AI in Wearable Headsets

From Basic LLM Integration to Advanced Agent

Zirui Xiao
Synteraction Lab
City University of Hong Kong

July 15, 2025

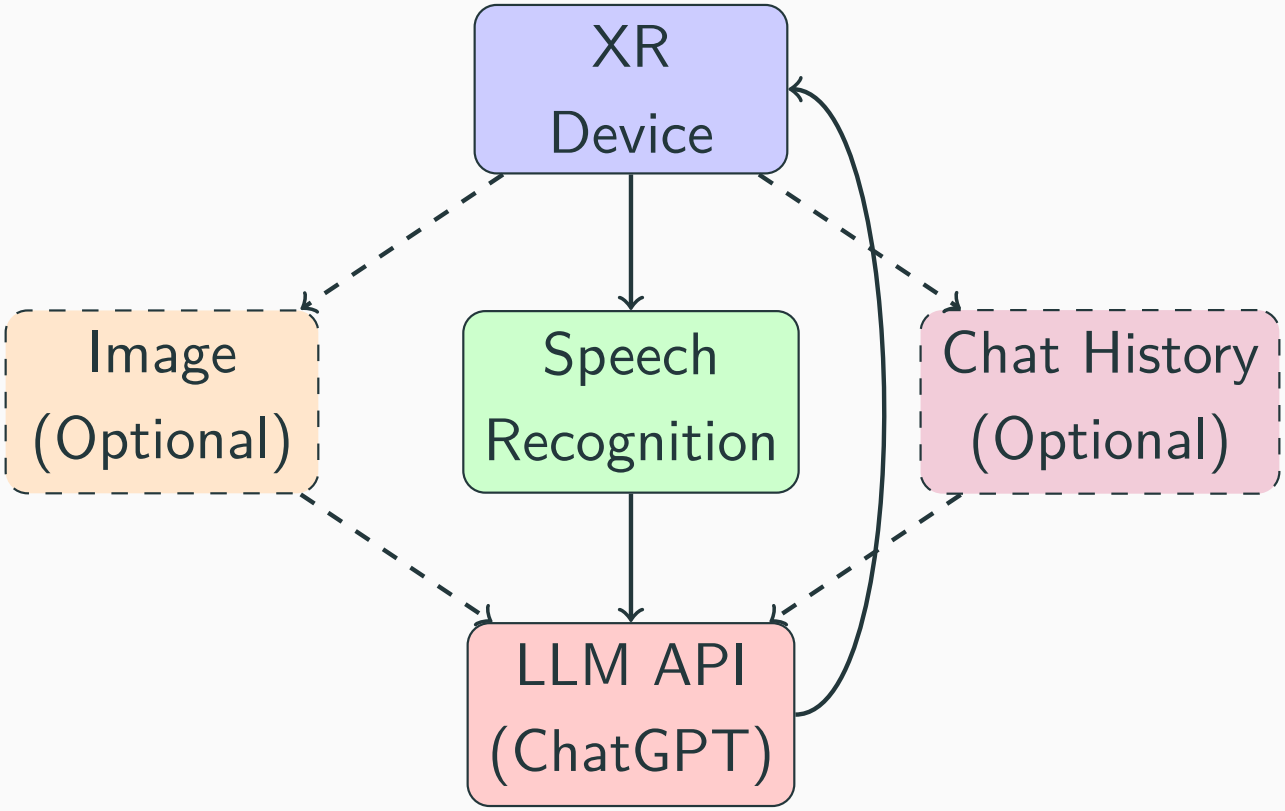
Overview of Topics

- Basic Approach: XR Device + LLM API
  - Building a minimal version inspired by Meta Ray-Ban
- Past Project: HoloWizard
  - Technical challenges in past project
- Advanced Agent Architectures
- Future Directions and Opportunities

