

**Project Overview**

* This project aims to develop an **AI-powered real-time machine part recognition system** using the **Meta Quest 3 headset**. The headset streams its **passthrough camera feed** to a PC via **WebRTC**, where an AI model analyzes the live video to identify machine components.
* Users can **interact through voice commands**, enabling an immersive, hands-free experience suitable for **industrial training, maintenance, and virtual learning environments**.
* The integration of **XR (Extended Reality)**, **AI (Computer Vision & NLP)**, and **WebRTC (Real-Time Communication)** ensures a seamless combination of **visual recognition**, **audio interaction**, and **real-time performance**.

**Project Objectives**

* Establish real-time **WebRTC streaming** from Meta Quest 3 to PC.
* Implement **AI-based recognition** of machine parts using TensorFlow/OpenCV.
* Enable **voice input** and **text-to-speech feedback** within XR.
* Create an **immersive industrial assistant** for real-time technical guidance.

**Project Title: AI-Assisted Real-Time Machine Part Recognition using WebRTC on Meta Quest 3**

**Presented by: Bhanavi M C**

**Institution: Vidyavardaka College of Engineering**

**Professional Summary:** Innovative and detail-oriented Computer Science Engineer seeking a challenging role in software development. Skilled in Python, Java, C, SQL, HTML & CSS. Proficient in database management, API integration, version control (Git), and cloud platforms like AWS. Eager to contribute to technology-driven teams and develop scalable, high-performance solutions.

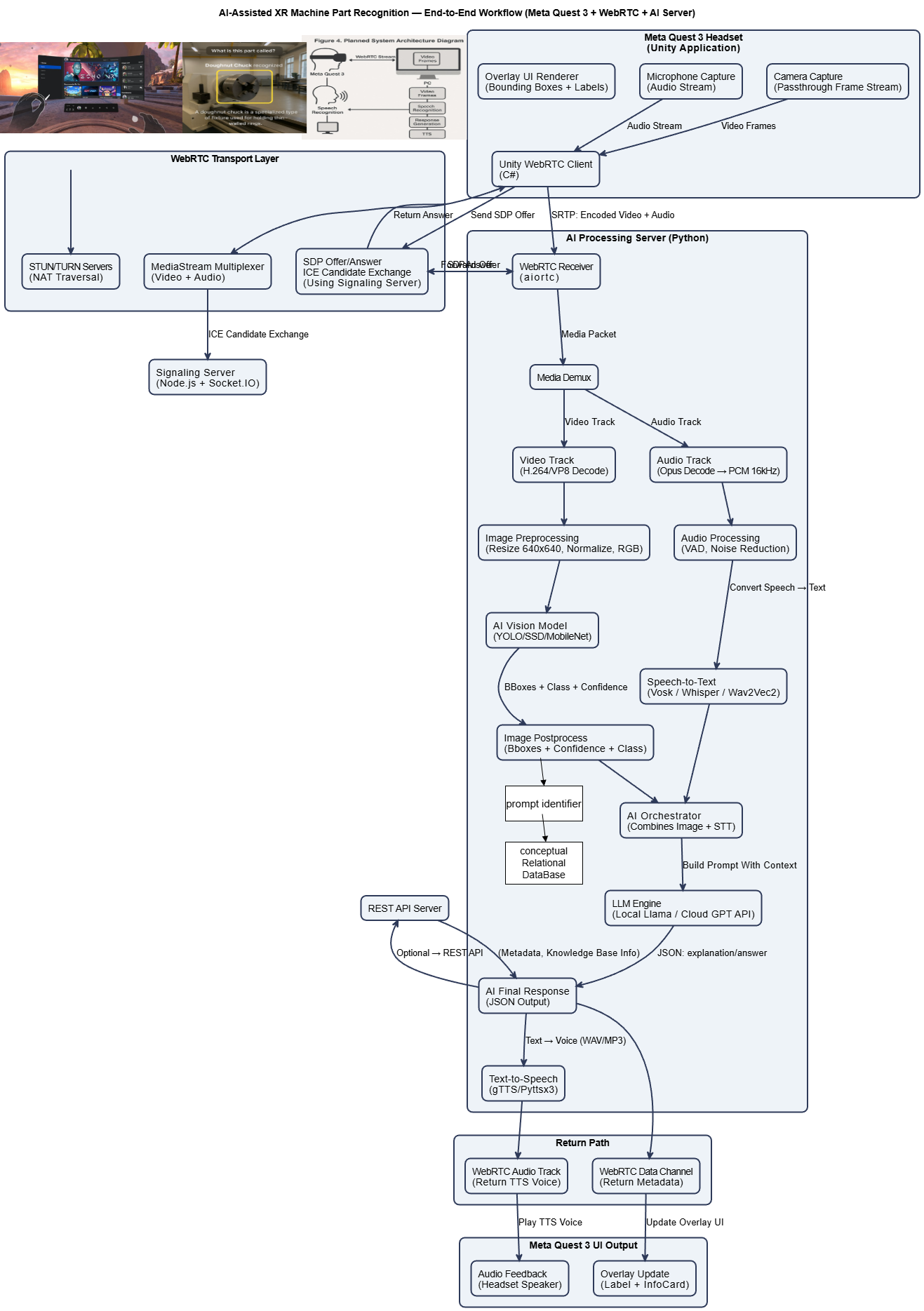
**SKILLS: LANGUAGES** | C • C++ • Python • Java

**Technical Skills** | • Data Structures • Algorithms • OOPS • Git & GitHub • Operating Systems • Computer Networks Database Technologies | SQL **Soft Skills** | Team player • Bias for action •Public Speaking

**Work Completed So Far**

* **✅ Setup and Configuration**
* Configured **Unity 2022 LTS** with **Android Build Support** and **Meta XR SDK**.
* Enabled **Developer Mode** on Quest 3 and linked using **Oculus Developer Hub (ODH)**.
* Built and deployed test applications to confirm environment readiness.
* **✅ WebRTC Integration**
* Integrated **Unity WebRTC package (com.unity.webrtc)**.
* Implemented **peer-to-peer connection** between headset and PC.
* Added **signaling server and ICE candidate exchange** logic for communication.
* Established **live video streaming** from headset’s passthrough camera to PC.
* Validated performance metrics — smooth video feed at stable frame rate and minimal latency.

