LMStudio Local LLM + Unity Quest 3 Comprehensive Setup Guide

This comprehensive guide will walk you through setting up **LMStudio** (a local LLM runner) and integrating it with **Unity** for **Quest 3** development. You'll learn how to run large language models locally on your PC and connect them to your VR applications for intelligent, responsive AI characters and interactions.

What You'll Build

- **Local AI Assistant:** Run LLMs without internet dependency
- **Intelligent NPCs:** Create characters that can have natural conversations
- **Smart Game Systems:** Build AI-powered game mechanics
- **Privacy-First AI:** Keep all data local on your machine

Key Benefits

- No expertise required: With LM Studio, you can run any compatible Large Language Model (LLM) from Hugging Face
- Your data stays local on your machine
- Requests and responses follow OpenAI's API format
- No recurring API costs or rate limits

System Requirements

PC Requirements (LMStudio Host)

- **OS:** Windows 10/11, macOS 10.15+, or Linux
- **CPU:** Modern multi-core processor (Intel i5/AMD Ryzen 5 or higher)
- **RAM:** 16GB minimum, 32GB recommended for larger models
- **GPU:** NVIDIA RTX 3060+ (12GB VRAM) or AMD equivalent (optional but recommended)
- **Storage:** 50GB+ free space for models
- **Network:** Local network connection

Quest 3 Requirements

- **Meta Quest 3** with developer mode enabled

- **Unity:** 2022.3 LTS or newer
- **Meta XR SDK:** Latest version
- **Local network:** Quest 3 and PC on same network

Model Size Guidelines

- **Small Models (3-7B parameters):** 4-8GB RAM, can run on CPU
- **Medium Models (13-15B parameters):** 16GB+ RAM, GPU recommended
- **Large Models (30B+ parameters):** 32GB+ RAM, high-end GPU required

---

LMStudio Installation & Setup

Step 1: Download and Install LMStudio

1. **Visit LMStudio Website:**
   - Go to https://lmstudio.ai/
   - Click **"Download LM Studio"**
   - Choose your operating system (Windows/macOS/Linux)
2. **Install LMStudio:**
   - Run the installer
   - Follow the setup wizard
   - Launch LMStudio after installation
3. **Initial Configuration:**
   - Accept the license terms
   - Choose installation directory with sufficient space
   - Configure GPU acceleration if available

Step 2: First Launch Setup

1. **Open LMStudio**
2. **Configure Settings:**
   - Go to **Settings** (gear icon)
   - Set **Download Location** for models
   - Enable **GPU Acceleration** if available
   - Configure **Memory Allocation**
3. **Network Settings:**

- Note your PC's local IP address
- Ensure Windows Firewall allows LMStudio
- Configure router if needed for network access

Model Management

Step 3: Download Your First Model

1. **Navigate to Model Search:**
   - Click on the **"Discover"** tab in LMStudio
   - Browse featured models or search for specific ones
2. **Recommended Models for Unity/Quest 3:**

   For Beginners (Fast, Low Resource):

   - **Phi-3-mini-4k-instruct** (3.8B parameters, ~2.3GB)
   - **TinyLlama-1.1B-Chat** (1.1B parameters, ~0.6GB)
3. **For Better Quality (Medium Resource):**
   - **Mistral-7B-Instruct** (7B parameters, ~4.1GB)
   - **Llama-3.2-7B-Instruct** (7B parameters, ~4.7GB)
4. **For Best Quality (High Resource):**
   - **Llama-3.1-13B-Instruct** (13B parameters, ~7.3GB)
   - **CodeLlama-13B-Instruct** (13B parameters, ~7.3GB)

Download Process:

```
 Search: "Mistral-7B-Instruct-v0.2-GGUF"
Select: TheBloke/Mistral-7B-Instruct-v0.2-GGUF
Choose: Q4_K_M.gguf (balanced quality/performance)
Click: Download
```

5.
6. **Model Storage:**
   - Models are stored in: `~/.cache/lm-studio/models/`
   - Windows: `C:\Users\[username]\.cache\lm-studio\models\`
   - macOS: `/Users/[username]/.cache/lm-studio/models/`

Step 4: Model Selection Guidelines

```
# Model Quantization Levels (Quality vs Size)
Q2_K   - Smallest, lowest quality    (~2.5GB for 7B model)
Q4_K_M - Balanced quality/performance (~4.1GB for 7B model) ⭐ RECOMMENDED
Q5_K_M - Higher quality              (~4.8GB for 7B model)
Q8_0   - Highest quality             (~7.2GB for 7B model)
```

LMStudio API Server Configuration

Step 5: Start the API Server

1. **Load a Model:**
   - Go to **"Chat"** tab in LMStudio
   - Select your downloaded model from dropdown
   - Wait for model to load (indicated by green status)
2. **Start the Local Server:**
   - Navigate to **"Developer"** tab
   - Click **"Start Server"**
   - Note the server URL (usually `http://localhost:1234`)

Configure Server Settings:

```
{
  "host": "0.0.0.0",          // Listen on all interfaces
  "port": 1234,               // Default port
  "cors": true,               // Enable CORS for web requests
  "max_tokens": 2048,         // Maximum response length
  "temperature": 0.7,         // Response creativity (0.0-1.0)
  "top_p": 0.9                // Response diversity
}
```

3.

Test the Server:

```
# Test with curl (in terminal/command prompt)
curl http://localhost:1234/v1/models

# Should return list of loaded models
```

Step 6: Enable Network Access

1. **Configure LMStudio for Network Access:**
   - In LMStudio Server settings, set host to `0.0.0.0`
   - Enable CORS support
   - Note your PC's IP address (e.g., `192.168.1.100`)

Firewall Configuration:

Windows:

```
# Allow LMStudio through Windows Firewall
# Go to: Windows Defender Firewall > Allow an app
# Add LMStudio.exe or allow port 1234
```

macOS:

```
# System Preferences > Security & Privacy > Firewall
# Add LMStudio to allowed applications
```

2.