Current AI Glasses & Headsets

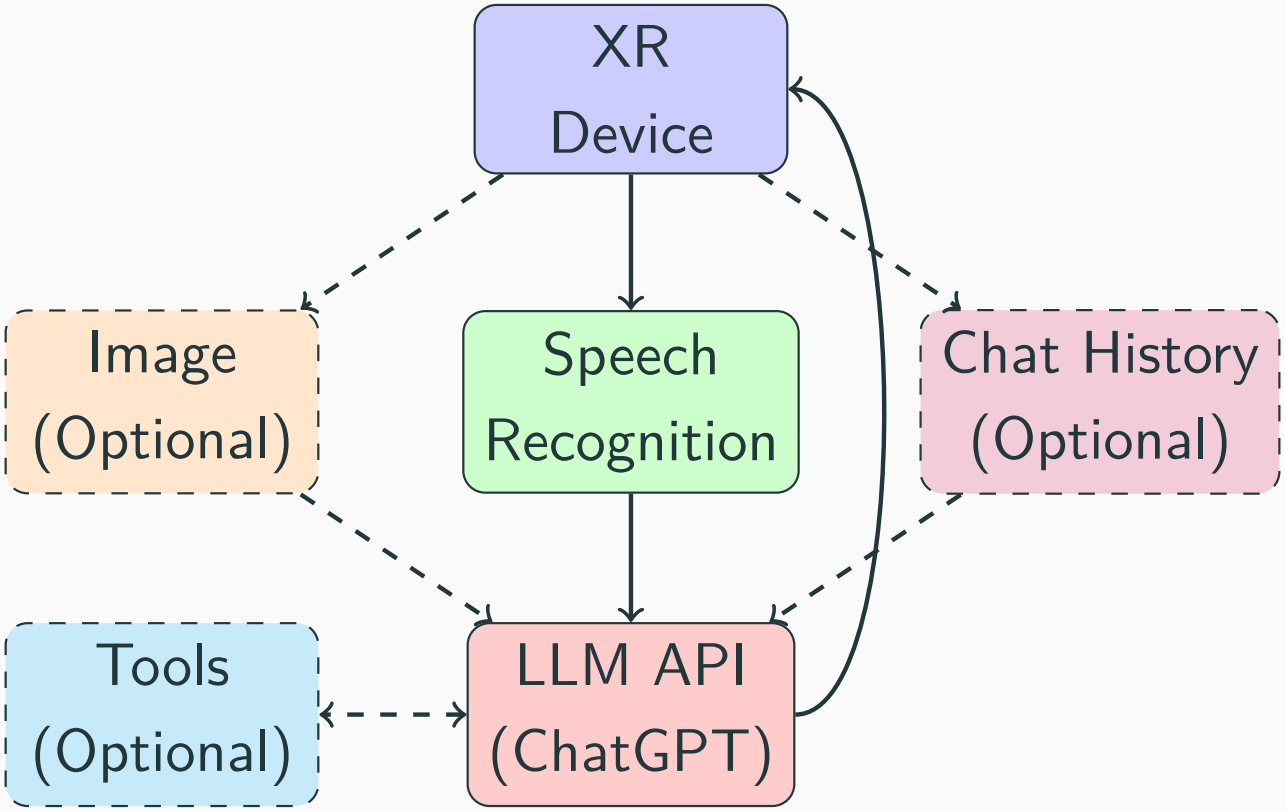
- **Meta Ray-Ban Smart Glasses**
  - Voice commands processed through Meta AI
  - Real-time image capture and analysis
  - Cloud-based LLM processing
- **Google Android XR**
  - Optional in-lens display
  - Gemini AI integration with contextual awareness
  - Mostly cloud-based processing
- **Common Technical Patterns**
  - Speech-to-text $\rightarrow$ LLM API $\rightarrow$ Text-to-speech
  - Multimodal input: voice + computer vision
  - Cloud APIs for natural language interaction and computer vision processing

System Architecture Overview



- **Required**: Speech transcription via ChatGPT API
- **Optional**: Image capture from HoloLens camera
- **Optional**: Previous chat history for context
- **Output**: LLM response displayed on HoloLens

System Architecture Overview



- **Required**: Speech transcription via ChatGPT API
- **Optional**: Image capture from HoloLens camera
- **Optional**: Previous chat history for context
- **Output**: LLM response displayed on HoloLens

