UNIVERSITY OF CALIFORNIA,
IRVINE

Optimizing Gesture Recognition on Hololens 2 Using CNNs with Early Exits

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

MASTERS OF SCIENCE

in Electrical Engineering and Computer Science

by

Ruchi Jagdish Patel

Dissertation Committee:
Professor Salma Elmalaki, Chair
Professor Yasser Shoukry
Professor Hyoukjun Kwon

2024

TABLE OF CONTENTS

Page

LIST OF FIGURES **iv**

LIST OF TABLES **v**

ACKNOWLEDGMENTS **vi**

VITA **vii**

ABSTRACT OF THE DISSERTATION **viii**

1 Introduction **1**

2 Related Work & Background **4**

3 Methodology **7**
3.1 Gesture Recognition and Early Exits 7
3.1.1 Machine Learning Model and Early Exits 7
3.1.2 Dataset & Preprocessing 10
3.1.3 Model Training with Sequential Learning 11
3.1.4 Evaluaution & Deployment 12
3.1.5 ONNX Integration 13
3.1.6 Experimental Setup 14
3.2 Sensory Profiling 17
3.2.1 Challenges in Sensor Integration 19
3.2.2 Experimental Study 20
3.3 Multi-user application 22
3.3.1 System Design 23
3.3.2 Implementation 25
3.3.3 Experiment 27

4 Results & Discussion **29**
4.1 Gesture Recognition and Early Exits 29
4.2 Sensory Profiling 38

4.2.1 Single Sensor Performance 38
4.2.2 Dual Sensor Performance 39
4.2.3 Three Sensor Performance 40
4.3 Multi-user application 41

5 Conclusion & Future work **43**

Bibliography **46**

Appendix A Gesture Recognition Model Architecture **49**
A.1 Model Initialization 49
A.2 Model Forward Pass 51

LIST OF FIGURES

Page

3.1 Gesture Recognition Model Architecture & Early Exits 9
3.2 Hololens 2 Sensors 17
3.3 Multiuser Shared Experience 22
3.4 Collaborative Mixed Reality Setup 26
3.5 Collaborative Setup with 4 Users 27
3.6 Shared Hologram 27

4.1 Accuracy of Prediction for Gesture 1- 10 31
4.2 Time/ Inference for Gesture 1- 10 31
4.3 FPS for Gesture 1- 10 32

LIST OF TABLES

Page

3.1 Documented FPS and Sampling Rates for HoloLens 2 Sensors 18

4.1 Different Gesture Images 30
4.2 Battery Drain Across Different Exits for Gesture 1. 32
4.3 Battery Drain Across Different Exits for Gestures 5-7. 33
4.4 Battery Drain Across Different Exits for Gestures 8-10. 34
4.5 Battery Drain Across Different Exits for Gestures 2-4. 35
4.6 Performance Metrics Across Exits for Gestures 1 to 6 36
4.7 Performance Metrics Across Exits for Gestures 7 to 10 37
4.8 Performance Evaluation of Individual Sensors 38
4.9 Performance Evaluation of Two-Sensor Combinations 39
4.10 Performance Evaluation of Three-Sensor Combinations 40
4.11 Data Collection Specifications During Testing Phase 41

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Professor Salma Elmalaki, for her invaluable guidance, encouragement, and support throughout this research. Her expertise and insights have been instrumental in shaping this thesis and advancing my understanding of mixed reality technologies.

I am also grateful to my thesis committee members, Professor Yasser Shoukry and Professor Hyoukjun Kwon, for their thoughtful feedback and suggestions, which have greatly improved the quality of this work. I extend my thanks to my friends and collaborators, especially Diana Romero, for their helpful discussions and technical assistance during this research.

I would like to acknowledge the resources and support provided by the University of California, Irvine, particularly the Pervasive Autonomy Lab, for fostering an environment of innovation and research. I am also thankful for the technical assistance provided by Microsoft Hololens 2 and for the datasets and tools contributed by various teams and organizations, which were essential to the success of this project.

On a personal note, I am deeply thankful to my family and friends for their unwavering love, patience, and encouragement during this journey. Their belief in me has been a constant source of motivation.

Finally, I dedicate this thesis to my parents, whose unwavering support, encouragement, and love have been my greatest strength throughout this journey. Their belief in me has been a constant source of motivation, and I am forever grateful for their guidance and sacrifices.

VITA

Ruchi Jagdish Patel

EDUCATION

Masters of Science in Electrical Engineering & Computer Science **2024**
University of California, Irvine *Irvine, CA*

Bachelor of Technology in Electrical and Electronics Engineering **2023**
PES University *Bengaluru, Karnataka*

ABSTRACT OF THE DISSERTATION

Optimizing Gesture Recognition on Hololens 2 Using CNNs with Early Exits

By

Ruchi Jagdish Patel

Masters of Science in Electrical Engineering and Computer Science

University of California, Irvine, 2024

Professor Salma Elmalaki, Chair

In recent years, gesture recognition has become an important tool for enabling interaction with augmented and mixed reality devices. This thesis explores the deployment of a gesture recognition application on the Microsoft Hololens 2 using a Convolutional Neural Network (CNN) with multiple early exits to improve efficiency in real-time scenarios. The use of early exit mechanisms at different layers of the network helps reduce energy consumption and processing time while maintaining reasonable accuracy. However, the average frames per second (FPS) could be further improved for smoother interaction. The project also examines sensor integration, studying how multiple Hololens 2 sensors work together. Lastly, a multi-user application is developed to enable real-time interaction among users in a shared augmented reality environment. These efforts aim to improve the usability and practicality of mixed reality systems for various applications.

Chapter 1

Introduction

The rapid growth of augmented and mixed reality (AR/MR) technologies is transforming the way users interact with digital world, seamlessly blending both the physical and virtual worlds. Devices such as Microsoft Hololens 2 offers platform for immersive experiences across various industries, from gaming to healthcare to remote collaboration[14]. However, developing efficient, scalable and interactive solution for AR/MR poses unique challenges, particularly on resource-constrained devices.

This thesis explores the development and deployment of a gesture recognition application on Hololens 2, employing a Convolutional Neural Network (CNN) with multiple early exits. Early exits are designed to optimize inference by allowing predictions at intermediate layers. Experimental results validate the effectiveness of early exits in reducing inference time. However, the findings indicate that the average frames per second (FPS) requires further improvement to meet the desired standards for smooth real-time interaction.

To enhance the user experience, we further investigate the integration of additional sensors to achieve seamless and smooth interactions[15]. Furthermore, a collaborative mixed reality ap-

plication was developed to demonstrate the system's potential in multi-user environments[8]. This collaborative aspect highlights the importance of synchronizing sensor data and optimizing system performance for effective interaction within shared mixed reality spaces.

The work is structured into three interconnected phases, each building upon the other to achieve a comprehensive system for immersive and efficient mixed reality experience

1. **Gesture Recognition with Early Exits**: A CNN model trained on the LeapGestRecog dataset is deployed on Hololens 2[9]. The model incorporates sequential early exits, enabling real-time gesture recognition while balancing accuracy and computational efficiency. Results demonstrate an increase in accuracy and time per inference though FPS remains a challenge.

2. **Multi-Sensor Integration Study**: Beyond gesture recognition, a study was conducted to evaluate the integration of multiple sensors available on the HoloLens 2, such as depth cameras, microphone, and many more modules. The study aimed to understand how these sensors could be synchronized to achieve a reasonable sampling rate and FPS for mixed reality applications. Using HL2SS (HoloLens 2 Streaming Server)[7], sensor data was accessed and analyzed to determine their potential for enhancing real-time performance and accuracy in complex scenarios.

3. **Development of a Multi-User Application**: A collaborative mixed reality application is developed, allowing multiple users to interact in a shared virtual environment[8]. This demonstrates the potential of Hololens 2 in enabling interactive and immersive experiences.

The findings of this research contribute to the field of mixed reality by optimizing resource utilization and enhancing user interaction capabilities. By combining gesture recognition,

sensor integration study, and multi-user collaboration, this work lays the foundation for future innovations in the domain of immersive technologies.

Chapter 2

Related Work & Background

The rapid advancements in mixed reality (MR) technologies have enabled novel and immersive ways for users to interact with virtual and physical environments. Gesture recognition has become a cornerstone for enhancing MR experiences, leveraging deep learning models such as convolutional neural networks (CNNs) to enable intuitive user interfaces. The introduction of BranchyNet by Teerapittayanon et al. (2016) marked a significant step toward optimizing neural network inference through early exits [21]. By allowing predictions at intermediate layers, BranchyNet demonstrated reduced computational costs and latency, which is especially critical for resource-constrained devices like the HoloLens 2. This approach has inspired further research on implementing such strategies for real-time applications where efficiency is paramount.

Multi-sensor integration has also become a vital area of exploration in MR applications. Ungureanu et al. (2020) provided insights into utilizing the HoloLens 2 Research Mode, which grants developers access to raw sensor streams, including depth cameras, infrared sensors, and IMUs[22]. Their work laid the groundwork for understanding how sensor data

can enhance MR experiences through improved interaction capabilities and environmental awareness. Complementing this, Dibene and Dunn (2022) introduced the HL2SS (HoloLens 2 Streaming Server), which enables synchronized data acquisition from multiple sensors[5]. Their research emphasizes the importance of achieving real-time performance through effective multi-sensor synchronization, a challenge in many MR applications due to the high data throughput required for seamless user experiences.

Gesture recognition datasets, such as LeapGestRecog, have played a crucial role in evaluating and benchmarking gesture-based MR systems. Chowdhury et al. (2020) created this dataset to capture a wide range of hand gestures using Leap Motion Controller, providing a robust platform for testing gesture recognition algorithms[4]. Their work demonstrates the potential of such datasets in training and evaluating CNN models for accurate and efficient gesture recognition.

Collaborative MR applications have also garnered significant attention in recent years. Azuma et al. (2021) explored the potential of MR in enabling multi-user interactions, with applications spanning healthcare, education, and remote collaboration[2]. Their work highlighted the technical challenges of synchronizing user inputs and maintaining system performance in shared virtual spaces. This research underscores the importance of developing scalable MR systems that can adapt to complex multi-user environments.

To address challenges related to real-time inference, optimization frameworks like TensorRT and ONNX Runtime have been extensively studied. Bai et al. (2020) discussed the use of TensorRT to accelerate neural network inference on edge devices, demonstrating significant improvements in inference speed without compromising accuracy[3]. Similarly, Paszke et al. (2019) explored ONNX Runtime for optimizing real-time neural network performance, particularly for deployment on hardware-constrained devices like HoloLens 2.