Test Network Access:

```
# From another device on same network
curl http://[PC_IP_ADDRESS]:1234/v1/models
# Replace [PC_IP_ADDRESS] with your PC's IP (e.g., 192.168.1.100)
```

Unity Project Setup for Quest 3

Step 7: Create Unity Project

1. **Unity Hub Setup:**
   - Open Unity Hub
   - Create **"New Project"**
   - Select **"3D"** template
   - Name: `LMStudio_Quest3_Demo`
   - Unity Version: **2022.3 LTS** or newer

2. **Platform Configuration:**
    - Go to **File > Build Settings**
    - Select **"Android"** platform
    - Click **"Switch Platform"**
3. **Install Meta XR SDK:**
    - Open **Window > Package Manager**
    - Change source to **"Unity Registry"**
    - Search for **"XR Interaction Toolkit"**
    - Click **"Install"**
    - Search for **"Meta XR SDK"** (if available) or use Meta XR All-in-One SDK
4. **Alternative: Manual Meta XR SDK Installation:**
    - Download Meta XR All-in-One SDK from Meta Developer Hub
    - Import the Unity package
    - Follow Meta's setup wizard

Step 8: Unity Project Configuration

Player Settings:

```
 // File > Build Settings > Player Settings
Company Name: YourCompany
Product Name: LMStudio Quest Demo
Package Name: com.yourcompany.lmstudioquest
Minimum API Level: Android 10.0 (API Level 29)
Target API Level: Android 13.0 (API Level 33)
Scripting Backend: IL2CPP
Target Architectures: ARM64 ✓
```

1.

XR Settings:

```
 // Edit > Project Settings > XR Plug-in Management
Initialize XR on Startup: ✓
Oculus (or OpenXR): ✓

// XR Plug-in Management > Oculus
Android Settings:
- Stereo Rendering Mode: Multiview
- Low Overhead Mode: Enabled
- Protected Context: Disabled
```

2.

Graphics Settings:

```
 // Edit > Project Settings > Graphics
Graphics APIs: Vulkan, OpenGLES3
Color Space: Linear (for better visual quality)
```

3.

Network Configuration & Permissions

Step 9: Android Permissions

Create Android Manifest: Create folder: `Assets/Plugins/Android/` Create file: `AndroidManifest.xml`

```
 <?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools">

    <!-- Internet permission for API requests -->
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

    <!-- Quest-specific permissions -->
    <uses-feature android:name="android.hardware.vr.headtracking" android:required="true" />
    <uses-feature android:name="oculus.software.handtracking" android:required="false" />

    <application android:theme="@style/UnityThemeSelector"
                 android:usesCleartextTraffic="true"
                 android:networkSecurityConfig="@xml/network_security_config">

        <activity android:name="com.unity3d.player.UnityPlayerActivity"
                  android:exported="true"
                  android:launchMode="singleTask"
                  android:screenOrientation="landscape"

android:configChanges="mcc|mnc|locale|touchscreen|keyboard|keyboardHidden|navigation|orie
ntation|screenLayout|uiMode|screenSize|smallestScreenSize|fontScale|layoutDirection|density"
>
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
                <category android:name="com.oculus.intent.category.VR" />
            </intent-filter>

        </activity>

    </application>
</manifest>
```

1.

Network Security Configuration: Create folder: `Assets/Plugins/Android/res/xml/`
Create file: `network_security_config.xml`

```
 <?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">localhost</domain>
        <domain includeSubdomains="true">127.0.0.1</domain>
        <domain includeSubdomains="true">192.168.1.0/24</domain>
        <domain includeSubdomains="true">10.0.0.0/8</domain>
    </domain-config>
</network-security-config>
```

2.

Step 10: Quest 3 Network Settings

CRITICAL: Fix Quest 3 Networking Issues

The solution is in the project settings. The option to force remove internet must be disabled by default, and internet access must be set to required for web requests to work on Quest.

Oculus/Meta XR Settings:

```
 // Edit > Project Settings > Oculus
Target Devices: Quest 3 ✓

// CRITICAL SETTINGS:
Internet Access: Required ⚠️ IMPORTANT!
Hand Tracking Support: Controllers and Hands
Passthrough Support: Enabled
```

// Android Settings
Force Internet Permission: DISABLED ⚠️ CRITICAL!

1.
2. **Verify Internet Access:**
    - Build Settings > Player Settings > Publishing Settings
    - **Internet Access: Require ✓**

Unity-LMStudio Integration Methods

Method 1: Direct HTTP Integration (Recommended)

Create a custom script to communicate directly with LMStudio's API.

Method 2: LLMUnity Package

LLM for Unity enables seamless integration of Large Language Models (LLMs) within the Unity engine using the LLMUnity package built on top of llama.cpp.

Installation:

Package Manager Method:

 Window > Package Manager
+ > Add package from git URL
Enter: https://github.com/undreamai/LLMUnity.git

1.
2. **Manual Method:**
    - Download from Unity Asset Store
    - Import package into project

Method 3: Custom Unity Package

For advanced users who want full control over the integration.

Code Implementation Examples

Method 1: Direct HTTP Integration

Step 11: Core LMStudio Manager

Create `Assets/Scripts/LMStudioManager.cs`:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using Newtonsoft.Json;

[System.Serializable]
public class LMStudioRequest
{
    public List<ChatMessage> messages;
    public float temperature = 0.7f;
    public int max_tokens = 2048;
    public bool stream = false;

    public LMStudioRequest()
    {
        messages = new List<ChatMessage>();
    }
}

[System.Serializable]
public class ChatMessage
{
    public string role;    // "system", "user", or "assistant"
    public string content;

    public ChatMessage(string role, string content)
    {
        this.role = role;
        this.content = content;
    }
}

[System.Serializable]
public class LMStudioResponse
{
    public List<Choice> choices;
    public Usage usage;

    [System.Serializable]
    public class Choice
    {
        public ChatMessage message;
        public string finish_reason;
    }

    [System.Serializable]
    public class Usage
    {
        public int prompt_tokens;
        public int completion_tokens;
        public int total_tokens;
    }
}

public class LMStudioManager : MonoBehaviour
{
    [Header("LMStudio Configuration")]
    public string serverIP = "192.168.1.100"; // Your PC's IP address
    public int serverPort = 1234;
    public float requestTimeout = 30f;

    [Header("AI Character Settings")]
    public string systemPrompt = "You are a helpful AI assistant in a virtual reality game. Keep
responses conversational and engaging.";