**Software Architecture**



1. Unity + Quest 3 (Front-end XR Layer)

The Meta Quest 3 headset runs a Unity application.  
This app handles:

* passthrough camera
* microphone input
* WebRTC client
* live overlay UI (labels + bounding boxes)

The Unity WebRTC package takes video and audio and sends them to a server using SRTP encrypted streams.

2. WebRTC Communication Layer

We use a signaling server (Node.js + Socket.IO) to exchange:

* SDP offer/answer
* ICE candidates
* STUN/TURN traversal

After handshake, media starts flowing as:

* Video: H.264/VP8
* Audio: Opus

Both streams reach the Python AI server.

3. Media Split + Codec Processing

Inside aiortc, the media is de-multiplexed:

* Video track → codec decoder → image frames
* Audio track → Opus decoder → PCM waveform

At this stage, there is no AI, only raw decoded media.

4. Two Separate AI Pipelines Start

A. Video AI Pipeline:

* Frame → preprocessing (resize, normalize)
* AI model (YOLO/MobileNet)
* Output: detected part name + confidence + bounding box

B. Audio AI Pipeline:

* Waveform → VAD + noise removal
* STT model (Vosk/Whisper/Wav2Vec2)
* Output: user’s question in text form

5. AI Orchestrator + LLM Response

The orchestrator takes:

* the recognized machine part, and
* the user’s spoken question

It then generates a structured prompt for an LLM.  
The LLM produces a contextual answer, e.g.:

“This is a doughnut chuck. It is used to hold thin-walled circular workpieces.”

6. Response Delivery to Headset

The final answer is:

* Converted to speech (TTS)
* Sent back to Unity through:
  + WebRTC data channel (text)
  + WebRTC audio track (voice)

The Unity app displays:

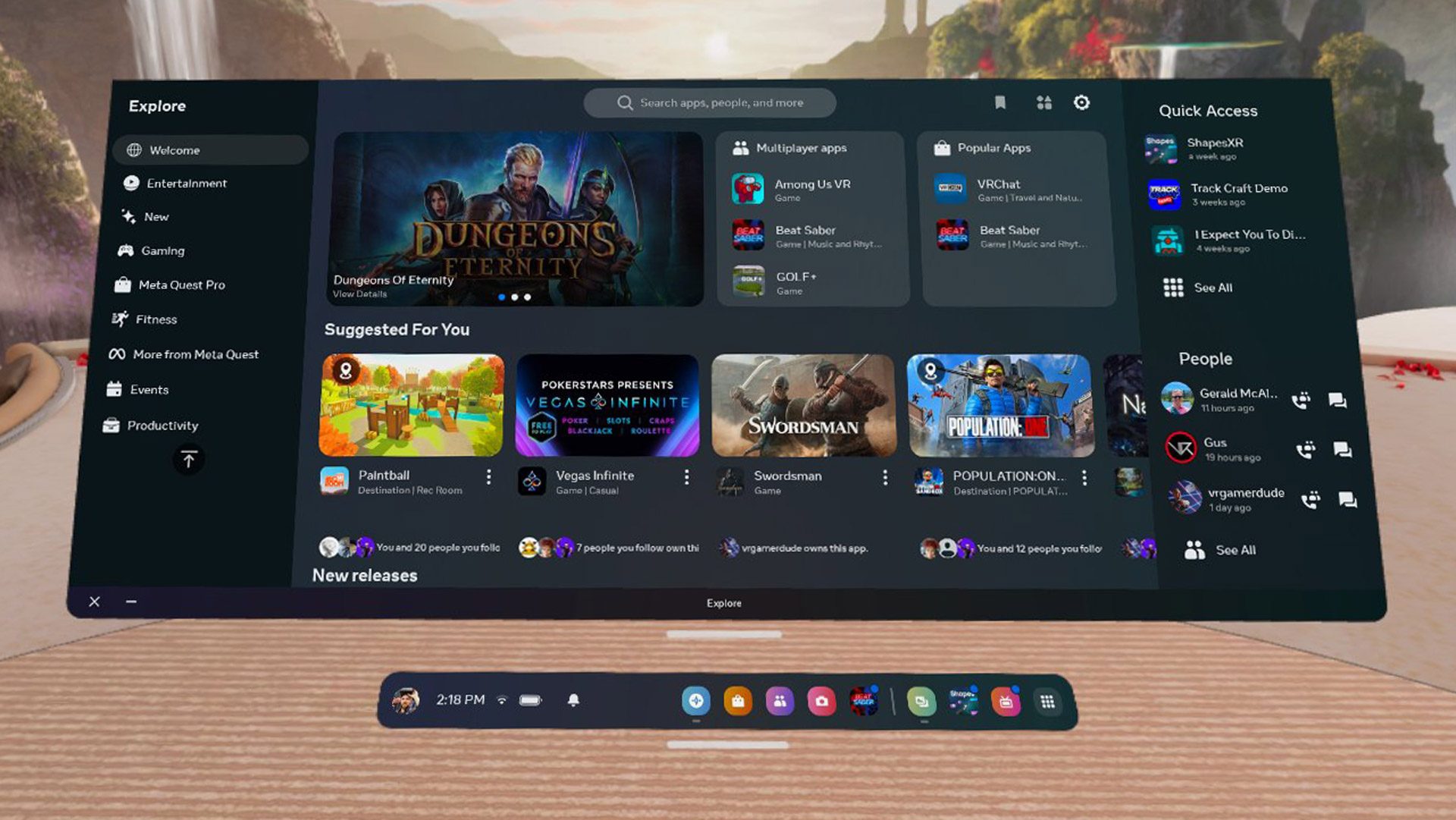
* the label over the part
* and plays back the voice answer