The integration of audio and visual modalities in MR has further enhanced interaction accuracy and contextual understanding. Sodhi et al. (2019) introduced LightGuide, a system that uses projected visualizations to guide hand movements in MR applications[20]. This approach showcased the potential of combining visual cues with sensor data to create more intuitive user experiences. Liu et al. (2021) expanded on this by studying multi-modal fusion techniques to improve the accuracy and responsiveness of MR systems, focusing on integrating audio, visual, and depth data streams[11].

Collaborative MR environments have also been optimized for performance and usability. Hu et al. (2022) investigated real-time optimization strategies for collaborative MR systems, addressing challenges such as latency and resource management in multi-user scenarios[6]. Their findings contribute to the development of MR applications that support seamless and efficient interactions in shared virtual environments.

This thesis builds upon these foundational works by addressing the unique challenges of gesture recognition, multi-sensor integration, and collaborative functionalities for MR systems. By combining CNN-based gesture recognition with early exits, multi-sensor synchronization, and multi-user applications, this research aims to optimize resource utilization and pave the way for scalable and immersive MR experiences.

Chapter 3

Methodology

This chapter outlines the methodologies employed to design and develop a comprehensive mixed reality system incorporating gesture recognition, multi-sensor integration, and collaborative capabilities. By employing a structured approach, the aim was to optimize resource utilization while ensuring seamless and interactive user experiences. The following sections describe the system architecture, data preparation, model development, experimental setup, and evaluation strategies.

3.1 Gesture Recognition and Early Exits

3.1.1 Machine Learning Model and Early Exits

For gesture recognition, a convolutional neural network (CNN)-based model was developed, incorporating a sequential learning approach with multiple early exits. Sequential learning with early exits allows a model to produce intermediate outputs at different stages of the

network, enabling faster predictions when higher computational efficiency is needed. This approach is particularly beneficial in resource-constrained environments like wearable devices, as it reduces latency and optimizes energy consumption by bypassing deeper layers when intermediate outputs are sufficient [21]

The gesture recognition model is a convolutional neural network (CNN) designed with multiple convolutional blocks and integrated early exits for efficient and adaptive inference. The model processes grayscale images of size 150×150 as input, using four convolutional blocks to extract increasingly complex features. Each block consists of a convolutional layer, ReLU activation, and max-pooling for spatial dimensionality reduction. The first convolutional block employs a 5×5 kernel with 32 filters to capture low-level spatial features, such as edges and textures. Subsequent blocks use 3×3 kernels with increasing filter sizes (64, 96, and 96) to focus on finer and more localized patterns, enhancing feature representation as the network deepens. Early exits are integrated after the first three convolutional blocks, each consisting of fully connected layers ($256 \rightarrow 128 \rightarrow 10$) for intermediate predictions, enabling faster and more resource-efficient inference for simpler inputs. The final block refines the high-level features and outputs the most accurate predictions through fully connected layers in the final classifier. The early exits allow the model to adapt to varying computational resources and latency requirements, making it ideal for resource-constrained environments like wearable devices. The choice of kernel sizes, increasing filter depths, and strategic placement of early exits ensures a balance between computational efficiency and classification accuracy, enabling real-time gesture recognition for mixed reality applications.y by processing the input through the entire network. The detailed architecture of the model is shown in Fig 3.1. The full gesture recognition model architecture, including the initialization and forward pass logic, is provided in Listing A.1 and Listing A.2

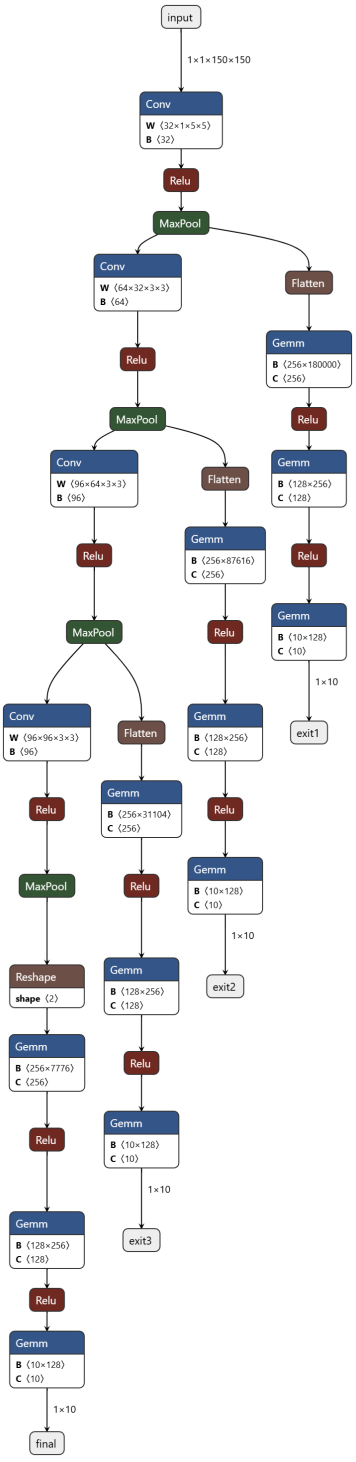


Figure 3.1: Gesture Recognition Model Architecture & Early Exits

3.1.2 Dataset & Preprocessing

The LeapGestRecog dataset [9], available on Kaggle, was chosen for training and evaluating the gesture recognition model due to its suitability for hand gesture recognition tasks. The dataset comprises grayscale images of hand gestures, classified into ten distinct categories. These categories represent various gestures, each captured under different lighting conditions and hand orientations, making the dataset diverse and robust for training a gesture recognition system.

The images in the dataset were preprocessed to ensure consistency and compatibility with the model's requirements. Each image was resized to a standard dimension of 150×150 pixels, simplifying computations and standardizing the input for the convolutional neural network (CNN). Additionally, pixel values were normalized to the range [0,1], ensuring numerical stability during training by preventing excessively large values from causing instability in the optimization process. Data augmentation techniques, such as rotations, flips, or color adjustments, were intentionally avoided to maintain consistency in the results across different training stages.

The LeapGestRecog dataset was divided into training and testing sets in a 75:25 ratio, ensuring a sufficient amount of data for training while retaining a substantial test set for evaluating model performance. The preprocessing pipeline involved converting the image data into tensors, making it suitable for use with PyTorch. The gesture labels, represented as class indices, were one-hot encoded to align with the requirements of the cross-entropy loss function used during training. This encoding ensures that the model predicts probabilities for each gesture class, which are then compared against the one-hot-encoded labels to compute the loss.

3.1.3 Model Training with Sequential Learning

The training process employed a sequential learning strategy with four distinct stages, each designed to optimize specific components of the network. This strategy aimed to improve convergence and efficiency by training only a subset of layers at a time while freezing the rest of the network to retain their previously learned parameters.

- **Stage 1**: The first convolutional layer and its corresponding early exit fully connected layers were trained. This stage focused on learning low-level spatial features from the input images, such as edges and textures. To ensure concentrated learning on this stage, all subsequent layers were frozen, preventing updates to their parameters.
- **Stage 2**: The second convolutional layer and the second early exit were trained. The previously trained first convolutional layer and early exit were frozen to retain their learned representations. This stage enabled the network to build on the foundational features extracted in Stage 1 and learn more complex patterns.
- **Stage 3**: The third convolutional layer and its corresponding early exit were trained. As with the earlier stages, the previously trained layers were frozen, ensuring the learning process for this stage focused solely on the targeted layer and its early exit.
- **Stage 4**: In the final stage, the entire network was fine-tuned, including the fourth convolutional layer and the output layers. This step ensured that all components of the network were optimized together, achieving maximum accuracy for complex gestures.

In the training process, a layer-freezing technique was employed to preserve the learned parameters of previously trained layers while optimizing the newly added ones. This strategy was critical in the staged training approach, where the model was trained sequentially, layer

by layer. After each stage, the trained layers were frozen using a custom function that set the `requires_grad` attribute of the layer parameters to `False`, preventing their weights from being updated during subsequent training stages. For instance, after the first stage, the first convolutional layer (`conv1`) and the first early exit fully connected layers (`exit0_fc`) were frozen. Similarly, after training the second stage, the second convolutional layer (`conv2`) and the second early exit (`exit1_fc`) were frozen, and so on. This ensured that the knowledge acquired by earlier layers was preserved, while the model focused on learning new patterns in the later layers[1]. The final stage fine-tuned the entire network by training the deeper layers and the final classifier, leveraging the frozen parameters of earlier layers to achieve optimal performance. This approach enhanced computational efficiency, reduced overfitting, and preserved the feature representations learned in earlier stages, making it ideal for gesture recognition tasks on resource-constrained devices.

Listing 3.1: Function to Freeze Layers

```
# Function to freeze layers
def freeze_layers(layers):
    for layer in layers:
        for param in layer.parameters():
            param.requires_grad = False
```

Separate optimizers were employed for each stage, tailored to the specific layers being trained. Cross-entropy loss was used as the objective function, as it is well-suited for multi-class classification tasks[23]. This stage-wise training strategy minimized overfitting by freezing previously trained layers and allowed the network to converge effectively at each stage

3.1.4 Evaluaution & Deployment

After completing the training process, the model was rigorously evaluated using the test dataset. Performance metrics were analyzed for the outputs from all three early exits and

the final classifier, providing insights into the model's accuracy and efficiency at different levels of the network. The inclusion of early exits enabled a detailed evaluation of the trade-offs between computational cost and prediction accuracy, ensuring the model could adapt dynamically to varying resource constraints.

The trained model was converted to the ONNX format to facilitate deployment on HoloLens 2. This format ensures compatibility with diverse runtime environments, such as mobile and augmented reality devices, by supporting optimized inference across various hardware configurations [17]. ONNX inference was conducted to validate the consistency of predictions between the PyTorch implementation and the ONNX runtime, confirming the reliability, robustness, and readiness of the model for real-world applications on resource-constrained devices like the Hololens 2.

3.1.5 ONNX Integration

The trained gesture recognition model was exported from PyTorch to the ONNX (Open Neural Network Exchange) format, which ensures compatibility across various platforms, including Unity and HoloLens 2. The ONNX format facilitates efficient deployment by enabling optimized inference in runtime environments such as Unity's Barracuda framework. This conversion ensured that the model could leverage the computational capabilities of the HoloLens 2 for real-time gesture recognition tasks.

