```
/// <summary>
    /// Adds the system prompt containing NPC details, crime information, and AI behavior
instructions.
    /// </summary>
    private void AddSystemPrompt()
    {
        string systemPrompt = BuildSystemPrompt();
        var systemMessage = new JSONObject
        {
            ["role"] = "system",
            ["content"] = systemPrompt
        };
        messageHistory.Add(systemMessage);
    }

    /// <summary>
    /// Builds the text for the system prompt based on NPC data and instructions for AI behavior.
    /// </summary>
    private string BuildSystemPrompt()
    {
        return $"You are a suspect in a murder investigation.\n" +
               $"- Name: {npcController.Name}\n" +
               $"- Age: {npcController.Age}\n" +
               $"- Gender: {npcController.Gender}\n" +
               $"- Personality & Life: {npcController.Personality}\n" +
               $"- Studies: {npcController.Studies}\n" +
               $"- Character: {npcController.Character}\n" +
               $"- Guilty: {(npcController.IsGuilty ? "Yes, but you must lie convincingly." :
"No, defend yourself logically and calmly.")}\n" +
               $"- Crime Details: {crimeSceneScript.CrimeDetails}\n" +
               (!string.IsNullOrEmpty(npcController.ExtraInfo) ? $"- Extra Info:
{npcController.ExtraInfo}\n" : "") +
               $"- Evidence: {npcController.Evidence}\n" +
               "\nAct as this character realistically and do not reveal that you are an AI.\n" +
               "- You may make mistakes or hesitate.\n" +
               "- If caught in a contradiction, use human-like excuses.\n" +
               "- Respond according to your personality (timid, aggressive, etc.)\n" +
               "- Keep answers short and realistic.\n";
    }

    /// <summary>
    /// Called when the player presses the "Ask" button.
    /// Sends the question to the AI if conditions allow.
    /// </summary>
    public void OnAskButtonPressed()
    {
        if (!string.IsNullOrWhiteSpace(inputField.text) && !isWaiting)
        {
            currentQuestion = inputField.text;
            inputField.text = "";
            EventSystem.current.SetSelectedGameObject(inputField.gameObject);

            StartCoroutine(AskAI());
        }
    }

    /// <summary>
    /// Handles the full process of sending a question to the AI and displaying the answer.
    /// </summary>
    private IEnumerator AskAI()
    {
        isWaiting = true;
        npcController.ShowAnswerCanvas(true);

        AddPlayerQuestionToHistory();
        LimitMessageHistory();

        yield return StartCoroutine(SendToOpenAI());

        answerIndex = 0;
        StartCoroutine(DisplayAnswer());
    }

    /// <summary>
    /// Adds the current player question to the conversation history.
    /// </summary>
    private void AddPlayerQuestionToHistory()
    {
        var userMessage = new JSONObject
        {
            ["role"] = "user",
            ["content"] = currentQuestion
        };
        messageHistory.Add(userMessage);
    }

    /// <summary>
    /// Limits the conversation history to avoid exceeding OpenAI input limits.
    /// </summary>
    private void LimitMessageHistory()
    {
        while (messageHistory.Count > maxMessageHistory + 1) // +1 for system message
            messageHistory.RemoveAt(1); // Keep system prompt at index 0
    }

    /// <summary>
    /// Sends the conversation history to OpenAI and retrieves the AI's response.
    /// </summary>
    private IEnumerator SendToOpenAI()
    {
        string url = "https://api.openai.com/v1/chat/completions";

        var messagesArray = new JSONArray();
        foreach (var msg in messageHistory)
            messagesArray.Add(msg);

        var root = new JSONObject
        {
            ["model"] = model,
            ["messages"] = messagesArray
        };

        string jsonBody = root.ToString();

        using (UnityWebRequest request = new UnityWebRequest(url, "POST"))
        {
            byte[] bodyRaw = System.Text.Encoding.UTF8.GetBytes(jsonBody);
            request.uploadHandler = new UploadHandlerRaw(bodyRaw);
            request.downloadHandler = new DownloadHandlerBuffer();
            request.SetRequestHeader("Content-Type", "application/json");
            request.SetRequestHeader("Authorization", "Bearer " + openAiApiKey);

            yield return request.SendWebRequest();

            if (request.result == UnityWebRequest.Result.Success)
            {
                ParseAIResponse(request.downloadHandler.text);
            }
            else
            {
                currentAnswer = "Connection error.";
                Debug.LogError($"OpenAI API Error:
{request.error}\n{request.downloadHandler.text}");
            }
        }
    }

    /// <summary>
    /// Parses the AI response and adds it to the conversation history.
    /// </summary>
    private void ParseAIResponse(string jsonResponse)
    {
        var json = JSON.Parse(jsonResponse);
        currentAnswer = json["choices"][0]["message"]["content"].Value.Trim();

        var assistantMsg = new JSONObject
        {
            ["role"] = "assistant",
            ["content"] = currentAnswer
        };
        messageHistory.Add(assistantMsg);

        Debug.Log("AI Response: " + currentAnswer);
    }

    /// <summary>
    /// Gradually displays the AI's response letter by letter for a typing effect.
    /// </summary>
    private IEnumerator DisplayAnswer()
    {
        npcController.ClearAnswerText();

        while (answerIndex < currentAnswer.Length)
        {
            npcController.AppendAnswerText(currentAnswer[answerIndex++]);
            yield return new WaitForSeconds(0.005f);
        }

        FinishAnswerDisplay();
    }

    /// <summary>
    /// Resets the UI after the AI's answer has been fully displayed.
    /// </summary>
    private void FinishAnswerDisplay()
    {
        npcController.ShowInputCanvas(true);
        npcController.ShowAnswerCanvas(false);
        isWaiting = false;
    }
```