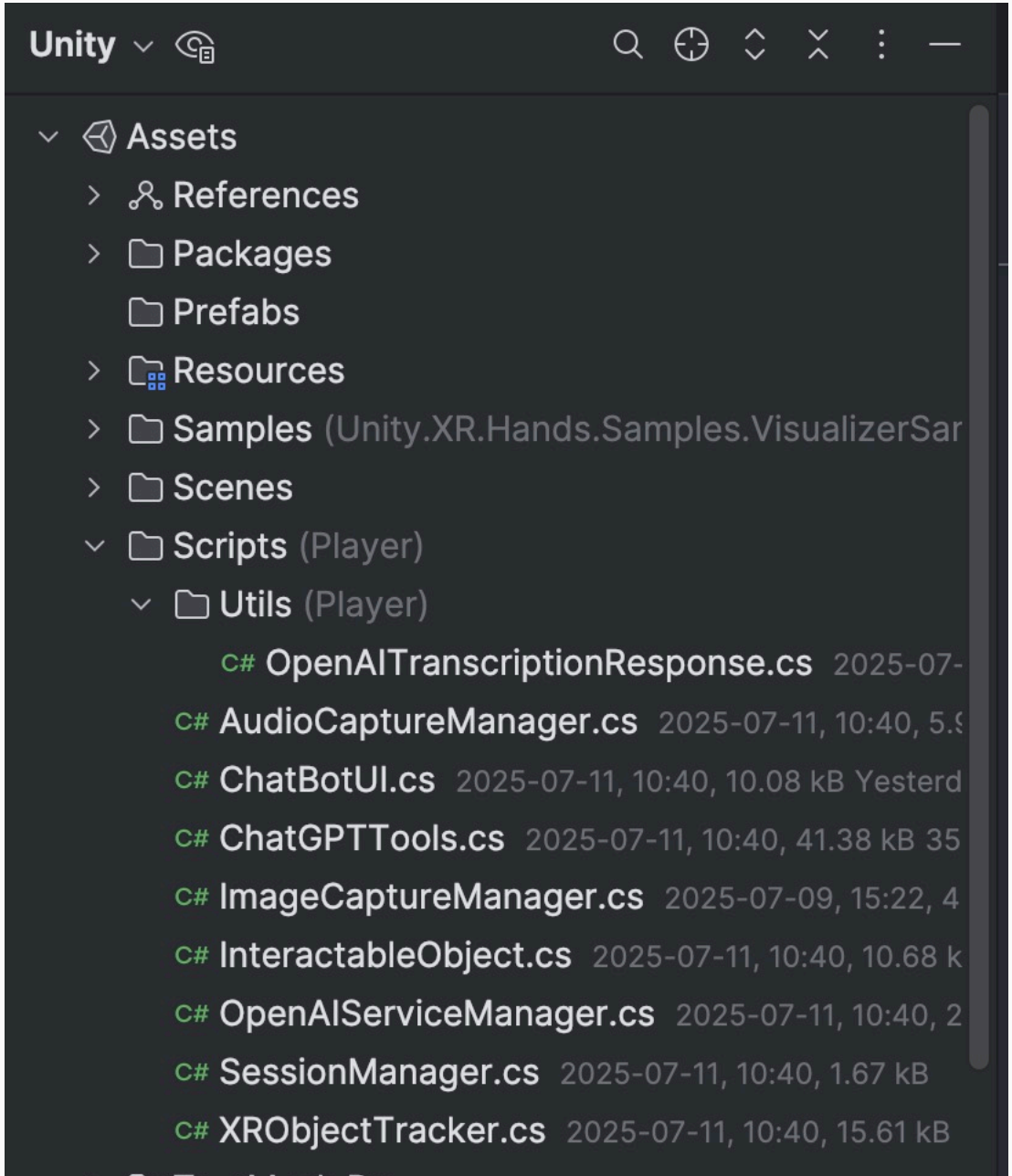
Available Libraries

Transcription + LLM:

- OpenAI .NET API Library
  - Official implementation that follows netstandard2.0 and is based on the net6.0 framework, with excellent compatibility with Unity
  - https://www.nuget.org/packages/OpenAI/
- Microsoft.SemanticKernel
  - An advanced library that supports more features such as vector search locally; however, it relies on newer .NET versions, and some features may not be compatible with Unity
  - https://www.nuget.org/packages/Microsoft.SemanticKernel

For easier use, we will use the OpenAI .NET API Library for this session.

Example Project Structure



Core Components in Example Project

- OpenAIServiceManager
  - Handles ChatGPT API calls and streaming responses, also includes the transcribe
- ChatGPTTools
  - Implemented 6 function calls examples for virtual environment manipulation
  - identify_pointed_object, get_all_tracked_objects, find_and_edit_object, get_camera_info, create_object, remove_object
- AudioCaptureManager && ImageCaptureManager
  - Handles audio and image capture from HoloLens
- SessionManager
  - Manages the session of the chatbot and the chat history

Example Project

If you just want to try out the project and reuse the project as a chatbot in your own project:

- ChatBotUI.cs - User interface with toggles for audio, image, and tools. It should run out of the box if you are using Hololens 2.
- If you are using other OpenXR compatible devices, you need to replace the AudioCaptureManager and ImageCaptureManager with the corresponding implementation that provided by the device manufacturer.