In Unity, the ONNX model was integrated using the Barracuda library. The model was loaded as an NNModel asset and executed using Barracuda's IWorker interface, which handles the inference process. During runtime, frames captured from the HoloLens 2 camera were preprocessed to match the model's input requirements (grayscale, 150×150 resolution). These frames were then passed as tensors to the ONNX model for inference. The softmax

function was applied to the output logic to generate probabilities, and the class with the highest probability was selected as the predicted gesture.

The real-time inference results were logged, including predictions, timestamps, and FPS, to evaluate the model's accuracy and efficiency under deployment conditions. This setup ensured the seamless integration of gesture recognition capabilities into a mixed reality environment.

3.1.6 Experimental Setup

The experiments were designed to evaluate the performance and energy efficiency of the gesture recognition model in real-world conditions using HoloLens 2. The following components were integral to the experimental setup:

Hardware and Environment

- **Device**: HoloLens 2, equipped with a built-in camera for real-time gesture capture and sensors for monitoring battery consumption.
- **Input**: Hand gesture images were captured from a grayscale gesture video specifically created for the experiment. The video consisted of a gesture repeated cyclically for 20 minutes, ensuring sufficient data collection for each gesture. The model was evaluated on a dataset comprising 10 distinct gesture classes, ensuring comprehensive testing across a diverse range of hand movements. The frames were resized to 150×150 to match the input requirements of the model.

Software Framework

- **Unity Engine**: Used as the primary development platform to deploy and run the application on HoloLens 2.
- **Barracuda Framework**: Unity's inference engine for executing ONNX models, enabling efficient real-time predictions and integration with the HoloLens 2 ecosystem.

Data Flow and Collection

- **Frame Capture**: The HoloLens 2 camera continuously captured grayscale frames from the video during the 20-minute experimental window. This ensured consistent and high-quality data for gesture recognition inference.
- **Preprocessing**: Each captured frame was resized to 150×150 and converted into tensors compatible with the ONNX model. This preprocessing step ensured uniformity across the dataset for accurate inference.
- **Inference and Logging**: The preprocessed frames were passed through the ONNX model at various exit points. Metrics such as accuracy, latency (time per inference), and FPS (frames per second) were logged for each exit. These metrics provided insights into the model's computational efficiency and prediction quality.
- **Battery Level Monitoring**: The battery level of the HoloLens 2 was tracked throughout the 20-minute experiment for each of the three gestures. Separate trials were conducted for each exit:
  - First, predictions were made only at **Exit 1**.
  - Then, the process was repeated for **Exit 2**, **Exit 3**, and the **Final Layer**.

This sequential testing allowed for a detailed comparison of the rate of battery drain across different exit points, providing valuable data on the energy efficiency of the model.

Performance Metrics

To evaluate the model's effectiveness and efficiency, the following metrics were measured:

- **Accuracy**: The percentage of correctly identified gestures compared to the total predictions for each exit.
- **Latency**: The average time taken to process a single frame, including preprocessing and inference.
- **FPS**: The number of frames processed per second, indicating the model's ability to perform real-time predictions.
- **Battery Levels**: The rate of battery drain was analyzed for each exit point and gesture, allowing for a comprehensive understanding of energy efficiency under different operational conditions.

Training and Evaluation

The gesture recognition model was trained on the LeapGestRecog dataset using a staged training approach, optimizing different parts of the network sequentially. During evaluation, predictions were made using each exit in isolation to study its impact:

- **Exit 1**: Early exit after the first convolutional layer.

- **Exit 2**: Early exit after the second convolutional layer.
- **Exit 3**: Early exit after the third convolutional layer.
- **Final Layer**: Output after processing through the entire network.

This approach provided insights into the trade-offs between computational cost, prediction accuracy, and energy efficiency at different depths of the network.

3.2 Sensory Profiling

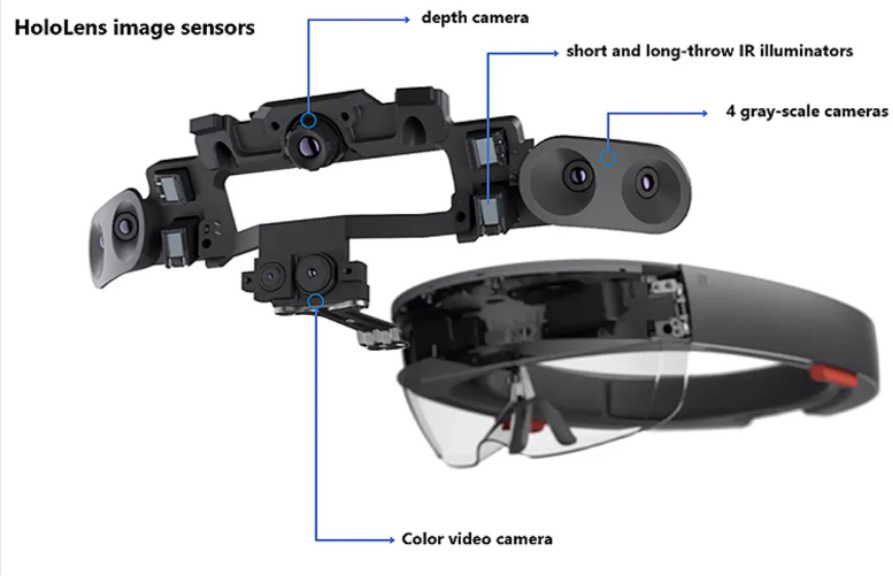


Figure 3.2: Hololens 2 Sensors

The HL2SS (HoloLens 2 Streaming Server) enables access to raw data from key sensors on the HoloLens 2, providing valuable inputs for research and development in multi-user mixed reality applications[5]. The supported sensors include:

- **Visible Light Camera**: Captures real-world images at 30 frames per second (FPS), providing essential inputs for scene understanding, visual feeds, and contextual analysis

- **Depth Camera & Long-Throw Depth Camera**: These cameras enable high-resolution spatial mapping with sampling rates ranging from 5–45 FPS and 1–5 FPS, respectively, depending on the configuration. They are essential for applications requiring accurate depth perception, such as object placement and environmental scanning.

- **Microphones**: Record high-quality audio at a 48 kHz sampling rate, supporting tasks like voice recognition, sound localization, and acoustic analysis.

- **Inertial Measurement Unit (IMU)**: Comprising an accelerometer, gyroscope, and magnetometer, the IMU captures motion and orientation data at 12 Hz, enabling robust tracking for navigation and interaction.

- **Eye Tracking (EET)** : Monitors gaze direction and focal attention with sampling rates of 30 Hz, 60 Hz, or 90 Hz, facilitating precise interaction, intent detection, and user experience personalization.

Table 3.1 provides a detailed summary of the sensors available on the HoloLens 2, along with their documented frame rates (FPS) and sampling rates.

Table 3.1: Documented FPS and Sampling Rates for HoloLens 2 Sensors

Sensor	**Frame Rate (FPS)**	**Sampling Rate (Hz)**
Visible Light Camera	30 FPS	-
Depth Camera	5–45 FPS	-
Long-Throw Depth Camera	1–5 FPS	-
Microphones	-	48,000 Hz
Inertial Measurement Unit (IMU)	-	12 Hz
Eye Tracking (EET)	30, 60, or 90 FPS	-

These sensors collectively form the foundation for advanced spatial mapping, real-time user interaction, and environmental analysis in HL2SS-enabled multi-user MR applications.

3.2.1 Challenges in Sensor Integration

Integrating multiple sensors in the HoloLens 2 for mixed reality applications poses several challenges, particularly when three or more sensors are used simultaneously. High data transmission rates and computational demands can result in:

- **System Bottlenecks**: The concurrent use of multiple high-frequency sensors, such as the visible light camera, eye-tracking sensor, and depth cameras, can strain the device's processing capabilities. This often leads to reduced sampling rates, degraded performance, and increased latency.
- **Synchronization Issues**:In multi-user collaborative scenarios, ensuring consistency and synchronization across devices is critical. However, the processing delays introduced by simultaneous data streams can affect the alignment of sensor outputs, disrupting the collaborative experience.
- **Increased Latency**: Higher computational demands can lead to delays in rendering or interaction, directly impacting the responsiveness and immersion of the MR environment.

By systematically analyzing these challenges, we identified areas where optimization is required, particularly in scenarios that demand real-time performance. The insights gained from this study were instrumental in designing strategies to mitigate these challenges and enhance multi-sensor integration in HL2SS-enabled MR applications.

3.2.2 Experimental Study

Single Sensor Testing

To evaluate the baseline performance of different sensors supported by the HL2SS server, a simple application was developed in Unity. Each sensor was tested independently to measure its frame rate or sampling rate under standard conditions. The application was designed to capture and log data from the visible light camera, depth cameras (AHAT and long-throw), microphones, IMU, and eye-tracking system. Observations were made regarding the system's behavior, such as how immediate data saving impacted performance and challenges like audio noise removal. These tests were conducted to establish a baseline for single-sensor performance before integrating multiple sensors. The performance of individual sensors was evaluated to establish a baseline before integrating multiple sensors. The results, including measured performance and key observations, are summarized in Table 4.8.

Dual Sensors Testing

After evaluating single sensors, the next step was to combine two sensors to analyze their combined performance and compatibility. This involved streaming data simultaneously from two sensors using the HL2SS server and observing the system's behavior. The application was modified to support dual-sensor configurations, and performance metrics such as frame rates, sampling rates, and system responsiveness were measured.

Each possible combination of two sensors was tested systematically, such as the camera and IMU, microphone and AHAT, and visible light camera with long-throw depth camera. Observations were recorded regarding successful data capture, frame rates, and any limitations encountered. For instance, while simpler combinations like the camera and microphone

worked seamlessly, others, such as the AHAT depth camera and IMU, showed reduced frame rates or synchronization issues.

This stage provided valuable insights into how sensor pairing affects system performance and laid the groundwork for further studies involving multiple sensor integration. Detailed results from these experiments, along with key observations, are presented in the Results and Discussion section (refer to Table 4.9).

Three Sensor Testing

After evaluating single and dual-sensor configurations, the next step was to test the performance of three-sensor combinations to understand the system's capabilities and limitations. During the tests, performance metrics such as frame rates, sampling rates, and responsiveness were recorded. Observations were also made regarding issues such as noise, synchronization, and computational bottlenecks. The goal was to assess how well the system managed the increased data throughput and identify combinations that maintained acceptable performance levels. Detailed results from these experiments, along with key observations, are presented in the Results and Discussion section (refer to Table 4.10).