    [Header("Debug")]
    public bool enableDebugLogging = true;

    private string apiUrl;
    private List<ChatMessage> conversationHistory;

    // Events
    public event System.Action<string> OnResponseReceived;
    public event System.Action<string> OnErrorOccurred;
    public event System.Action OnRequestStarted;

    void Start()
    {
        InitializeManager();
    }
```

```
void InitializeManager()
{
    apiUrl = $"http://{serverIP}:{serverPort}/v1/chat/completions";
    conversationHistory = new List<ChatMessage>();

    // Add system prompt
    if (!string.IsNullOrEmpty(systemPrompt))
    {
        conversationHistory.Add(new ChatMessage("system", systemPrompt));
    }

    LogDebug($"LMStudio Manager initialized. API URL: {apiUrl}");

    // Test connection
    StartCoroutine(TestConnection());
}

IEnumerator TestConnection()
{
    string testUrl = $"http://{serverIP}:{serverPort}/v1/models";

    using (UnityWebRequest request = UnityWebRequest.Get(testUrl))
    {
        request.timeout = 5; // Short timeout for connection test
        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.Success)
        {
            LogDebug("✅ Successfully connected to LMStudio server!");
        }
        else
        {
            LogDebug($"❌ Failed to connect to LMStudio server: {request.error}");
            LogDebug("🔧 Troubleshooting tips:");
            LogDebug("1. Make sure LMStudio is running and server is started");
            LogDebug("2. Check if PC firewall is blocking port 1234");
            LogDebug("3. Verify Quest 3 and PC are on same network");
            LogDebug($"4. Try accessing {testUrl} in a browser");
        }
    }
}

public void SendMessage(string userMessage)
```

```
{
    if (string.IsNullOrEmpty(userMessage))
    {
        LogDebug("Cannot send empty message");
        return;
    }

    StartCoroutine(SendMessageCoroutine(userMessage));
}

IEnumerator SendMessageCoroutine(string userMessage)
{
    OnRequestStarted?.Invoke();
    LogDebug($"Sending message: {userMessage}");

    // Add user message to conversation history
    conversationHistory.Add(new ChatMessage("user", userMessage));

    // Create request
    LMStudioRequest request = new LMStudioRequest();
    request.messages = new List<ChatMessage>(conversationHistory);
    request.temperature = 0.7f;
    request.max_tokens = 2048;
    request.stream = false;

    string jsonRequest = JsonConvert.SerializeObject(request);
    LogDebug($"Request JSON: {jsonRequest}");

    // Send HTTP request
    using (UnityWebRequest webRequest = new UnityWebRequest(apiUrl, "POST"))
    {
        byte[] jsonToSend = System.Text.Encoding.UTF8.GetBytes(jsonRequest);
        webRequest.uploadHandler = new UploadHandlerRaw(jsonToSend);
        webRequest.downloadHandler = new DownloadHandlerBuffer();
        webRequest.SetRequestHeader("Content-Type", "application/json");
        webRequest.timeout = (int)requestTimeout;

        yield return webRequest.SendWebRequest();

        if (webRequest.result == UnityWebRequest.Result.Success)
        {
            string responseJson = webRequest.downloadHandler.text;
            LogDebug($"Response received: {responseJson}");
            try
            {
                LMStudioResponse response =
JsonConvert.DeserializeObject<LMStudioResponse>(responseJson);

                if (response.choices != null && response.choices.Count > 0)
                {
                    string aiResponse = response.choices[0].message.content;

                    // Add AI response to conversation history
                    conversationHistory.Add(new ChatMessage("assistant", aiResponse));

                    // Trim conversation history if it gets too long
                    TrimConversationHistory();

                    OnResponseReceived?.Invoke(aiResponse);
                    LogDebug($"AI Response: {aiResponse}");
                }
                else
                {
                    OnErrorOccurred?.Invoke("No response from AI");
                }
            }
            catch (Exception e)
            {
                LogDebug($"Error parsing response: {e.Message}");
                OnErrorOccurred?.Invoke($"Error parsing response: {e.Message}");
            }
        }
        else
        {
            string errorMessage = $"Request failed: {webRequest.error}";
            LogDebug(errorMessage);
            LogDebug($"Response Code: {webRequest.responseCode}");
            LogDebug($"Response: {webRequest.downloadHandler.text}");
            OnErrorOccurred?.Invoke(errorMessage);
        }
    }
}

void TrimConversationHistory()
{
    // Keep conversation history manageable (last 20 messages + system prompt)
    if (conversationHistory.Count > 21)
    {
        // Keep system prompt (first message) and last 20 messages
        ChatMessage systemMessage = conversationHistory[0];
        conversationHistory.RemoveRange(1, conversationHistory.Count - 21);
        conversationHistory[0] = systemMessage;
    }
}

public void ClearConversation()
{
    conversationHistory.Clear();
    if (!string.IsNullOrEmpty(systemPrompt))
    {
        conversationHistory.Add(new ChatMessage("system", systemPrompt));
    }
    LogDebug("Conversation history cleared");
}

public void UpdateSystemPrompt(string newSystemPrompt)
{
    systemPrompt = newSystemPrompt;

    // Update or add system prompt at the beginning
    if (conversationHistory.Count > 0 && conversationHistory[0].role == "system")
    {
        conversationHistory[0].content = systemPrompt;
    }
    else
    {
        conversationHistory.Insert(0, new ChatMessage("system", systemPrompt));
    }