Example Project

If you want to add more features to the chatbot, you can follow this pattern:

- Add new tool in GetAvailableTools() method in ChatGPTTools.cs
- Implement handler in toolHandlers dictionary
- Define JSON schema for parameters
- Tips: HandleGetAllTrackedObjects probably is the easiest one in the example project, you can check it out to see how to implement a new tool

HoloWizard Project Overview

HOLO WIZARD

Settings

WebRTC Stream

Transcripts

Raw Transcript | Cleaned Transcript

Very cool device! When was it released?

- (x) Use Raw Transcript
- ( ) Use Cleaned Transcript

Send to AI | Search with AI

Controls

Take Image and Start Recording

Just Take Image

AI Response

This device looks like the Microsoft HoloLens. The first HoloLens was released in 2016, and the HoloLens 2 was released in 2019.

Search Results

Key Challenges: Beyond Chatbot Functionality

Implementing the chatbot using an API is easy in the wearable headset, but adding more features that are not included in the API is not easy in the current XR development ecosystem.

Key Challenges: Beyond Chatbot Functionality

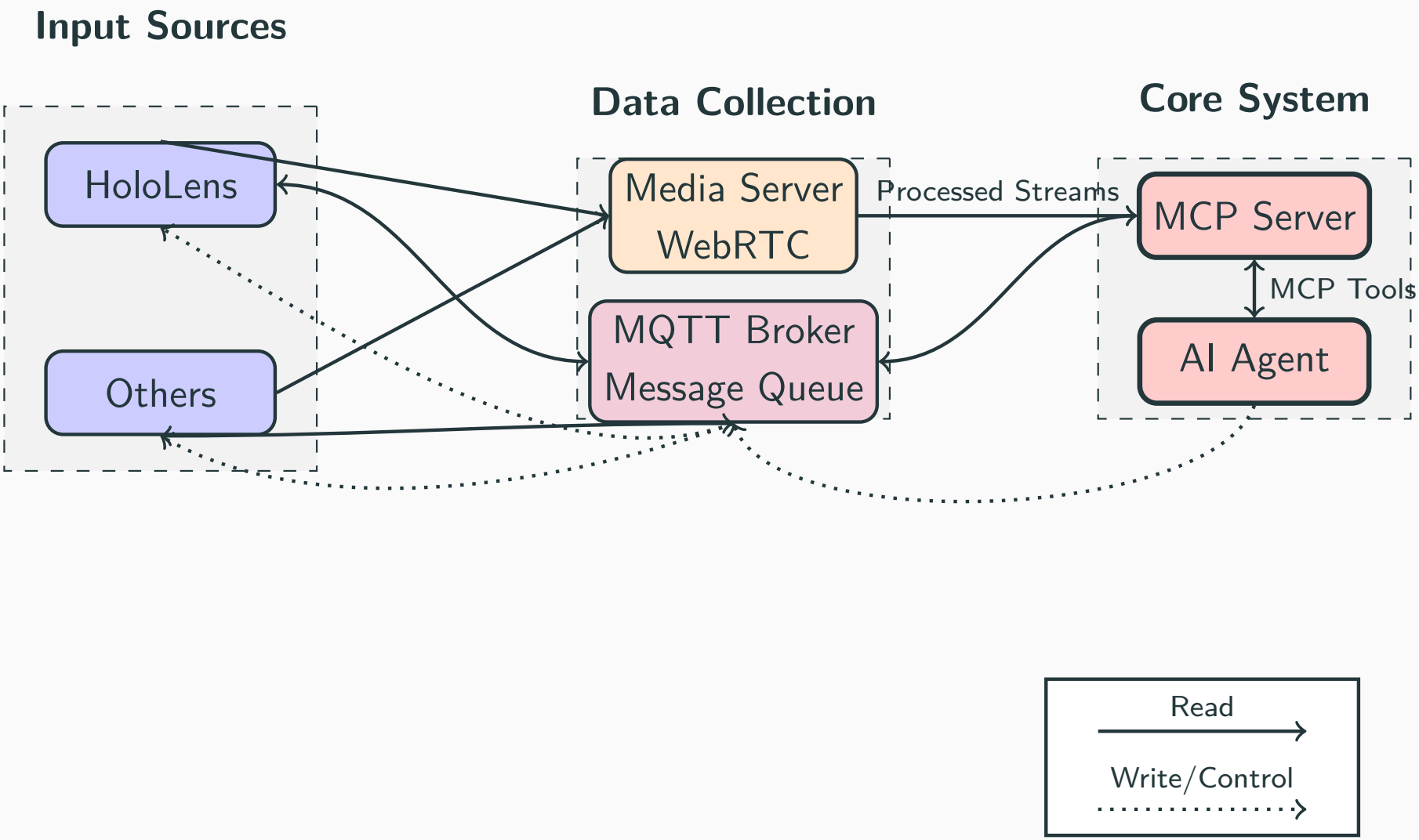
Implementing the chatbot using an API is easy in the wearable headset, but adding more features that are not included in the API is not easy in the current XR development ecosystem.
Here are some of the challenges I faced during the development:

- **Limited Development Tools**
- **Hardware Constraints**
- **Multi-Modal Integration Complexity**

Other Limitations of Basic Approach

- **Maintenance Nightmare**: M $\times$ N problem - multiple tools $\times$ multiple data sources
- **Limited Scalability**: Adding new components requires substantial rewiring
- **Vendor Lock-in**: Difficult to switch between different AI providers or tools

Advanced Distributed Framework Solutions



Distributed Architecture vs Standalone

Why is distributed architecture better in this case?

- **Less programming language constraints**: Wearable headsets typically use C# and C++, which have limited AI development ecosystems compared to Python and JavaScript
- **Less on-device compatibility issues**: No need to worry about code compatibility with the wearable headset
- **Fewer hardware constraints**: Some LLM agent architectures may include features like Retrieval-augmented generation (RAG) or use time-series databases, which are impossible to run on the wearable headset due to memory limitations
- **Easy to utilize LLM "tool use"**: Game engines such as Unity have different lifecycle designs that are not suited for providing many stateful tools for LLM use

Example

```
public class VirtualAssistant : MonoBehaviour
{
    [ServiceDescription("position", "Current 3D position of the assistant", EndpointType.DataSource)]
    [AccessMode(AccessMode.ReadOnly)]
    public Vector3 Position => transform.position;

    [ServiceDescription("moveTo", "Move assistant to a target 3D location", EndpointType.Method)]
    [AccessMode(AccessMode.WriteOnly)]
    [InputMetadata(
        new ParamDescription("x", typeof(float), "1.0", "Target X coordinate"),
        new ParamDescription("y", typeof(float), "1.5", "Target Y coordinate"),
        new ParamDescription("z", typeof(float), "2.0", "Target Z coordinate")
    )]
    public void MoveTo(float x, float y, float z)
    {
        transform.position = new Vector3(x, y, z);
    }

    [ServiceDescription("greet", "Assistant speaks a greeting message", EndpointType.Method)]
    [AccessMode(AccessMode.WriteOnly)]
    public void Greet(string name)
    {
        Debug.Log($"Hello, {name}!");
    }
}
```

Advanced Distributed Framework Pros & Cons

Pros:

- **Standardized MCP Support**: Automatically translate the data source and tools by using annotations in the code to MCP compatible format
- **Standardized Data Flow**: Consistent communication patterns between components and AI agent for better context retrieval and understanding
- **Plug-and-Play Architecture**: Easy integration of new models and tools
- **Action-Oriented AI**: Beyond context retrieval to execution
- **Auto Resource Discovery**: Automatically discover the data source and tools that provided by wearable headsets and other devices

Advanced Distributed Framework Pros & Cons

Cons:

- More context and tools don't equal to smarter AI, and can cause overthinking issues that make AI confused. It is developers' responsibility to use available tools and context to make the AI agent more intelligent

AR/MR Integration with AI

- **Spatial Understanding**: Agent knows what you're looking at and your physical context
- **Virtual Objects Interaction with AI**: AI can manipulate and reference virtual elements in your space
- **Hands-Free Interaction**: Natural voice + gesture or gaze commands while maintaining visual focus
- ...

Questions?

ziruixiao2-c@my.cityu.edu.hk