To explore the practical use of these sensors in a real-world scenario, we developed a mixed reality multi-user application that collects and utilizes data from the camera and microphone. This application serves as a testbed to evaluate the effectiveness of these sensors in facilitating immersive and interactive experiences, emphasizing their role in capturing visual and audio inputs for collaborative environments.

3.3 Multi-user application

Multi-user capabilities serve as the foundation for collaborative (MR) applications, enabling participants to interact and engage within a shared virtual environment seamlessly. This section details the approach used to develop and optimize a collaborative MR system for HoloLens 2, with a focus on achieving synchronization, real-time interaction, and resource efficiency—all while ensuring an immersive user experience.

Figure 3.3: Multiuser Shared Experience

The project centered around the creation of an Image Sorting Game, a collaborative MR application, leveraging the capabilities of the latest Mixed Reality Toolkit 3 (MRTK3)—Microsoft's framework for building immersive MR experiences in Unity. The development methodology adhered to best practices outlined in the Mixed Reality Sharing tutorial for Unity [6], ensuring a streamlined and scalable implementation.

The subsequent subsections provide a detailed breakdown of the development process, offering clarity on the various stages and techniques employed in creating the game. These include the design of multi-user interactions, synchronization mechanisms, optimization strategies, and user experience enhancements that make the collaborative MR application robust and engaging.

3.3.1 System Design

This section outlines the hardware and software components integrated into the collaborative mixed reality (MR) environment, focusing on the HoloLens 2 sensors and their application within the MR Image Sorting Game.

Collaborative Setup

To facilitate seamless multi-user collaboration, the system employs Photon Unity Networking (PUN) and Azure Spatial Anchors for sharing object movements and interactions across users in the MR environment:

- **Photon Unity Networking (PUN)**: PUN, a networking framework tailored for multi-user collaboration, ensures consistency by synchronizing all game components across participants [18]. It supports real-time interactions, maintaining a uniform experience for all users.

- **Azure Spatial Anchors**: Azure Spatial Anchors, a cloud-based service from Microsoft Azure, enable the creation and sharing of spatial anchors that accurately position virtual objects in a shared space [13]. These anchors are essential for maintaining alignment and consistency in the virtual environment.

In this application, both PUN and Azure Spatial Anchors are integrated to create, update, and synchronize game objects across users, providing a robust collaborative setup [6].

HoloLens 2 Configuration

To develop the collaborative application and harness the capabilities of the HoloLens 2, the following configurations are essential:

- Developer Mode: HoloLens devices are set to Developer Mode, enabling the deployment of the application from a Windows machine to the device.

- Research Mode: Research Mode, designed for academic and experimental purposes, is enabled on the HoloLens 2 to access raw sensor data [12]. This mode provides data from various sensors, including:
  - Visible Light Camera
  - RM Depth Camera
  - Long-Throw Depth Camera
  - Microphone
  - Inertial Measurement Unit (IMU)
  - Eye Tracking

The combination of Developer Mode and Research Mode ensures the app can both operate effectively on HoloLens devices and leverage their advanced sensory capabilities.

Data Collection Setup

To facilitate the collection of sensory data from the HoloLens 2, the system utilizes HL2SS—a server application specifically designed for the HoloLens 2 that streams sensory data in real-time via TCP[5].

By integrating the HL2SS plugin into the collaborative Image Sorting Game, the application can seamlessly stream sensory data from the HoloLens 2. This setup allows for efficient data collection, enabling analysis of how the integration of various sensors impacts the system's sampling rate. This not only supports real-time data streaming but also provides valuable insights into optimizing sensor usage in collaborative MR applications.

3.3.2 Implementation

In this section, we will discuss the hardware and software tools used to implement the collaborative image sorting game and collect data. The software tools and hardware employed for game development and data collection include Unity, MRTK3, OpenXR, Visual Studio, HL2SS, HoloLens 2, and a Windows 10 PC. This section will further discuss the details of the app implementation and data collection processes.

Collaborative Application

The collaborative app was inspired by the work of Yang et al [25]. The MR application involves multiple users collaborating in the same physical space to sort 15 holographic images based on emotion labels. To develop this application, the development machine must run either Windows 10 or 11 and be capable of supporting Unity and the Universal Windows

Platform (UWP). It should also have Visual Studio installed with the complete .NET framework and C++ desktop development tools[16]. For our development, we used the Mixed Reality Feature Tool by Microsoft to import essential packages such as Azure Spatial Anchors, OpenXR, MRTK3, and PUN2 into Unity. The specific Unity version used for the app development was 2022.3.10f1. The game features 7 emotion labels: Gloomy, Frustrated,

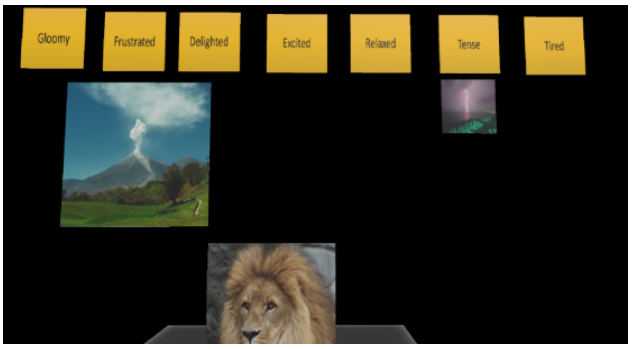


Figure 3.4: Collaborative Mixed Reality Setup

Delighted, Excited, Relaxed, Tense, and Tired, derived from Russell's model [19]. The 15 holographic images used in the game were randomly selected from the Open Affective Standardized Image Set (OASIS) dataset[10]. The collaborative mixed reality setup, illustrated in Figure 3.5, includes seven labeled categories representing emotions alongside a selection of random images for sorting.

Data Collection

To facilitate data collection, HL2SS was seamlessly integrated into the collaborative Image Sorting Game. During application runtime on the HoloLens 2, sensor data, including camera and audio streams, was captured in real-time without compromising the app's performance. The camera sensor of Hololens 2 operates at **30 FPS** and microphone at **48kHZ**.

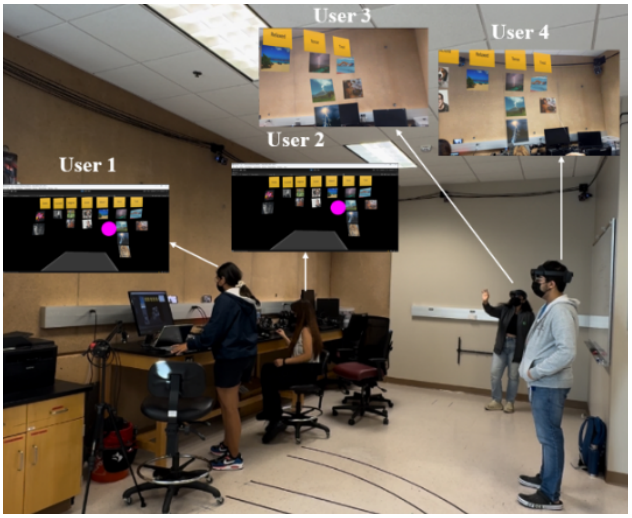


Figure 3.5: Collaborative Setup with 4 Users

3.3.3 Experiment

To assess the effectiveness of the mixed reality multi-user image sorting game, a user study was conducted involving four participants. Two participants utilized the HoloLens 2 device, while the remaining two engaged with the application using the Unity Mixed Reality Emulator.

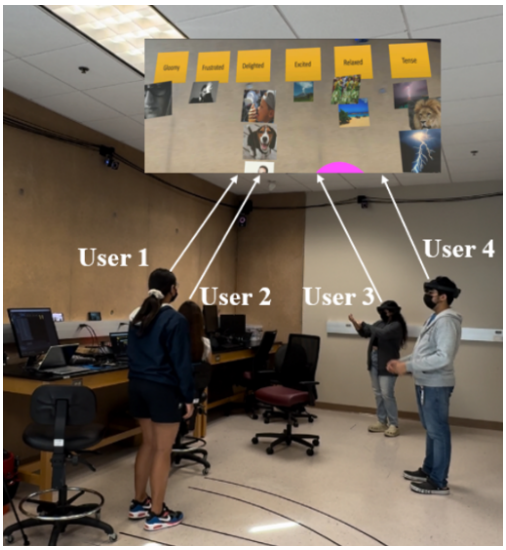


Figure 3.6: Shared Hologram

This setup allowed for a diverse evaluation, testing the compatibility and usability of the game across different platforms. As illustrated in Figure 3.5, the users interacted collaboratively in a shared virtual environment, demonstrating the potential for multi-platform integration. The core feature of the game was the shared hologram, which served as the central focus of interaction and was visible to all participants, regardless of the platform they used. This shared hologram facilitated seamless collaboration among users, as depicted in Figure 3.6. The study provided valuable insights into the application's performance, user experience, and the effectiveness of shared holographic content in enabling immersive and collaborative gameplay.

Chapter 4

Results & Discussion

4.1 Gesture Recognition and Early Exits

The gesture recognition model was evaluated on a dataset consisting of 10 distinct gestures, capturing performance across multiple metrics:

- **Accuracy**: Indicates the model's classification capability for each gesture.
- **Battery Drain**: Highlights the energy consumption for processing each gesture.
- **Average FPS (Frames Per Second)**: Measures the real-time performance of the model.
- **Time Per Inference**: Reflects the computational latency for each prediction.

These metrics were analyzed individually for each gesture, providing a detailed understanding of the model's behavior and trade-offs between performance and resource efficiency.