    LogDebug($"System prompt updated: {systemPrompt}");
}

void LogDebug(string message)
{
    if (enableDebugLogging)
    {
        Debug.Log($"[LMStudio] {message}");
    }
}

void OnDestroy()
```

```
    {
        // Clean up any ongoing requests
        StopAllCoroutines();
    }
}
```

Step 12: AI Character Controller

Create `Assets/Scripts/AICharacter.cs`:

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class AICharacter : MonoBehaviour
{
    [Header("UI References")]
    public TextMeshProUGUI dialogueText;
    public TMP_InputField inputField;
    public Button sendButton;
    public Button clearButton;
    public GameObject thinkingIndicator;

    [Header("Character Settings")]
    public string characterName = "Assistant";
    public string characterPersonality = "friendly and helpful";

    [Header("Audio (Optional)")]
    public AudioSource voiceSource;
    public AudioClip thinkingSound;

    private LMStudioManager lmStudioManager;
    private bool isProcessing = false;

    void Start()
    {
        InitializeCharacter();
        SetupUI();
    }

    void InitializeCharacter()
    {
        // Find or create LMStudio Manager
        lmStudioManager = FindObjectOfType<LMStudioManager>();
        if (lmStudioManager == null)
        {
            GameObject managerGO = new GameObject("LMStudio Manager");
            lmStudioManager = managerGO.AddComponent<LMStudioManager>();
        }

        // Set character-specific system prompt
        string systemPrompt = $"You are {characterName}, a {characterPersonality} AI character in
a VR game. " +
                            "Keep your responses natural, engaging, and in character. " +
                            "Respond as if you're talking face-to-face with the player.";

        lmStudioManager.UpdateSystemPrompt(systemPrompt);

        // Subscribe to events
        lmStudioManager.OnResponseReceived += HandleAIResponse;
        lmStudioManager.OnErrorOccurred += HandleError;
        lmStudioManager.OnRequestStarted += HandleRequestStarted;
    }

    void SetupUI()
    {
        if (sendButton != null)
        {
            sendButton.onClick.AddListener(SendMessage);
        }

        if (clearButton != null)
        {
            clearButton.onClick.AddListener(ClearConversation);
        }

        if (inputField != null)
        {
            inputField.onEndEdit.AddListener(OnInputFieldSubmit);
        }

        if (dialogueText != null)
        {
            dialogueText.text = $"{characterName}: Hello! I'm ready to chat. What would you like to
talk about?";
        }

        if (thinkingIndicator != null)
        {
            thinkingIndicator.SetActive(false);
        }
    }

    void OnInputFieldSubmit(string text)
    {
        if (Input.GetKeyDown(KeyCode.Return) || Input.GetKeyDown(KeyCode.KeypadEnter))
        {
            SendMessage();
        }
    }

    public void SendMessage()
    {
        if (isProcessing || string.IsNullOrEmpty(inputField.text))
            return;

        string userMessage = inputField.text.Trim();
        inputField.text = "";

        // Display user message
        UpdateDialogue($"You: {userMessage}");

        // Send to AI
        lmStudioManager.SendMessage(userMessage);
    }

    void HandleRequestStarted()
    {
        isProcessing = true;

        if (thinkingIndicator != null)
        {
            thinkingIndicator.SetActive(true);
        }

        if (sendButton != null)
        {
            sendButton.interactable = false;
        }

        if (voiceSource != null && thinkingSound != null)
        {
            voiceSource.PlayOneShot(thinkingSound);
        }

        UpdateDialogue($"{characterName}: *thinking...*");
    }

    void HandleAIResponse(string response)
    {
        isProcessing = false;

        if (thinkingIndicator != null)
        {
            thinkingIndicator.SetActive(false);
        }

        if (sendButton != null)
        {
            sendButton.interactable = true;
        }

        UpdateDialogue($"{characterName}: {response}");

        // Focus input field for next message
        if (inputField != null)
        {
            inputField.Select();
        }
    }

    void HandleError(string error)
    {
        isProcessing = false;

        if (thinkingIndicator != null)
        {
            thinkingIndicator.SetActive(false);
        }

        if (sendButton != null)
        {
            sendButton.interactable = true;
        }
        UpdateDialogue($"{characterName}: Sorry, I'm having trouble connecting right now.
({error})");
    }

    void UpdateDialogue(string newText)
    {
        if (dialogueText != null)
        {
            dialogueText.text = newText;

            // Auto-scroll to bottom if in a scroll view
            Canvas.ForceUpdateCanvases();
        }
    }

    public void ClearConversation()
    {
        lmStudioManager.ClearConversation();
        UpdateDialogue($"{characterName}: Conversation cleared. What would you like to talk
about?");
    }

    void OnDestroy()
    {
        // Unsubscribe from events
        if (lmStudioManager != null)
        {
            lmStudioManager.OnResponseReceived -= HandleAIResponse;
            lmStudioManager.OnErrorOccurred -= HandleError;
            lmStudioManager.OnRequestStarted -= HandleRequestStarted;
        }
    }
}
```

Step 13: Voice Input Integration (Optional)

Create `Assets/Scripts/VoiceInputManager.cs`:

```
using UnityEngine;
using System.Collections;

public class VoiceInputManager : MonoBehaviour
{
    [Header("Voice Settings")]
```

```
public KeyCode voiceActivationKey = KeyCode.Space;
public bool useVoiceActivation = true;
public float recordingTimeout = 10f;

[Header("UI Feedback")]
public GameObject recordingIndicator;
public UnityEngine.UI.Text statusText;

private bool isRecording = false;
private bool microphoneInitialized = false;
private AICharacter aiCharacter;

void Start()
{
    aiCharacter = FindObjectOfType<AICharacter>();
    InitializeMicrophone();
}

void InitializeMicrophone()
{
    if (Microphone.devices.Length > 0)
    {
        microphoneInitialized = true;
        Debug.Log($"Microphone initialized: {Microphone.devices[0]}");
    }
    else
    {
        Debug.LogWarning("No microphone devices found");
        useVoiceActivation = false;
    }
}

void Update()
{
    if (!useVoiceActivation || !microphoneInitialized)
        return;

    // Handle voice activation key
    if (Input.GetKeyDown(voiceActivationKey) && !isRecording)
    {
        StartRecording();
    }
    else if (Input.GetKeyUp(voiceActivationKey) && isRecording)
    {
        StopRecording();
    }
}

void StartRecording()
{
    if (isRecording) return;

    isRecording = true;

    if (recordingIndicator != null)
    {
        recordingIndicator.SetActive(true);
    }

    if (statusText != null)
    {
        statusText.text = "🎤 Recording...";
    }

    Debug.Log("Started voice recording");

    // Start recording timeout
    StartCoroutine(RecordingTimeoutCoroutine());
}

void StopRecording()
{
    if (!isRecording) return;

    isRecording = false;

    if (recordingIndicator != null)
    {
        recordingIndicator.SetActive(false);
    }

    if (statusText != null)
    {
        statusText.text = "🔄 Processing...";
    }

    Debug.Log("Stopped voice recording");
```

```
        // Process recorded audio (placeholder)
        ProcessRecordedAudio();
    }

    IEnumerator RecordingTimeoutCoroutine()
    {
        yield return new WaitForSeconds(recordingTimeout);

        if (isRecording)
        {
            Debug.Log("Recording timeout reached");
            StopRecording();
        }
    }

    void ProcessRecordedAudio()
    {
        // Placeholder for speech-to-text processing
        // You would integrate with a speech recognition service here

        // For demo purposes, simulate speech recognition
        StartCoroutine(SimulateVoiceProcessing());
    }

    IEnumerator SimulateVoiceProcessing()
    {
        yield return new WaitForSeconds(1f); // Simulate processing time

        // Simulated recognized text
        string[] samplePhrases = {
            "Hello, how are you today?",
            "What can you tell me about this place?",
            "I'm looking for some help",
            "What's your favorite color?",
            "Tell me a joke"
        };

        string recognizedText = samplePhrases[Random.Range(0, samplePhrases.Length)];

        if (statusText != null)
        {
            statusText.text = $"Recognized: \"{recognizedText}\"";
        }
        // Send recognized text to AI character
        if (aiCharacter != null)
        {
            // You would set the input field text and trigger send
            // aiCharacter.SendMessage(recognizedText);
        }

        // Clear status after a delay
        StartCoroutine(ClearStatusAfterDelay(3f));
    }

    IEnumerator ClearStatusAfterDelay(float delay)
    {
        yield return new WaitForSeconds(delay);

        if (statusText != null)
        {
            statusText.text = $"Press {voiceActivationKey} to talk";
        }
    }
}
```