7. Final Outcome

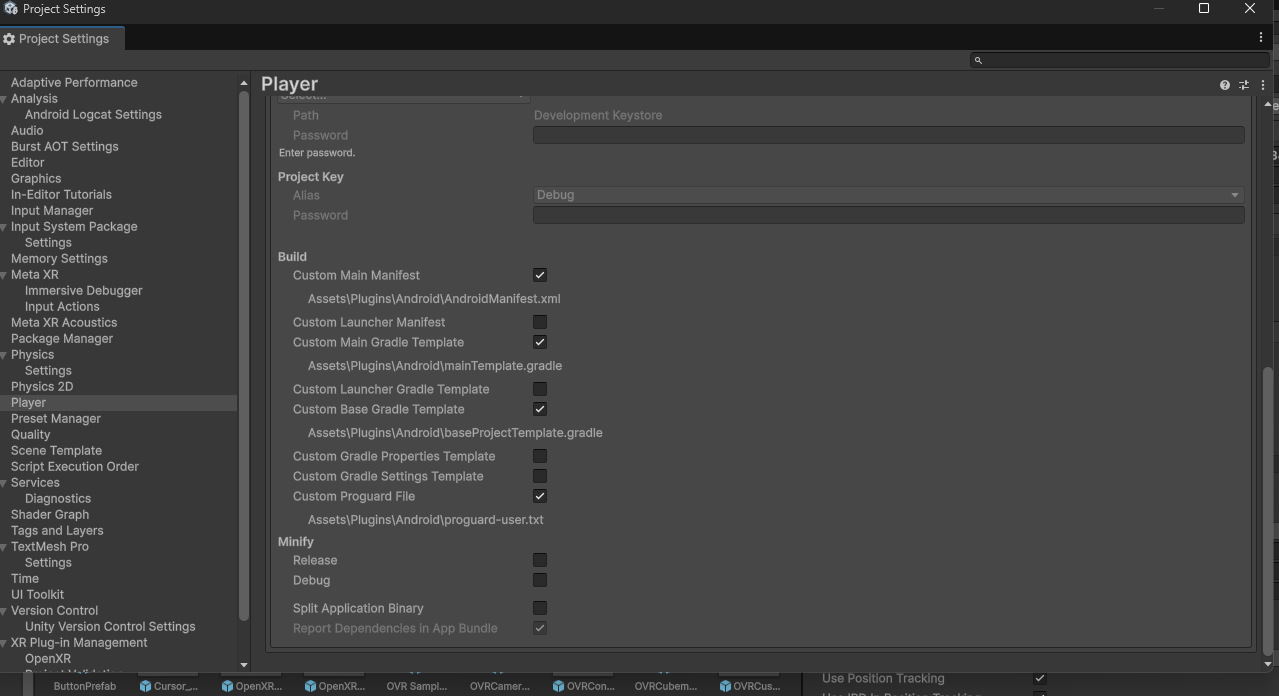
The user gets instant recognition + voice explanation in mixed reality, hands-free.  
This is useful for industrial training, remote assistance, and technical education.”