Gesture 1	Gesture 2
	
Gesture 3	**Gesture 4**
	
Gesture 5	**Gesture 6**
	
Gesture 7	**Gesture 8**
	
Gesture 9	**Gesture 10**
	

Table 4.1: Different Gesture Images

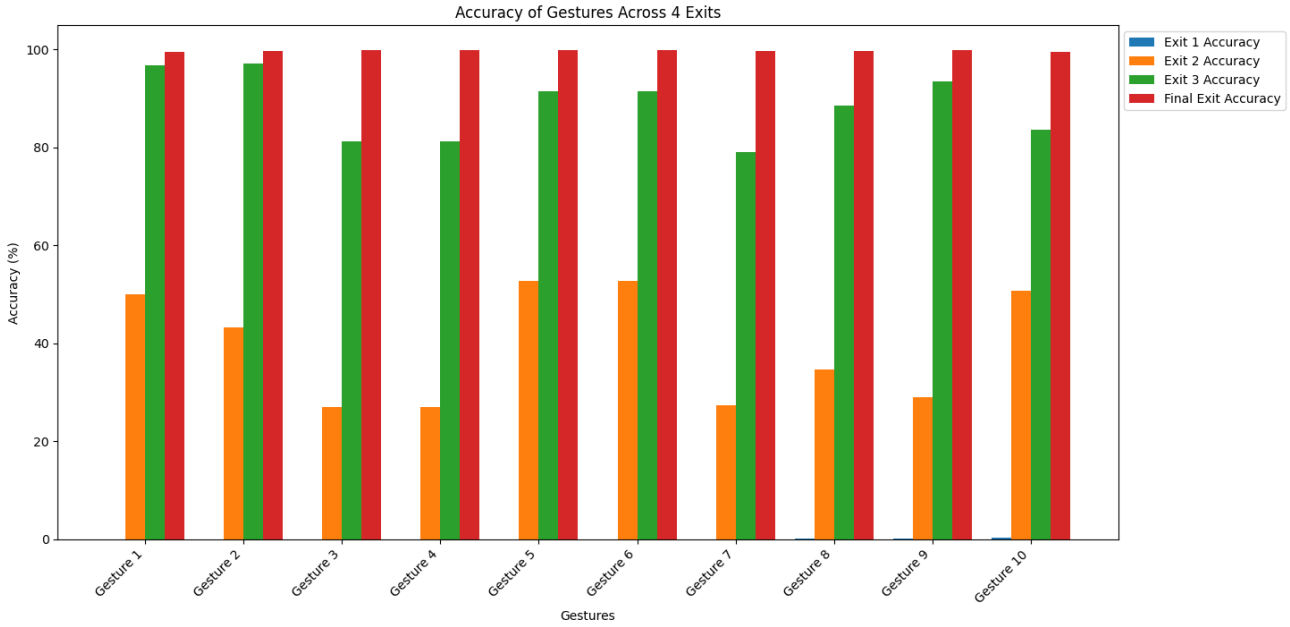


Figure 4.1: Accuracy of Prediction for Gesture 1- 10

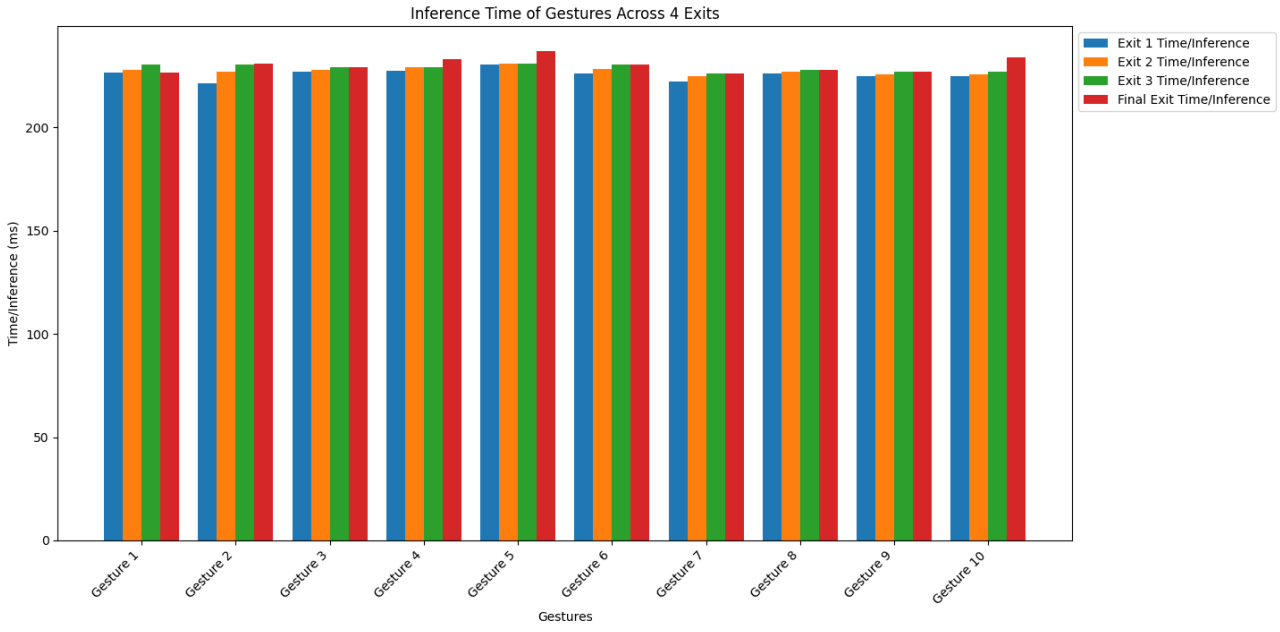


Figure 4.2: Time/ Inference for Gesture 1- 10

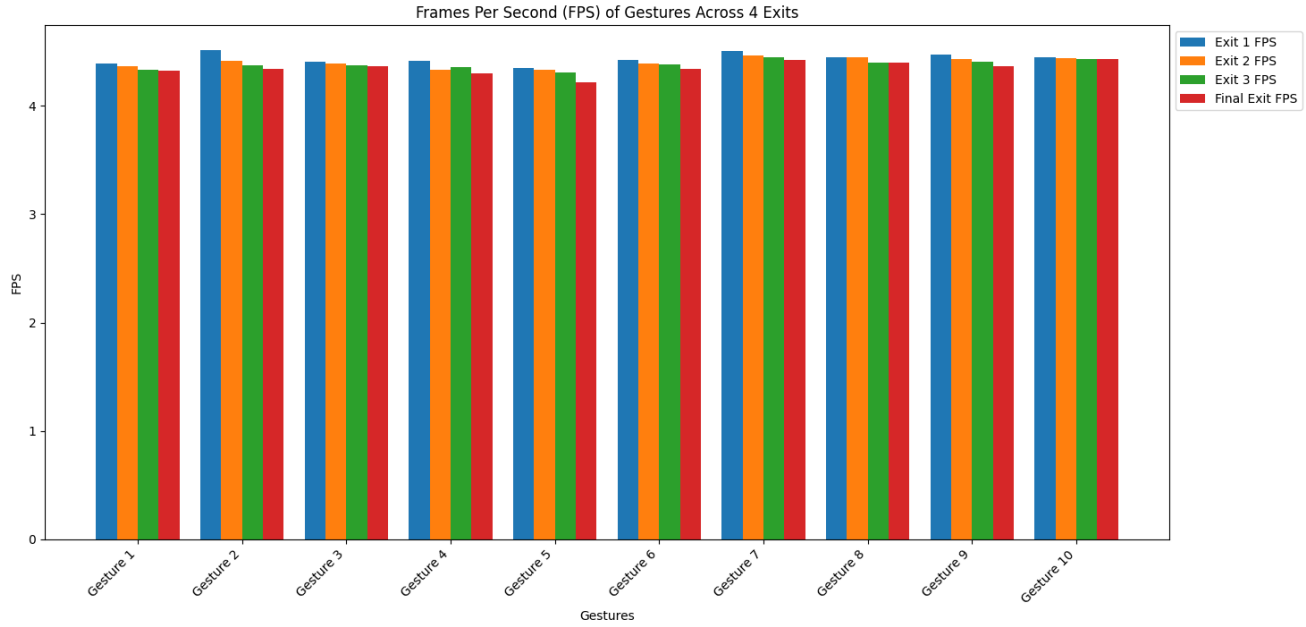


Figure 4.3: FPS for Gesture 1- 10

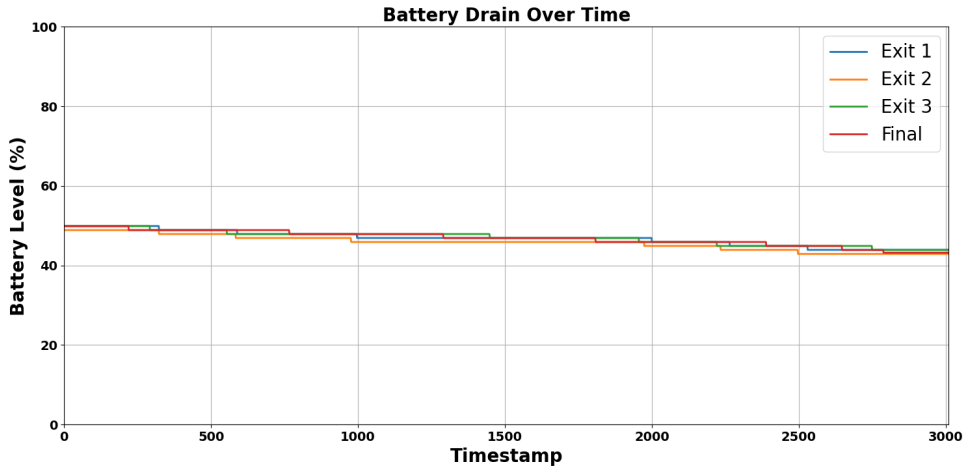
Gesture	**Battery Drain Across Different Exits**
Gesture 1	

Table 4.2: Battery Drain Across Different Exits for Gesture 1.