Quest 3 Specific Configuration

Step 14: Scene Setup for Quest 3

Create the Main Scene:

```
 // Create new scene: File > New Scene
// Save as: Assets/Scenes/LMStudio_Quest3_Demo.unity
```

1.

Setup XR Rig:

```
 // Delete default Main Camera
// Add XR Rig: GameObject > XR > XR Origin (XR Rig)
// Configure XR Origin:
//   - Tracking Origin Mode: Floor
//   - Camera Y Offset: 0
```

2.

Create UI Canvas:

```
 // Add Canvas: GameObject > UI > Canvas
// Canvas settings:
//   - Render Mode: World Space
//   - Position: (0, 2, 2)
//   - Scale: (0.01, 0.01, 0.01)
//   - Width: 400, Height: 300
```

3.

Setup AI Character UI:

```
 // Add to Canvas:
// 1. Panel (Background)
// 2. Text (Title): "AI Assistant"
// 3. ScrollView > Viewport > Content > Text (Dialogue)
// 4. InputField (User Input)
// 5. Button (Send)
// 6. Button (Clear)
```

4.

Step 15: Hand Tracking Integration

Create `Assets/Scripts/HandInputManager.cs`:

```
using UnityEngine;
using UnityEngine.UI;

public class HandInputManager : MonoBehaviour
{
    [Header("Hand Tracking")]
    public Transform leftHandTransform;
    public Transform rightHandTransform;

    [Header("Virtual Keyboard")]
    public GameObject virtualKeyboard;
    public Button[] keyButtons;

    [Header("Gesture Controls")]
    public float pinchThreshold = 0.8f;
    public float pointingThreshold = 0.7f;

    private AICharacter aiCharacter;
    private TMP_InputField inputField;
    private bool isLeftPinching = false;
    private bool isRightPinching = false;

    void Start()
    {
        aiCharacter = FindObjectOfType<AICharacter>();

        if (aiCharacter != null)
        {
            inputField = aiCharacter.GetComponentInChildren<TMP_InputField>();
        }

        SetupVirtualKeyboard();
    }

    void SetupVirtualKeyboard()
    {
        if (virtualKeyboard == null) return;

        // Setup virtual keyboard buttons
        Button[] buttons = virtualKeyboard.GetComponentsInChildren<Button>();

        foreach (Button button in buttons)
        {
            string letter = button.GetComponentInChildren<Text>().text;
            button.onClick.AddListener(() => OnVirtualKeyPressed(letter));
        }
    }

    void Update()
    {
        UpdateHandTracking();
        HandleGestures();
    }

    void UpdateHandTracking()
    {
        // This would integrate with Meta XR SDK hand tracking
        // Placeholder for hand tracking data

        if (OVRInput.Get(OVRInput.Button.PrimaryHandTrigger, OVRInput.Controller.LTouch))
        {
            // Left hand trigger pressed
```

```
            HandleHandInteraction(true);
        }

        if (OVRInput.Get(OVRInput.Button.PrimaryHandTrigger, OVRInput.Controller.RTouch))
        {
            // Right hand trigger pressed
            HandleHandInteraction(false);
        }
    }

    void HandleHandInteraction(bool isLeftHand)
    {
        // Handle hand-based UI interaction
        // This would use ray casting from hand position

        Transform handTransform = isLeftHand ? leftHandTransform : rightHandTransform;

        if (handTransform != null)
        {
            // Cast ray from hand
            Ray ray = new Ray(handTransform.position, handTransform.forward);
            RaycastHit hit;

            if (Physics.Raycast(ray, out hit, 5f))
            {
                Button button = hit.collider.GetComponent<Button>();
                if (button != null)
                {
                    button.onClick.Invoke();
                }
            }
        }
    }

    void HandleGestures()
    {
        // Gesture recognition for quick actions
        // Example: Thumbs up to send message

        if (Input.GetKeyDown(KeyCode.T)) // Placeholder for gesture detection
        {
            if (aiCharacter != null)
            {
                aiCharacter.SendMessage();
            }
        }
    }

    void OnVirtualKeyPressed(string key)
    {
        if (inputField == null) return;

        switch (key.ToLower())
        {
            case "space":
                inputField.text += " ";
                break;
            case "backspace":
                if (inputField.text.Length > 0)
                {
                    inputField.text = inputField.text.Substring(0, inputField.text.Length - 1);
                }
                break;
            case "enter":
                if (aiCharacter != null)
                {
                    aiCharacter.SendMessage();
                }
                break;
            default:
                inputField.text += key;
                break;
        }
    }

    public void ToggleVirtualKeyboard()
    {
        if (virtualKeyboard != null)
        {
            virtualKeyboard.SetActive(!virtualKeyboard.activeInHierarchy);
        }
    }
}
```

Step 16: Build and Deploy

Build Settings:

```
 // File > Build Settings
Platform: Android ✓
Scenes: Add Open Scenes ✓

// Player Settings
Company Name: Your Company
Product Name: LMStudio Quest Demo
Package Name: com.yourcompany.lmstudioquest
Version: 1.0

// Publishing Settings
Configuration: Release
Scripting Backend: IL2CPP
Target Architectures: ARM64 ✓
```

1.