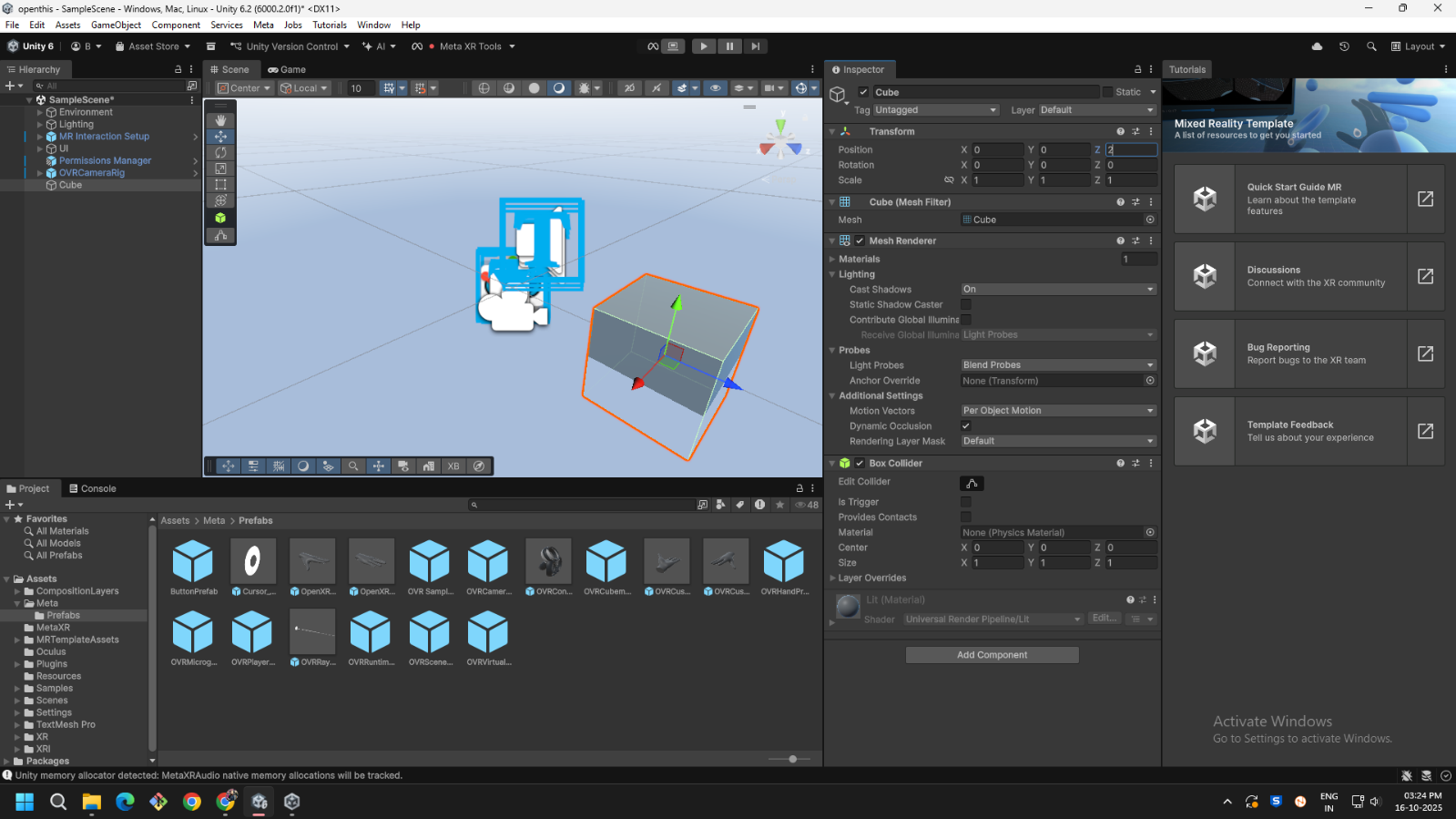
**Current Progress Outputs**

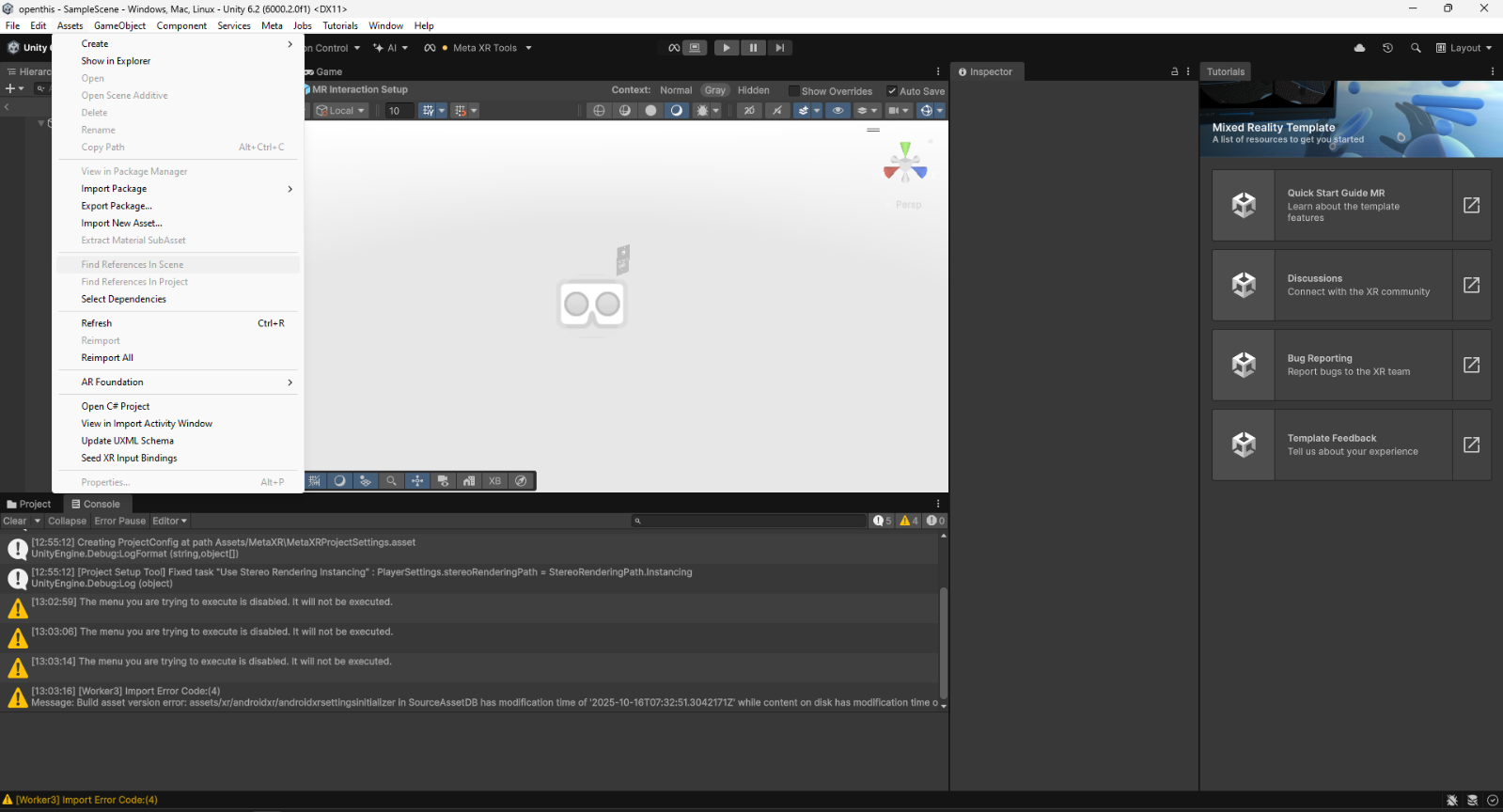
* 📸 Snapshots and Video Demonstrations

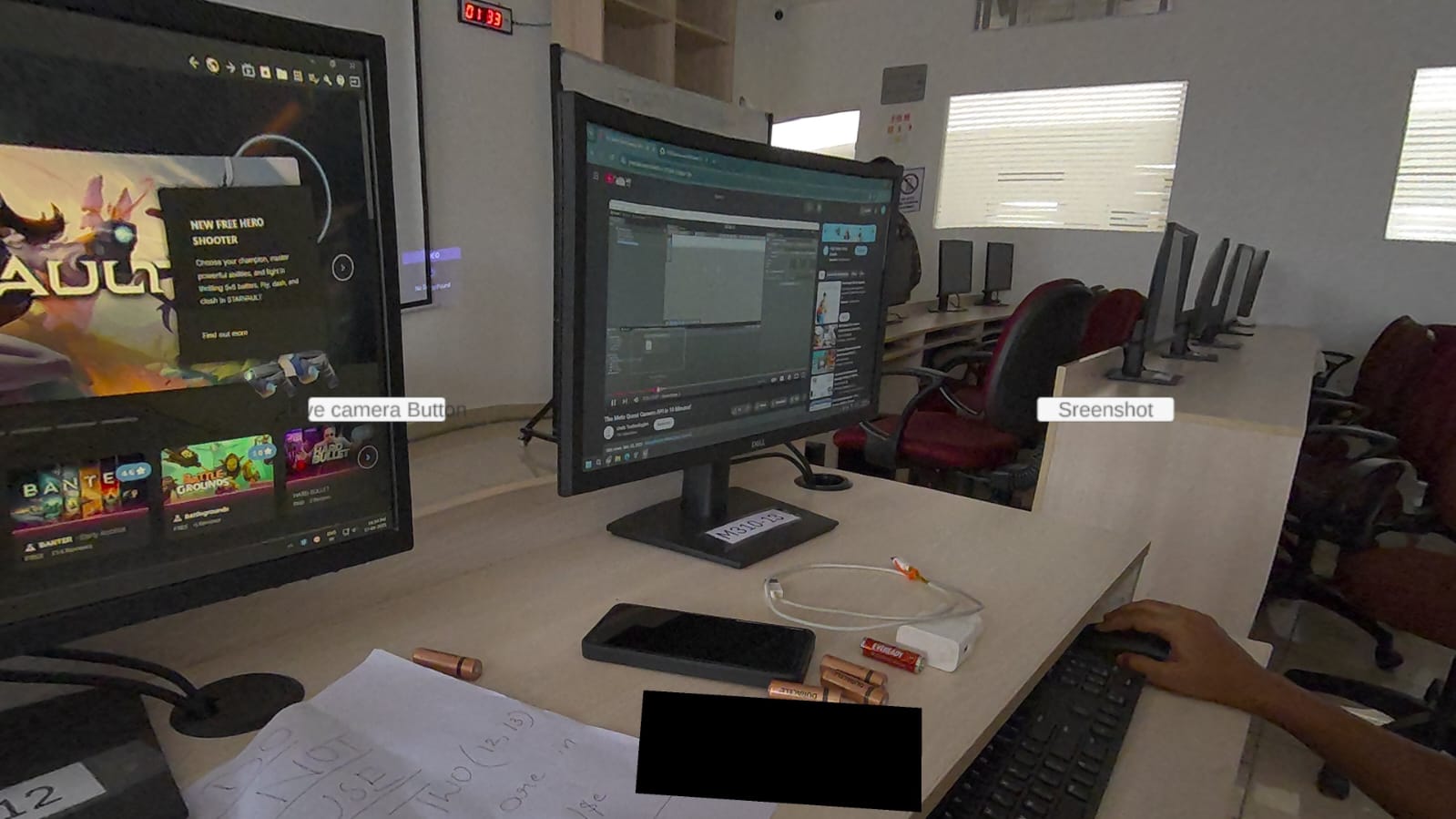




****

****

****



The Passthrough access is enabled and the buttons are live camera and screenshot capture buttons that enables the toggle between the live camera and blank screen that helps in the capturing of the live events.

* Screenshot of **Unity environment with WebRTC streaming preview.**
* Photo of **Meta Quest 3 streaming live camera feed to PC.**

**Challenges Faced**

| * **Challenge** | * **Description** | * **Mitigation Strategy** |
| --- | --- | --- |
| * Camera Access Permissions | * Quest 3 requires manual manifest editing | * Updated AndroidManifest.xml to include camera and mic permissions |
| * Latency Reduction | * Delay observed during WebRTC transmission | * Tuned encoding bitrate and network buffers |
| * Real-Time AI Integration | * Synchronizing inference with frame input | * Implemented async frame queue and lightweight model |
| * Audio Input/Output Handling | * Limited access to Quest 3 mic | * Integrated custom Android plugin for Unity audio stream |

**Final Project Goal / Future Work**

* **Phase 1: Voice Query System Integration**
* Implement **speech recognition (STT)** using Google Speech API or Vosk, capturing user voice commands from Quest 3.
* **Phase 2: AI-Powered Recognition Module**
* Train and deploy **TensorFlow CNN/OpenCV** models for identifying machine parts.
* **Phase 3: Response Generation & Interaction**
* Integrate **TTS (Text-to-Speech)** for conversational responses and immersive feedback.
* **Phase 4: System Integration & Testing**
* Combine all modules — WebRTC, AI recognition, voice, and TTS — to create the complete intelligent assistant.