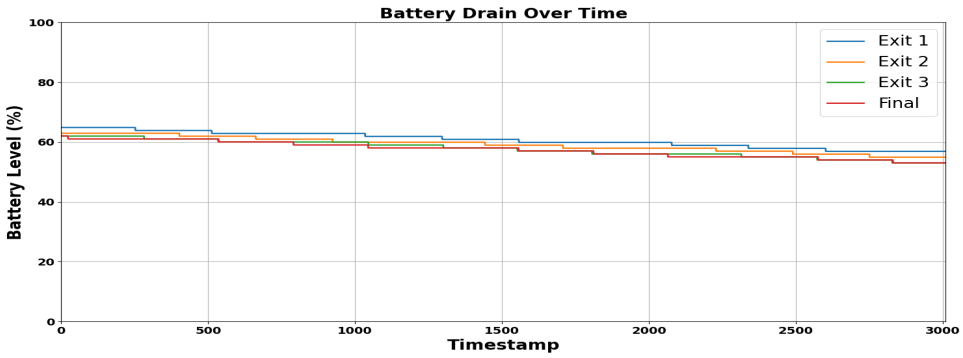
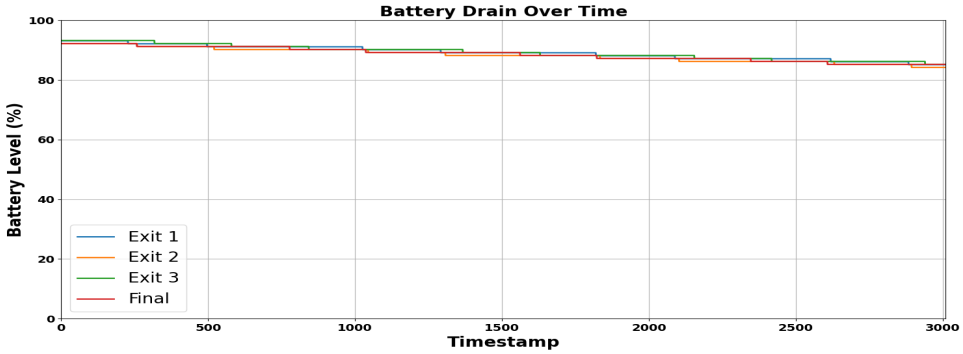
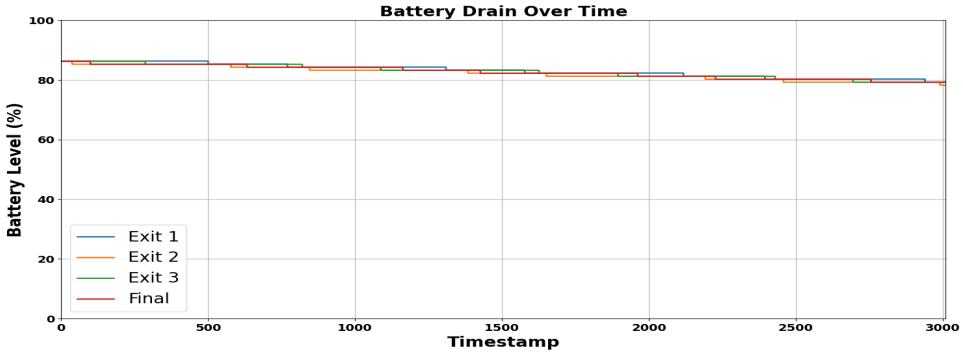
Gesture	Battery Drain Across Different Exits
Gesture 5	
Gesture 6	
Gesture 7	

Table 4.3: Battery Drain Across Different Exits for Gestures 5-7.

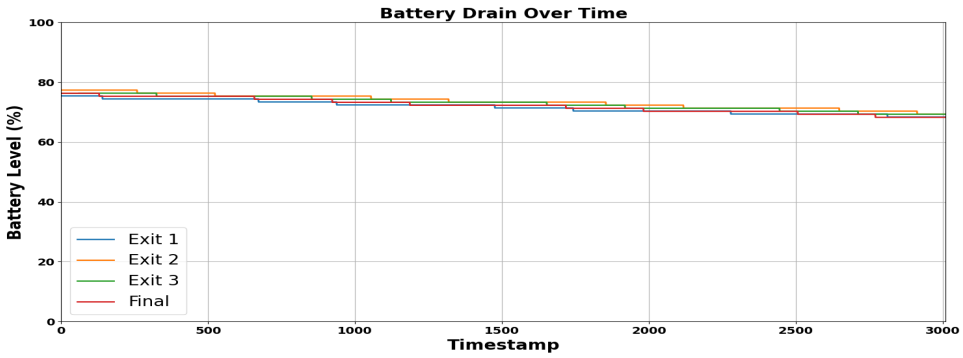
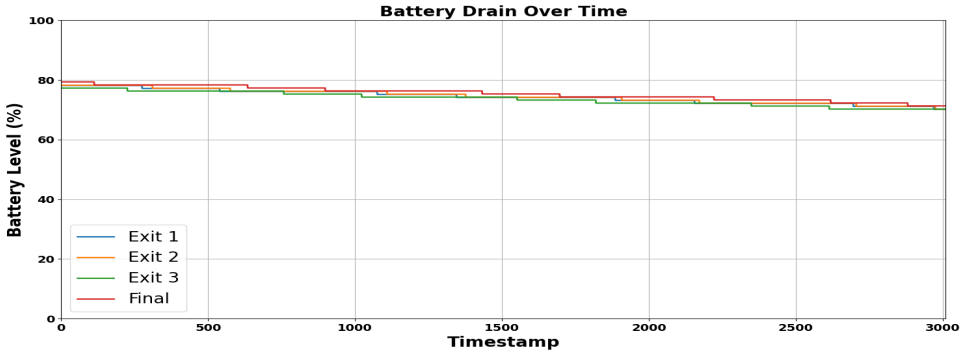
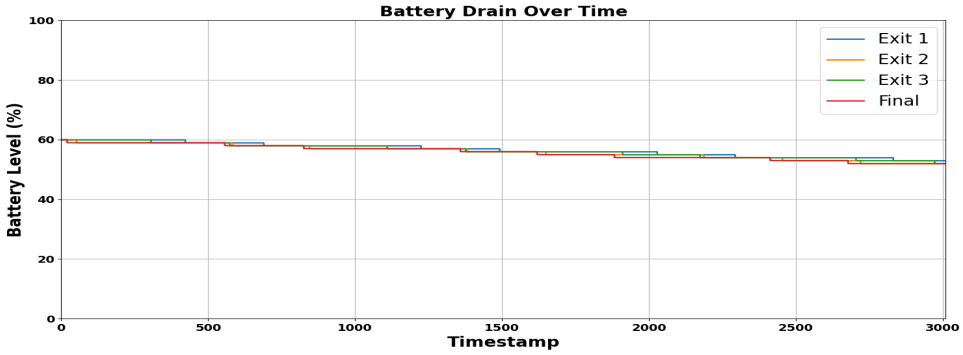
Gesture	Battery Drain Across Different Exits
Gesture 8	
Gesture 9	
Gesture 10	

Table 4.4: Battery Drain Across Different Exits for Gestures 8-10.

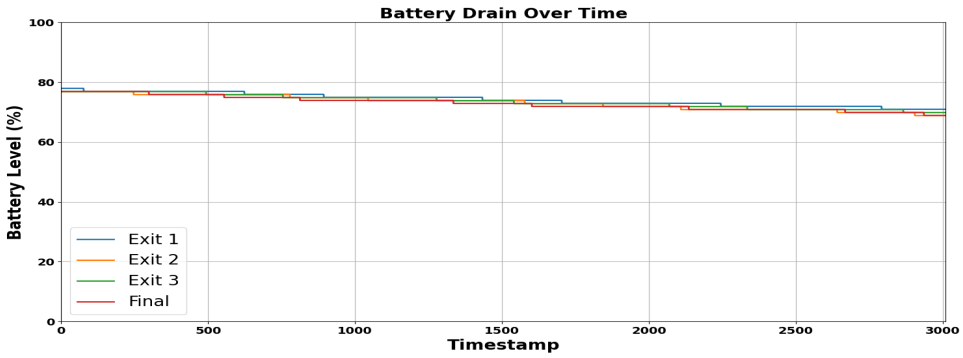
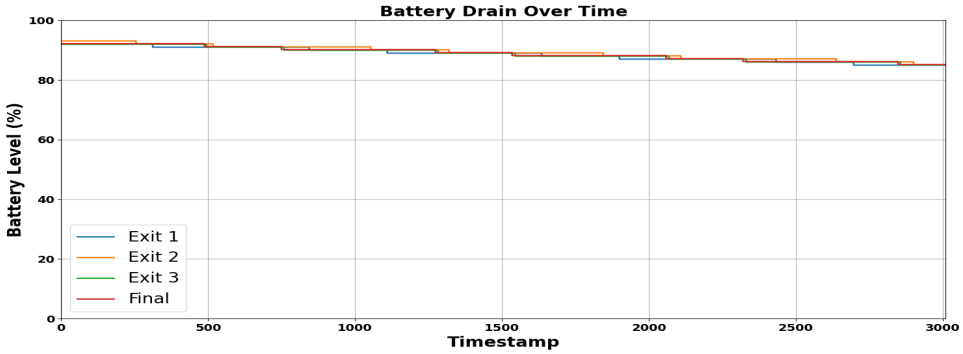
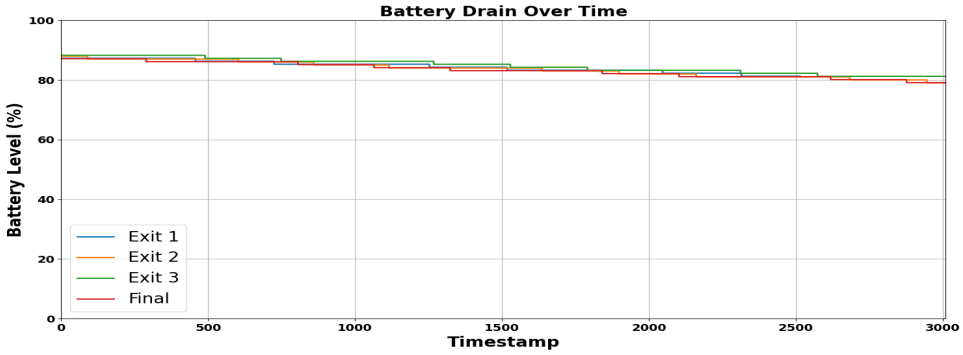
Gesture	Battery Drain Across Different Exits
Gesture 2	
Gesture 3	
Gesture 4	

Table 4.5: Battery Drain Across Different Exits for Gestures 2-4.