Quest 3 Connection:

```
 # Enable Developer Mode on Quest 3
# 1. Meta Quest app on phone
# 2. Menu > Devices > Quest 3 > Developer Mode
# 3. Enable Developer Mode

# Connect via USB cable
# Allow USB debugging when prompted on Quest 3
```

2.

Deploy to Quest 3:

```
 // In Unity Build Settings:
// 1. Click "Build and Run"
// 2. Choose APK location
// 3. Wait for build and automatic deployment
```

3.

Troubleshooting Guide

Common Issues and Solutions

Issue 1: "Cannot resolve destination host" on Quest 3

This is a common problem with UnityWebRequest on Quest 3. The "force disable Internet" option might be enabled in settings.

Solutions:

Check Internet Access Setting:

```
 // Edit > Project Settings > Oculus
// Set "Internet Access" to "Required"
// NOT "Auto" or "Not Required"
```

1.

Disable Force Remove Internet:

```
 // Edit > Project Settings > Player > Publishing Settings
// Force Internet Permission: UNCHECKED ⚠️
```

2.

Android Manifest Check:

```
 <!-- Ensure these permissions are in AndroidManifest.xml -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

3.

Issue 2: LMStudio Server Connection Failed

Symptoms:

- "Connection refused" errors
- "No route to host" errors
- Timeouts

Solutions:

Verify LMStudio Server:

```
 # Check if LMStudio server is running
# Open LMStudio > Developer tab > Start Server
# Should show "Server running on http://localhost:1234"
```

1.

Test Server Locally:

```
 # In browser or terminal on PC:
curl http://localhost:1234/v1/models

# Should return JSON with model information
```

2.

Network Configuration:

```
 # Find your PC's IP address:
# Windows: ipconfig
# macOS/Linux: ifconfig

# Test from Quest 3 browser:
# http://[PC_IP]:1234/v1/models
# Replace [PC_IP] with your actual IP
```

3.

Firewall Settings:

```
 # Windows Firewall:
# Control Panel > System and Security > Windows Defender Firewall
# Click "Allow an app or feature through Windows Defender Firewall"
# Add LMStudio or allow port 1234

# macOS:
# System Preferences > Security & Privacy > Firewall > Options
# Add LMStudio to allowed applications
```

4.

Issue 3: JSON Parsing Errors

Solutions:

Install Newtonsoft.Json:

```
 // Window > Package Manager
// Unity Registry > Search "Newtonsoft Json"
// Install "com.unity.nuget.newtonsoft-json"
```

1.

Alternative JSON Parsing:

```
 // If Newtonsoft.Json unavailable, use Unity's JsonUtility
[System.Serializable]
public class SimpleResponse
{
    public Choice[] choices;

    [System.Serializable]
    public class Choice
    {
        public Message message;
    }

    [System.Serializable]
    public class Message
    {
        public string content;
    }
}

// Parse with Unity's built-in JSON
SimpleResponse response = JsonUtility.FromJson<SimpleResponse>(jsonString);
```

2.

Issue 4: Performance Issues

Solutions:

Optimize Model Choice:

```
 # Use smaller, faster models for VR:
# - Phi-3-mini (3.8B) - Very fast
# - TinyLlama (1.1B) - Extremely fast
# - Mistral-7B Q4_K_M - Good balance
```

1.

Request Optimization:

```
 // Limit response length
```

```
request.max_tokens = 512; // Instead of 2048

// Use higher temperature for faster responses
request.temperature = 0.9f; // Instead of 0.7f

// Trim conversation history more aggressively
if (conversationHistory.Count > 10) // Instead of 20
{
   // Keep only recent messages
}
```

2.

Network Optimization:

```
 // Use shorter timeout for VR
webRequest.timeout = 15; // Instead of 30 seconds

// Implement request queuing
if (isProcessing) return; // Don't send multiple requests
```

3.

Issue 5: Hand Tracking Not Working

Solutions:

Enable Hand Tracking:

```
 // Edit > Project Settings > Oculus
// Hand Tracking Support: Controllers and Hands
```

1.

Quest 3 Settings:

```
 # On Quest 3:
# Settings > Hands and Controllers
# Hand Tracking: On
# Auto Switch: On (optional)
```

2.

Code Implementation:

```
 // Check hand tracking status
```

```
if (OVRInput.IsControllerConnected(OVRInput.Controller.LHand))
{
    // Left hand is being tracked
    Vector3 leftHandPos = OVRInput.GetLocalControllerPosition(OVRInput.Controller.LHand);
}
```

3.

Debug Tools

Network Debug Script

Create `Assets/Scripts/NetworkDebugger.cs`:

```
using UnityEngine;
using UnityEngine.Networking;
using System.Collections;

public class NetworkDebugger : MonoBehaviour
{
    [Header("Debug Settings")]
    public string targetIP = "192.168.1.100";
    public int targetPort = 1234;
    public bool runTests = true;

    void Start()
    {
        if (runTests)
        {
            StartCoroutine(RunNetworkTests());
        }
    }

    IEnumerator RunNetworkTests()
    {
        Debug.Log("🔍 Starting network diagnostic tests...");

        // Test 1: Basic connectivity
        yield return TestConnectivity();

        // Test 2: LMStudio API
        yield return TestLMStudioAPI();

        // Test 3: JSON parsing
        yield return TestJSONParsing();
        Debug.Log("✅ Network diagnostic tests completed");
    }