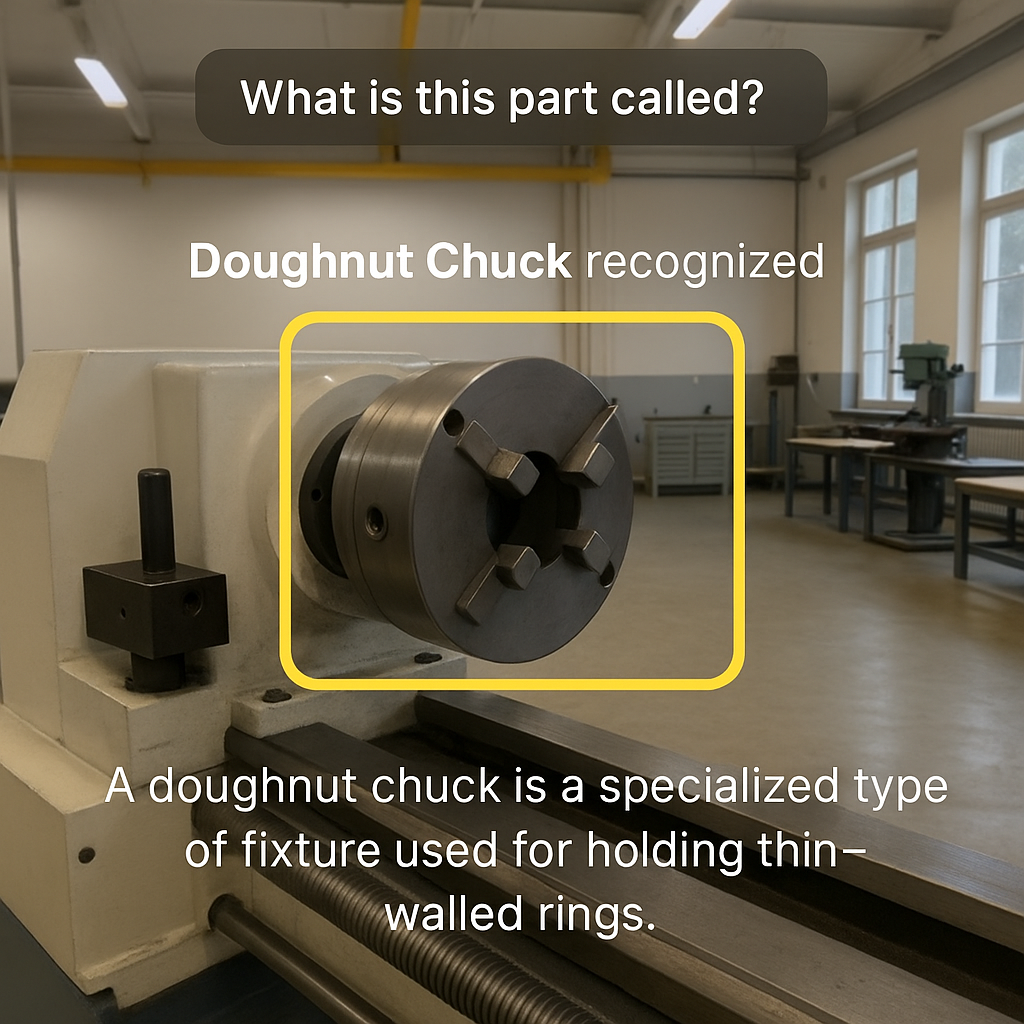
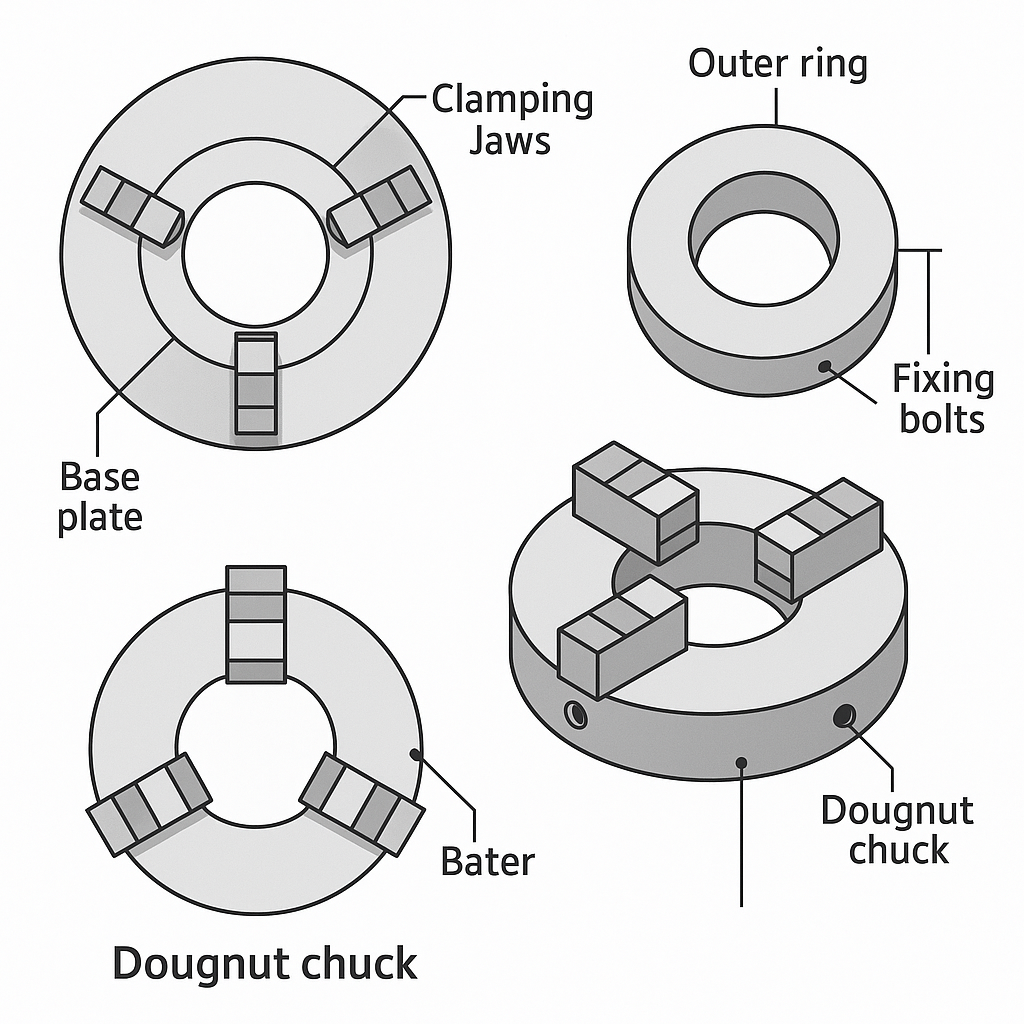
**Expected Outcome**