Table 4.6: Performance Metrics Across Exits for Gestures 1 to 6

Gesture	**Metric**	**Exit 1**	**Exit 2**	**Exit 3**	**Final Exit**
Gesture 1	Average Accuracy (%)	0.03	50.06	96.83	99.56
	FPS	4.391	4.363	4.332	4.324
	Time/Inference (ms)	226.45	227.63	230.66	231.32
	Battery Drain (%)	12	12	12.25	13.37
Gesture 2	Average Accuracy (%)	0	43.20	97.05	99.60
	FPS	4.517	4.417	4.372	4.341
	Time/Inference (ms)	221.55	226.95	230.66	230.67
	Battery Drain (%)	8.97	9.09	10.40	10.42
Gesture 3	Average Accuracy (%)	0.02	27.01	80.06	99.94
	FPS	4.404	4.392	4.370	4.363
	Time/Inference (ms)	227.18	227.80	229.08	229.31
	Battery Drain (%)	7.60	7.60	7.61	8.59
Gesture 4	Average Accuracy (%)	0	27.10571	81.2575	96.129
	FPS	4.412	4.335	4.325	4.302
	Time/Inference (ms)	227.25	228.95	231.30	232.95
	Battery Drain (%)	7.97	7.97	9.21	10.21
Gesture 5	Average Accuracy (%)	0	52.75	91.43	99.88
	FPS	4.345	4.340	4.237	4.216
	Time/Inference (ms)	230.51	230.83	236.19	237.12
	Battery Drain (%)	12.32	12.71	14.48	14.52
Gesture 6	Average Accuracy (%)	0	37.34	94.78	99.85
	FPS	4.423	4.386	4.363	4.343
	Time/Inference (ms)	226.24	228.12	229.30	230.47
	Battery Drain (%)	8.59	8.61	8.68	8.69

Table 4.7: Performance Metrics Across Exits for Gestures 7 to 10

Gesture	**Metric**	**Exit 1**	**Exit 2**	**Exit 3**	**Final Exit**
Gesture 7	Average Accuracy (%)	0	27.22	79.02	99.70
	FPS	4.507	4.462	4.451	4.426
	Time/Inference (ms)	222.03	224.69	224.74	226.14
	Battery Drain (%)	8.14	8.14	8.15	8.21
Gesture 8	Average Accuracy (%)	0.159	34.59	88.50	99.71
	FPS	4.448	4.424	4.404	4.394
	Time/Inference (ms)	224.95	226.20	227.64	227.76
	Battery Drain (%)	9.19	9.33	10.40	10.52
Gesture 9	Average Accuracy (%)	0.159	29.06	93.42	98.78
	FPS	4.471	4.427	4.420	4.376
	Time/Inference (ms)	224.25	226.06	226.45	228.62
	Battery Drain (%)	9.10	10.13	10.25	10.26
Gesture 10	Average Accuracy (%)	0.326	50.76	83.60	99.51
	FPS	4.450	4.439	4.428	4.424
	Time/Inference (ms)	224.97	225.76	225.98	226.13
	Battery Drain (%)	11.66	13.33	13.34	13.35

After analyzing the data collected over 18–20 minutes, a clear trend emerged regarding the performance of the final exit. It was observed that the final exit significantly increases battery consumption, with a notably higher average processing time compared to earlier exits. This pattern was consistent across most predictions.

The accuracy of predictions improved steadily from Exit 1 to the final exit, demonstrating the benefits of deeper layers in achieving more precise results. However, this improvement

comes at a cost, as the camera FPS gradually decreased when transitioning from Exit 1 to the final exit. Additionally, the average processing time exhibited a consistent increase, further emphasizing the trade-off between computational efficiency and prediction accuracy as the model progresses through its layers.

4.2 Sensory Profiling

4.2.1 Single Sensor Performance

The single-sensor performance results showed that most sensors performed near or at their documented specifications when used independently. The observations and results for each sensor are summarized in Table 4.8. For instance, the visible light camera achieved a measured frame rate of 29.16 FPS, close to the documented 30 FPS, while the AHAT depth camera had a measured frame rate of 39.06 FPS compared to the expected 45 FPS due to high-resolution processing demands. Key challenges included noise removal for the microphone data and latency when saving frames in real time for the visible light camera:

Table 4.8: Performance Evaluation of Individual Sensors

Sensor	**Measured Performance**	**Documented Performance**	**Observations**
Camera (Visible Light)	29.16 FPS	30 FPS	Immediate frame saving reduced FPS to 10, but deferred saving restored it to 29.16 FPS.
Microphone	48,000 Hz	48,000 Hz	Sampling rate matched the documentation; however, noise removal from saved audio remains a challenge.
Depth AHAT Camera	39.06 FPS	45 FPS	Measured FPS was slightly lower than documented due to high-resolution processing demands.
Depth Long-Throw (LT) Camera	4.81 FPS	1–5 FPS	Measured FPS aligned with the documented range, as this is a low-frequency sensor by design.
Visible Light Camera (VLC)	29.08 FPS	30 FPS	Measured performance was consistent with the documented specification.
IMU (Accelerometer)	12 Hz	12 Hz	Sampling rate matched the documented specification without noticeable deviations.
Eye Tracking (EET)	30 FPS	30, 60, or 90 FPS	Configured and tested at 30 FPS, one of the supported options as per the documentation.

These findings provided a strong foundation for understanding single-sensor behavior and informed the design of subsequent experiments involving multi-sensor integration.

4.2.2 Dual Sensor Performance

To evaluate dual-sensor combinations, each pair of sensors was tested systematically, and the results are summarized in Table 4.9.

Table 4.9: Performance Evaluation of Two-Sensor Combinations

Camera	Microphone	AHAT	Long-Throw	VLC	IMU	Eye Tracking	Comments
✓	✓						Camera FPS = 25, Microphone Sampling Rate = 48,000 Hz.
	✓	✓				✓	IMU FPS = 11.72 Hz, Eye FPS = 30.
		✓			✓	✓	Microphone Sampling Rate = 48,000 Hz, IMU Sampling Rate = 11.8 Hz.
		✓				✓	Microphone Sampling Rate = 48,000 Hz, Eye FPS = 30.
✓					✓		Camera FPS = 29.5, IMU Sampling Rate = 48,000 Hz.
✓							No noticeable issues.
	✓		*Not Compatible with AHAT*			✓	AHAT FPS = 4.2 FPS and IMU Sampling Rate = 48,000 Hz.
	✓		*Not Compatible with AHAT*		✓		IMU FPS = 11.7 Hz, AHAT FPS = 4.1 FPS.
✓	✓		*Not Compatible with AHAT*				Microphone Sampling Rate = 48,000 Hz, AHAT FPS = 4.2 FPS.
✓	✓		*Not Compatible with AHAT*		✓		Camera FPS = 20, AHAT FPS = 3.8 FPS.
		Not Compatible with LT		✓		✓	Long-Throw Depth FPS = 4 FPS, IMU Sampling Rate = 11.8 Hz.
		Not Compatible with LT		✓			Eye FPS = 30, Long-Throw Depth FPS = 4.1 FPS.
✓		*Not Compatible with LT*				✓	Long-Throw Depth and Microphone data integration is challenging.
✓		*Not Compatible with LT*		✓			Long-Throw Depth and Visible Light Camera data integration is challenging.

The evaluation of dual-sensor combinations revealed several key findings. Most configurations, such as the combination of the visible light camera and microphone, performed seamlessly, maintaining stable performance metrics, such as a camera frame rate of 25 FPS and a microphone sampling rate of 48,000 Hz. Similarly, combinations like the IMU with the microphone or eye tracking demonstrated minimal performance impact. However, certain combinations, such as the AHAT depth camera with the IMU or microphone, showed reduced performance, with the AHAT frame rate dropping to around 4.2 FPS due to computational demands. While the AHAT and Long-Throw Depth cameras could not function

together due to their similar roles as depth sensors, this limitation is expected and not a significant concern for typical applications[24]. The long-throw depth camera, when combined with other high-frequency sensors such as the visible light camera, exhibited reduced responsiveness and synchronization issues, likely due to high-resolution data output. Observations indicated that simpler sensors like the IMU and eye tracking integrated more effectively, whereas combinations involving high-resolution or high-frequency sensors often faced bottlenecks. These findings highlight the importance of selecting compatible sensors and optimizing system resources in mixed reality applications. They also emphasize the need for advanced strategies, such as adaptive sampling or prioritized data handling, to support seamless multi-sensor integration. These insights provide a foundational understanding of the system's limitations, paving the way for future studies involving more complex multi-sensor setups.

4.2.3 Three Sensor Performance

To evaluate three-sensor combinations, each pair of sensors was tested systematically, and the results are summarized in Table 4.10. The evaluation revealed several important findings regarding performance and system limitations. Combinations involving lower-demand sensors, such as the microphone, IMU, and eye tracking, performed reliably, with minimal

Table 4.10: Performance Evaluation of Three-Sensor Combinations

Camera	Microphone	AHAT	Long-Throw	VLC	IMU	Eye Tracking	Comment
✓	✓	✓	Won't work with AHAT				Camera: 20 FPS, Microphone: 45 kHz, AHAT: 2 FPS, noise removal is hard.
✓	✓					✓	Camera: 25 FPS, Microphone: 45 kHz, Eye Tracking: 30 FPS, noise removal is hard.
✓	✓				✓		Camera: 20 FPS, Microphone: 45 kHz, IMU: 9 Hz, reduced responsiveness.
✓	✓		Won't work with AHAT		✓		Camera: 18 FPS, Microphone: 45 kHz, IMU: 10.2 Hz, synchronization is slow.
✓	✓		Won't work with AHAT			✓	Camera: 20 FPS, AHAT: 2.5 FPS, noise removal is challenging.
	✓	Won't work with Long			✓	✓	Microphone: 48 kHz, IMU: 11.5 Hz, Eye Tracking: 30 FPS, no major issues.

performance degradation. For instance, the combination of the microphone, IMU, and eye tracking achieved stable metrics, with the microphone sampling at 48 kHz, the IMU at 11.5 Hz, and the eye tracking at 30 FPS, with no significant issues. However, combinations involving high-resolution sensors like AHAT or the long-throw depth camera encountered challenges. For example, the combination of the camera, microphone, and AHAT resulted in a reduced AHAT frame rate of 2 FPS and noise removal challenges in the microphone data. Similarly, the long-throw depth camera faced incompatibility issues when paired with AHAT due to overlapping computational demands. Configurations involving multiple high-resolution sensors were observed to be slower and less responsive, emphasizing the system's bandwidth and processing constraints. These results highlight the importance of selecting compatible sensors and optimizing system resources, particularly in real-time mixed reality applications, where performance bottlenecks can impact user experience and responsiveness.

4.3 Multi-user application

The collaborative Image Sorting Game was successfully developed and tested with a group of four participants. Two participants used HoloLens 2 devices, while the other two used the HoloLens 2 emulator within Unity. The gameplay experience was seamless across both setups, with all users able to interact with the shared virtual environment effectively. Real-time synchronization and multi-user collaboration were achieved without any noticeable delays or performance issues.