    IEnumerator TestConnectivity()
    {
        Debug.Log("📡 Testing basic connectivity...");

        string url = $"http://{targetIP}:{targetPort}/v1/models";

        using (UnityWebRequest request = UnityWebRequest.Get(url))
        {
            request.timeout = 5;
            yield return request.SendWebRequest();

            if (request.result == UnityWebRequest.Result.Success)
            {
                Debug.Log("✅ Basic connectivity: PASS");
                Debug.Log($"Response: {request.downloadHandler.text}");
            }
            else
            {
                Debug.LogError($"❌ Basic connectivity: FAIL - {request.error}");
                Debug.LogError($"URL tested: {url}");
                Debug.LogError("Troubleshooting steps:");
                Debug.LogError("1. Check if LMStudio server is running");
                Debug.LogError("2. Verify PC firewall settings");
                Debug.LogError("3. Confirm Quest 3 and PC are on same network");
            }
        }
    }

    IEnumerator TestLMStudioAPI()
    {
        Debug.Log("🤖 Testing LMStudio API...");

        string url = $"http://{targetIP}:{targetPort}/v1/chat/completions";

        var testRequest = new
        {
            messages = new[]
            {
                new { role = "user", content = "Hello" }
            },
            max_tokens = 50,
            temperature = 0.7f
        };

        string jsonData = JsonUtility.ToJson(testRequest);

        using (UnityWebRequest request = new UnityWebRequest(url, "POST"))
        {
            byte[] jsonBytes = System.Text.Encoding.UTF8.GetBytes(jsonData);
            request.uploadHandler = new UploadHandlerRaw(jsonBytes);
            request.downloadHandler = new DownloadHandlerBuffer();
            request.SetRequestHeader("Content-Type", "application/json");
            request.timeout = 10;

            yield return request.SendWebRequest();

            if (request.result == UnityWebRequest.Result.Success)
            {
                Debug.Log("✅ LMStudio API: PASS");
                Debug.Log($"Response: {request.downloadHandler.text}");
            }
            else
            {
                Debug.LogError($"❌ LMStudio API: FAIL - {request.error}");
                Debug.LogError($"Response code: {request.responseCode}");
                Debug.LogError($"Response: {request.downloadHandler.text}");
            }
        }
    }

    IEnumerator TestJSONParsing()
    {
        Debug.Log("📝 Testing JSON parsing...");

        string sampleJson = @"{
            ""choices"": [
                {
                    ""message"": {
                        ""content"": ""Hello, this is a test response!""
                    }
                }
            ]
        }";
```

```
        try
        {
            // Test with JsonUtility (Unity built-in)
            var response = JsonUtility.FromJson<TestResponse>(sampleJson);
            Debug.Log("✅ JSON parsing (JsonUtility): PASS");
            Debug.Log($"Parsed content: {response.choices[0].message.content}");
        }
        catch (System.Exception e)
        {
            Debug.LogError($"❌ JSON parsing: FAIL - {e.Message}");
        }

        yield return null;
    }

    [System.Serializable]
    public class TestResponse
    {
        public TestChoice[] choices;

        [System.Serializable]
        public class TestChoice
        {
            public TestMessage message;
        }

        [System.Serializable]
        public class TestMessage
        {
            public string content;
        }
    }
}
```

Performance Optimization

LMStudio Server Optimization

Model Selection for VR:

```
# Recommended models for Quest 3 integration:
```

```
# Fast Response (< 1 second):
- TinyLlama-1.1B-Chat-v1.0.Q4_K_M.gguf     (~0.6GB)
- Phi-3-mini-4k-instruct.Q4_K_M.gguf       (~2.3GB)

# Balanced (1-3 seconds):
- Mistral-7B-Instruct-v0.2.Q4_K_M.gguf     (~4.1GB)
- Llama-3.2-7B-Instruct.Q4_K_M.gguf        (~4.7GB)

# High Quality (3-10 seconds):
- Llama-3.1-13B-Instruct.Q4_K_M.gguf       (~7.3GB)
```

1.

Server Configuration:

```
{
  "n_ctx": 2048,           // Context window (lower = faster)
  "n_threads": 8,          // CPU threads (match your CPU cores)
  "n_gpu_layers": 35,      // GPU acceleration (if available)
  "temperature": 0.8,      // Higher = more creative but faster
  "top_p": 0.9,            // Response diversity
  "repeat_penalty": 1.1    // Prevent repetition
}
```

2.

Unity Optimization

Request Management:

```
public class OptimizedLMStudioManager : MonoBehaviour
{
    [Header("Performance Settings")]
    public int maxConcurrentRequests = 1;
    public float requestCooldown = 0.5f;
    public int maxConversationLength = 10; // Reduce memory usage

    private Queue<string> requestQueue = new Queue<string>();
    private bool isProcessingQueue = false;
    private float lastRequestTime = 0f;

    public void QueueMessage(string message)
    {
        if (Time.time - lastRequestTime < requestCooldown)
        {
            Debug.Log("Request too soon, queuing...");
            requestQueue.Enqueue(message);
            return;
        }

        SendMessage(message);
        lastRequestTime = Time.time;
    }

    void Update()
    {
        ProcessRequestQueue();
    }

    void ProcessRequestQueue()
    {
        if (requestQueue.Count > 0 && !isProcessingQueue &&
            Time.time - lastRequestTime >= requestCooldown)
        {
            string nextMessage = requestQueue.Dequeue();
            SendMessage(nextMessage);
            lastRequestTime = Time.time;
        }
    }
}
```

1.

Memory Management:

```
 void OptimizeConversationHistory()
{
    // Keep conversation history small for VR performance
    if (conversationHistory.Count > maxConversationLength)
    {
        // Keep system prompt and last N messages
        var systemMessage = conversationHistory[0];
        var recentMessages = conversationHistory
            .Skip(conversationHistory.Count - maxConversationLength + 1)
            .ToList();

        conversationHistory.Clear();
        conversationHistory.Add(systemMessage);
        conversationHistory.AddRange(recentMessages);
        // Force garbage collection
        System.GC.Collect();
    }
}
```

2.

Quest 3 Specific Optimizations

Frame Rate Optimization:

```
 void Start()
{
    // Target 72 FPS for Quest 3
    Application.targetFrameRate = 72;

    // Reduce quality for better performance
    QualitySettings.SetQualityLevel(2); // Medium quality

    // Disable VSync for lower latency
    QualitySettings.vSyncCount = 0;
}
```

1.

UI Optimization:

