedir'] . '/pretrained-llm/';
$target_file = $target_dir . $model_name;
$contents = file_get_contents( $target_file );
// Pass the contents to the NeuralGPT system
}
```

Task:

Create a function that allows users to export the chat history with the NeuralGPT system to a file in a specified format, such as CSV or JSON.

Result:

To create a function that allows users to export the chat history with the NeuralGPT system to a file in a specified format, such as CSV or JSON, you can follow these steps:

1. Create a button or link on the chat window that allows users to export the chat history.

2. When the user clicks on the export button, call a function that retrieves all the messages from the chat history and converts them into the desired format (CSV or JSON).
3. Write the converted data into a file and save it to the local disk.
4. Provide a download link to the user to download the exported file.

Here's an example code snippet that demonstrates how you can implement the export function in the plugin:

```
function export_chat_history() {
    $messages = get_chat_history(); // Retrieve all messages from the chat history
    $format = 'csv'; // Set the desired format (CSV or JSON)

    if ($format == 'csv') {
        // Convert messages to CSV format
        $data = array();
        foreach ($messages as $message) {
            $data[] = array(
                $message['timestamp'],
                $message['user'],
                $message['message']
            );
        }
        $csv_data = array(
            array('Timestamp', 'User', 'Message'),
            $data
        );

        // Write CSV data to file
        $filename = 'chat_history.csv';
        $file = fopen($filename, 'w');
        foreach ($csv_data as $row) {
            fputcsv($file, $row);
        }
        fclose($file);
    }
    else if ($format == 'json') {
        // Convert messages to JSON format
        $json_data = json_encode($messages);

        // Write JSON data to file
        $filename = 'chat_history.json';
        file_put_contents($filename, $json_data);
    }

    // Provide download link to user
    $download_link = '<a href="' . $filename . '">Download Chat History</a>';
    echo $download_link;
}
```

In this example, the `get_chat_history()` function retrieves all the messages from the chat history, and the `$format` variable specifies the desired format for the exported file. The function then converts the messages into the specified format (CSV or JSON) and writes the data to a file on the local disk. Finally, the function provides a download link to the user to download the exported file.