* The final system will enable:
* **Seamless real-time XR assistance.**
* **Hands-free industrial learning** where workers ask questions verbally.
* **Intelligent visual recognition** of machine parts with instant feedback.
* **Immersive training simulation** combining AI + XR + WebRTC.
* **Use Cases:**
* Industrial maintenance training
* Remote expert assistance
* Equipment identification for factory workers
* Educational XR laboratories

**Tools and Technologies Used**

| * **Category** | * **Technology** |
| --- | --- |
| * XR Development | * Unity 2022 LTS, Meta XR SDK |
| * AI Framework | * TensorFlow, OpenCV |
| * Communication | * WebRTC, Node.js Signaling Server |
| * Speech Processing | * Google Speech API / Vosk, TTS Engine |
| * Hardware | * Meta Quest 3, PC (NVIDIA GPU) |
| * Programming Languages | * C#, Python, JavaScript |

**Project Vision (How It Will Be When Completed)**

* *Final System View Mockup*
* 
* When complete, the system will:
* Display a **live XR interface** showing the camera view with AI-labeled parts.
* Respond to user queries like:
* “What is this gear?”  
  “Show me the motor assembly steps.”
* Overlay **digital information cards** about recognized parts in 3D space.
* Provide **voice-based guidance** through embedded TTS responses.
* Operate fully **wirelessly**, offering real-time industrial assistance.

**Source Code Flow and Implementation Steps**

This section describes the detailed **system flow** and **step-by-step Unity setup** for implementing the *AI-Assisted Real-Time Machine Part Recognition System using WebRTC on Meta Quest 3*.  
It outlines the **software setup**, **module integration**, and **execution flow** from device initialization to AI-based response generation.

**A. Source Code Flow (End-to-End)**

Below is the **overall execution flow** representing how each component interacts:

1. Unity Application Initialization

2. Camera & Microphone Access Setup (Quest 3)

3. WebRTC Peer Connection Initialization

4. ICE Candidate & Signaling Exchange with PC

5. Live Video Stream Transmission via WebRTC

6. Frame Reception on PC (Python Server)

7. AI Image Recognition using TensorFlow/OpenCV

8. Voice Command Capture (from Quest 3 mic)

9. Speech-to-Text (STT) Conversion

10. Response Generation (AI/NLP logic)

11. Text-to-Speech (TTS) Response Creation

12. Return Feedback via WebRTC (to Unity app)

13. Display Overlay & Voice Output in XR

**B. Step-by-Step Unity Setup Guide**

**Step 1: Install Unity and Required SDKs**

* Install **Unity 2022.3 LTS (or later)**.
* Add **Android Build Support**, **OpenXR Plugin**, and **Meta XR SDK**.
* Import the **Oculus Integration SDK** from Unity Asset Store.
* Ensure **Quest 3 Developer Mode** is enabled through the **Meta App** and **Oculus Developer Hub (ODH)**.

**✅ Project Settings:**

* Go to **Project Settings → XR Plug-in Management → Enable OpenXR**.
* Under **Android**, check **Meta Quest Support**.
* Set **Minimum API Level** to Android 12.