```
 [Header("VR UI Settings")]
public float uiUpdateInterval = 0.1f; // Update UI less frequently
private float lastUIUpdate = 0f;

void Update()
{
    // Only update UI periodically to save performance
    if (Time.time - lastUIUpdate > uiUpdateInterval)
    {
        UpdateUI();
        lastUIUpdate = Time.time;
    }
}
```

2.

Advanced Features

Multi-Model Support

```
public class MultiModelManager : MonoBehaviour
{
    [System.Serializable]
    public class ModelConfig
    {
        public string name;
        public string personality;
        public string systemPrompt;
        public float responseSpeed; // Expected response time
    }

    [Header("Available Models")]
    public ModelConfig[] availableModels;
    public int currentModelIndex = 0;

    public void SwitchModel(int modelIndex)
    {
        if (modelIndex >= 0 && modelIndex < availableModels.Length)
        {
            currentModelIndex = modelIndex;
            var selectedModel = availableModels[modelIndex];

            // Switch model in LMStudio (requires API call)
            StartCoroutine(SwitchModelCoroutine(selectedModel.name));

            // Update system prompt
            lmStudioManager.UpdateSystemPrompt(selectedModel.systemPrompt);

            Debug.Log($"Switched to model: {selectedModel.name}");
        }
    }

    IEnumerator SwitchModelCoroutine(string modelName)
    {
        // Implementation would depend on LMStudio's model switching API
        // This is a placeholder for the concept
        yield return null;
    }
}
```

Conversation Analytics

```
public class ConversationAnalytics: MonoBehaviour
{
    [System.Serializable]
    public class ConversationMetrics
    {
        public int totalMessages;
        public float averageResponseTime;
        public float totalConversationTime;
        public string[] commonTopics;
    }

    private ConversationMetrics metrics = new ConversationMetrics();
    private float conversationStartTime;
    private List<float> responseTimes = new List<float>();

    void Start()
    {
        conversationStartTime = Time.time;

        // Subscribe to LMStudio events
        var lmStudio = FindObjectOfType<LMStudioManager>();
        lmStudio.OnRequestStarted += OnRequestStarted;
        lmStudio.OnResponseReceived += OnResponseReceived;
    }

    private float requestStartTime;

    void OnRequestStarted()
    {
        requestStartTime = Time.time;
    }

    void OnResponseReceived(string response)
    {
        float responseTime = Time.time - requestStartTime;
        responseTimes.Add(responseTime);

        metrics.totalMessages++;
        metrics.averageResponseTime = responseTimes.Average();
        metrics.totalConversationTime = Time.time - conversationStartTime;

        Debug.Log($"Response time: {responseTime:F2}s, Average:
{metrics.averageResponseTime:F2}s");
```

```
    }

    public ConversationMetrics GetMetrics()
    {
        return metrics;
    }
}
```

Context-Aware Responses

```
public class ContextManager : MonoBehaviour
{
    [Header("Context Settings")]
    public Transform player;
    public string[] availableObjects;
    public string currentScene = "VR Laboratory";

    private LMStudioManager lmStudioManager;

    void Start()
    {
        lmStudioManager = FindObjectOfType<LMStudioManager>();
        UpdateContextualPrompt();
    }

    void UpdateContextualPrompt()
    {
        string contextInfo = GatherContextualInformation();
        string enhancedPrompt = $"You are an AI assistant in a {currentScene}. " +
                               $"Current context: {contextInfo}. " +
                               $"Be aware of the user's surroundings and reference them naturally in
conversation.";

        lmStudioManager.UpdateSystemPrompt(enhancedPrompt);
    }

    string GatherContextualInformation()
    {
        string context = "";

        // Player position context
        if (player != null)
        {
            Vector3 pos = player.position;
            context += $"Player is at position ({pos.x:F1}, {pos.y:F1}, {pos.z:F1}). ";
        }

        // Nearby objects context
        string nearbyObjects = GetNearbyObjects();
        if (!string.IsNullOrEmpty(nearbyObjects))
        {
            context += $"Nearby objects: {nearbyObjects}. ";
        }

        // Time context
        context += $"Current time: {System.DateTime.Now:HH:mm}. ";

        return context;
    }

    string GetNearbyObjects()
    {
        // Simple proximity detection
        var nearbyObjects = new List<string>();

        foreach (string objName in availableObjects)
        {
            GameObject obj = GameObject.Find(objName);
            if (obj != null && player != null)
            {
                float distance = Vector3.Distance(player.position, obj.transform.position);
                if (distance < 3f) // Within 3 meters
                {
                    nearbyObjects.Add(objName);
                }
            }
        }

        return string.Join(", ", nearbyObjects);
    }

    void Update()
    {
        // Update context periodically
        if (Time.frameCount % 300 == 0) // Every 5 seconds at 60 FPS
        {
            UpdateContextualPrompt();
        }
    }
}
```

Conclusion

You now have a complete setup for integrating LMStudio with Unity for Quest 3 development! This guide provides:

✅ **Complete LMStudio setup** with local model management ✅ **Unity Quest 3 configuration** with proper networking ✅ **Production-ready code** for AI character integration ✅ **Troubleshooting solutions** for common issues ✅ **Performance optimizations** for VR applications ✅ **Advanced features** for professional development

Next Steps

1. **Start Simple:** Begin with the basic AI character implementation
2. **Test Thoroughly:** Use the network debugging tools to ensure connectivity
3. **Optimize Performance:** Choose appropriate models for your use case
4. **Add Features:** Implement voice input, hand tracking, and context awareness
5. **Deploy and Iterate:** Test on Quest 3 and refine based on user feedback

Key Resources

- **LMStudio Documentation:** https://lmstudio.ai/docs
- **Meta Developer Hub:** https://developer.oculus.com/
- **Unity XR Documentation:** https://docs.unity3d.com/Manual/XR.html
- **LLMUnity Package:** https://github.com/undreamai/LLMUnity

Remember to always test your implementation on the actual Quest 3 device, as networking behavior can differ significantly between the Unity editor and the deployed application!

Community Support

- **Unity XR Forum:** https://forum.unity.com/forums/ar-vr-xr-discussion.80/
- **Meta Developer Community:** https://communityforums.atmeta.com/
- **LMStudio Discord:** Available through their website