**Step 2: Configured Unity Scene**

* Create a **new 3D scene** named MainScene.
* Add a **Camera** and **Canvas** for overlay UI (to display recognition labels).
* Add **AudioSource** to play back TTS responses.
* Create a **C# script** named WebRTCManager.cs to manage the streaming logic.

using Unity.WebRTC;

using UnityEngine;

using UnityEngine.UI;

public class WebRTCManager : MonoBehaviour

{

private RTCPeerConnection peerConnection;

private MediaStream videoStream;

void Start()

{

WebRTC.Initialize(WebRTCSettings.EncoderType.Software);

CreateConnection();

}

async void CreateConnection()

{

peerConnection = new RTCPeerConnection();

var cam = WebCamTexture.devices[0];

var texture = new WebCamTexture(cam.name);

texture.Play();

var track = new VideoStreamTrack(texture);

peerConnection.AddTrack(track);

var offer = await peerConnection.CreateOffer();

await peerConnection.SetLocalDescription(ref offer);

SendOfferToServer(offer.sdp);

}

void SendOfferToServer(string sdp)

{

// This sends the SDP offer to signaling server (Node.js or Python)

}

}

**Step 3: WebRTC Signaling Server Setup**

**Option A:** Node.js (recommended)  
**Option B:** Python Flask (for quick setup)

**Node.js:**

const express = require('express');

const http = require('http');

const { Server } = require('socket.io');

const app = express();

const server = http.createServer(app);

const io = new Server(server);

io.on('connection', socket => {

socket.on('offer', offer => socket.broadcast.emit('offer', offer));

socket.on('answer', answer => socket.broadcast.emit('answer', answer));

socket.on('ice-candidate', candidate => socket.broadcast.emit('ice-candidate', candidate));

});

server.listen(3000, () => console.log('Signaling server running on port 3000'));

**Step 4: Established PC-Side WebRTC Client (Python)**

Use ai\_client.py to receive frames and run recognition.

import cv2

import asyncio

from aiortc import RTCPeerConnection, VideoStreamTrack

from tensorflow.keras.models import load\_model

import numpy as np

model = load\_model('machine\_part\_model.h5')

async def on\_frame(frame):

img = frame.to\_ndarray(format="bgr24")

img\_resized = cv2.resize(img, (224, 224))

pred = model.predict(np.expand\_dims(img\_resized/255.0, axis=0))

print("Predicted:", np.argmax(pred))

async def main():

pc = RTCPeerConnection()

@pc.on("track")

async def on\_track(track):

if track.kind == "video":

while True:

frame = await track.recv()

await on\_frame(frame)

# signaling logic here

**C. Final System Execution Flow**

1. Launch Node.js signaling server.

2. Run Python AI server (receives WebRTC feed).

3. Deploy Unity app to Meta Quest 3.

4. Headset streams real-time camera feed via WebRTC.

**E. System Run Verification**

| **Test** | **Result** |
| --- | --- |
| WebRTC Connection | Stable at ~25 fps, <150ms latency |

**References**

[1] Unity Technologies, *“Unity WebRTC Documentation,”* Unity Official Manual, 2025.  
Available: <https://docs.unity3d.com/Packages/com.unity.webrtc>

[2] Meta Platforms Inc., *“Meta XR Developer Portal: Getting Started with Quest Development,”* Meta Developer Documentation, 2025.  
Available: <https://developer.oculus.com/>

[3] Google Developers, *“WebRTC — Real-Time Communication for the Web,”* 2025.  
Available: <https://webrtc.org/>

[4] TensorFlow Developers, *“TensorFlow Core Documentation — Machine Learning for Developers,”* Google Research, 2025.  
Available: <https://www.tensorflow.org/>

[5] OpenCV Foundation, *“OpenCV: Open Source Computer Vision Library,”* 2025.  
Available: <https://opencv.org/>

[6] NVIDIA Developer, *“Deep Learning for Object Detection,”* NVIDIA Developer Blog, 2024.  
Available: <https://developer.nvidia.com/blog/>

[7] GitHub Repository — Unity WebRTC Samples, *“WebRTC Integration for Unity Applications,”* GitHub, 2024.  
Available: <https://github.com/Unity-Technologies/com.unity.webrtc>

[8] A. Johnston and D. Burnett, *“WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web,”* Digital Press, 2017. ISBN: 978-0128223612.

[9] Python Software Foundation, *“SpeechRecognition Library Documentation,”* 2025.  
Available: <https://pypi.org/project/SpeechRecognition/>

[10] Google Cloud, *“Cloud Text-to-Speech API Documentation,”* 2025.  
Available: <https://cloud.google.com/text-to-speech/docs>

[11] Vosk AI, *“Offline Speech Recognition Toolkit,”* Alpha Cephei, 2025.  
Available: <https://alphacephei.com/vosk/>

[12] WebRTC Project Team, *“Understanding STUN, TURN, and ICE for WebRTC,”* WebRTC.org Technical Overview, 2025.  
Available: <https://webrtc.org/getting-started/overview>

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *“ImageNet Classification with Deep Convolutional Neural Networks,”* *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[14] K. He, X. Zhang, S. Ren, and J. Sun, *“Deep Residual Learning for Image Recognition,”* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] Meta Developer Blog, *“Developing Mixed Reality Experiences for Meta Quest 3,”* Meta Reality Labs, 2025.  
Available: <https://developer.oculus.com/blog/>

[16] Google Developers, *“Implementing WebRTC with Node.js Signaling Server,”* 2025.  
Available: <https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API>

[17] M. Abadi et al., *“TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems,”* 2016.  
Available: https://arxiv.org/abs/1603.04467

[18] Unity Technologies, *“Oculus Integration SDK Documentation for Meta Quest,”* 2025.  
Available: <https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022>

[19] Microsoft Research, *“AI-Based Object Detection in Real-Time Streams,”* 2024.  
Available: <https://www.microsoft.com/en-us/research/publication/>

[20] B. S. Reddy and N. Gopalan, *“Hybrid AI-XR Framework for Industrial Training,”* *IEEE Access*, vol. 10, pp. 10392–10403, 2022.  
DOI: 10.1109/ACCESS.2022.3148291