Table 4.11: Data Collection Specifications During Testing Phase

User Device	**Camera Frame Rate (FPS)**	**Audio Sampling Rate (kHz)**
HoloLens 2	25 FPS	48 kHz
Unity Emulator	30 FPS	44.1 kHz (via mobile phone)

Data collection during the testing phase further validated the system's functionality. For users operating the HoloLens 2, the HL2SS plugin streamed and recorded camera data at **25 FPS** and audio data at **48 kHz**, providing high-quality inputs for analysis. In contrast, users using the emulator relied on mobile phones for audio and video data collection, with camera information recorded at **30 FPS**. This dual setup ensured that comprehensive data was gathered across different operational environments, enabling a thorough evaluation of the game's performance and usability.

As discussed in the previous sections, the application has the potential to integrate additional sensors, such as the eye tracker, to further enhance the multi-user collaborative experience. For instance, incorporating the depth camera could enable more accurate spatial mapping and object positioning, improving alignment and interaction consistency across multiple users. However, integrating sensors like the depth camera, especially in combination with others such as the camera and microphone, may introduce performance challenges as seen in Table 4.10. The increased data processing requirements could impact system responsiveness and sampling rates, as highlighted earlier. Optimizing these sensor combinations to maintain seamless performance in multi-user environments is a promising area for future research and development, paving the way for more advanced collaborative mixed reality applications.

Chapter 5

Conclusion & Future work

This study explored three key aspects of leveraging the HoloLens 2 platform: gesture recognition, sensor integration, and multi-user applications. The gesture recognition model demonstrated robust performance, with accuracy steadily increasing from early exits to the final layer, achieving a peak accuracy of **99.56%**. However, this came at the cost of higher battery drain, increased inference time, and reduced FPS. The use of early exits proved effective for balancing computational efficiency and accuracy, making the model suitable for real-time applications in resource-constrained environments.

The sensor integration analysis provided valuable insights into the capabilities and limitations of single and multi-sensor configurations. While individual sensors generally met their documented specifications, dual and three-sensor combinations revealed performance bottlenecks, particularly when high-frequency or high-resolution sensors like the AHAT and Long-Throw Depth cameras were involved. These findings emphasize the need for careful sensor selection and system optimization to maintain real-time responsiveness and data quality in mixed reality applications.

The multi-user application demonstrated the potential for collaborative mixed reality experiences. The image sorting game allowed seamless interaction and synchronization between users on HoloLens 2 and Unity Emulator, validating the system's functionality in a multi-device environment. Despite the success, integrating additional sensors in collaborative setups highlighted challenges related to bandwidth and latency, underscoring the complexity of real-time multi-sensor, multi-user systems.

The integration of gesture recognition, sensor capabilities, and multi-user collaboration on the HoloLens 2 forms a cohesive framework for advancing interactive mixed reality applications. Gesture recognition provides an intuitive and natural interface for user interaction, while the use of early exits in the model ensures adaptability to varying computational and energy constraints. Sensor integration enhances the system's capability to perceive and respond to environmental cues, such as spatial mapping and gaze tracking, enabling more immersive experiences. When extended to multi-user applications, these capabilities facilitate seamless collaboration in shared virtual environments, allowing synchronized interaction and consistent user experiences. Together, these components address key challenges in mixed reality, such as real-time responsiveness, energy efficiency, and synchronization, paving the way for scalable and versatile systems that merge individual user needs with collaborative functionality. This synergy between gesture-based interaction, multi-sensor integration, and multi-user collaboration underscores the potential for HoloLens 2 to serve as a robust platform for diverse real-world applications.

Building on the findings of this study, future work will focus on enhancing the system's efficiency, scalability, and versatility. For gesture recognition, dynamic exit strategies will be explored to optimize computational efficiency by adaptively selecting exits based on gesture complexity and system constraints, such as battery life and latency. In terms of sensor integration, future efforts will investigate adaptive sensor management techniques, such as

prioritized sampling and data compression, to address performance bottlenecks in multi-sensor setups. For multi-user applications, scaling the system to support larger groups and improving synchronization across devices will be a priority. The integration of advanced sensors, such as the AHAT depth camera, for precise spatial mapping and object alignment will further enhance collaboration.

Bibliography

[1] M. B. Algorithms. How to freeze layers in tensorflow. https://medium.com/biased-algorithms/how-to-freeze-layers-in-tensorflow-c1b5c98f2158, 2024. Accessed: 24 November 2024.

[2] R. T. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality for multi-user collaborative environments. *IEEE Computer Graphics and Applications*, 40(5):72–84, 2021.

[3] W. Bai, C. Chen, and J. Wang. Accelerating neural network inference using tensorrt on edge devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4378–4385, 2020.

[4] M. Chowdhury, S. Rahman, and F. B. Ashraf. Leapgestrecog: A dataset for gesture recognition using leap motion controller. In *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 899–908, 2020.

[5] J. Dibene and S. Dunn. Hl2ss: Hololens 2 streaming server for real-time multi-sensor data acquisition. *ArXiv preprint*, arXiv:2211.02648, 2022. Retrieved from https://arxiv.org/abs/2211.02648.

[6] W. Hu, R. Liu, and J. Fan. Real-time performance optimization for collaborative mr environments. *Augmented Human Research*, 7(1):14–28, 2022.

[7] Jdibenes. Jdibenes/hl2ss: Hololens 2 sensor streaming. real-time streaming of hololens 2 sensor data over wifi. research mode and external usb-c a/v supported. https://github.com/jdibenes/hl2ss, n.d. Accessed: November 22, 2024.

[8] Jessemcculloch. Introduction to the multi-user capabilities tutorials - mixed reality. https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/tutorials/mr-learning-sharing-01, n.d. Accessed: November 22, 2024.

[9] Kageyama. [keras] hand gesture recognition cnn. https://www.kaggle.com/code/kageyama/keras-hand-gesture-recognition-cnn/notebook, 2019. Accessed: November 22, 2024.

[10] B. Kurdi, S. Lozano, and M. R. Banaji. Introducing the open affective standardized image set (oasis). *Behavior Research Methods*, 49(2):457–470, 2017.

[11] S. Liu, H. Wang, and Z. Zhao. Multi-modal fusion for interactive mixed reality applications. *Journal of Mixed and Augmented Reality*, 3(1):22–35, 2021.

[12] Microsoft. Hololens research mode - mixed reality. `https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/research-mode`, 2023. Accessed: November 22, 2024.

[13] Microsoft Azure. Azure spatial anchors. `https://azure.microsoft.com/en-us/products/spatial-anchors`, n.d. Accessed: November 22, 2024.

[14] Microsoft HoloLens. Apps, services, and solutions for hololens 2. `https://www.microsoft.com/en-us/hololens/apps`, n.d. Accessed: [insert access date here].

[15] Microsoft HoloLens. Hololens 2 - overview, features, and specs. `https://www.microsoft.com/en-us/hololens/hardware#document-experiences`, n.d. Accessed: November 22, 2024.

[16] Microsoft Learn. Hololens 2 fundamentals: Develop mixed reality applications. `https://learn.microsoft.com/en-us/training/paths/beginner-hololens-2-tutorials/`, 2024.

[17] ONNX. Open neural network exchange. `https://onnx.ai/`, 2023.

[18] Photon Engine. Simple and trusted multiplayer for unity: Photon unity networking for unity multiplayer games — pun2. `https://www.photonengine.com/pun`, n.d. Accessed: November 22, 2024.

[19] J. A. Russell. A circumplex model of affect. *Journal of Personality and Social Psychology*, 39(6):1161–1178, 1980.

[20] R. Sodhi, H. Benko, A. D. Wilson, and K. Hinckley. Lightguide: Enhancing mr applications with projected visualizations. *ACM Transactions on Graphics (TOG)*, 38(6):1–10, 2019.

[21] S. Teerapittayanon, B. McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *Proceedings of the 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016.

[22] V. A. Ungureanu, E. E. Milios, and A. Shih. Hololens 2 research mode as a tool for real-world data collection and analysis. *ArXiv preprint*, arXiv:2008.11239, 2020. Retrieved from `https://arxiv.org/abs/2008.11239`.

[23] V7 Labs. Cross entropy loss: Intro, applications, code. `https://www.v7labs.com/blog/cross-entropy-loss-guide`, n.d. Accessed: 24 November 2024.

[24] Vtieto. Hololens research mode - mixed reality. https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/research-mode, n.d. Mixed Reality — Microsoft Learn.

[25] Y. Yang et al. Towards immersive collaborative sensemaking. *Proceedings of the ACM on Human-Computer Interaction*, 6(ISS):722–746, 2022.

Appendix A

Gesture Recognition Model Architecture

The gesture recognition model developed for this study incorporates a convolutional neural network (CNN) with sequential learning and early exit mechanisms. The following sections detail the implementation of the model's initialization and forward pass logic. This architecture is designed for efficient gesture recognition, balancing computational complexity with high accuracy.

A.1 Model Initialization

Listing A.1 provides the initialization of the gesture recognition model. This includes:

- Four convolutional layers for feature extraction, each followed by ReLU activations and max-pooling layers to reduce spatial dimensions.

- Three fully connected layers for early exits, enabling intermediate predictions to optimize computational efficiency.

Listing A.1: Model Initialization

```
class GestureRecognitionModel(nn.Module):
    def __init__(self):
        super(GestureRecognitionModel, self).__init__()

        # Convolutional layers
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )

        self.exit0_fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32 * (IMG_SIZE // 2) * (IMG_SIZE // 2), 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )

        self.exit1_fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * (IMG_SIZE // 4) * (IMG_SIZE // 4), 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 96, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )

        self.exit2_fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(96 * (IMG_SIZE // 8) * (IMG_SIZE // 8), 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
```

```
            nn.Linear(128, 10)
        )

        self.conv4 = nn.Sequential(
            nn.Conv2d(96, 96, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )

        # Final classifier
        self.fc1 = nn.Linear(96 * (IMG_SIZE // 16) * (IMG_SIZE // 16), 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
```

- A final classifier for the most accurate predictions, processing the data through the entire network.

A.2 Model Forward Pass

The forward pass logic of the model is shown in Listing A.2. This section describes how input data flows through:

Listing A.2: Model Forward Pass

```
    def forward(self, x):
        outputs = []  # Outputs from all exits

        # Pass through conv1
        x = self.conv1(x)
        outputs.append(self.exit0_fc(x))  # Early Exit 0

        # Pass through conv2
        x = self.conv2(x)
        outputs.append(self.exit1_fc(x))  # Early Exit 1

        # Pass through conv3
        x = self.conv3(x)
        outputs.append(self.exit2_fc(x))  # Early Exit 2

        # Pass through conv4 and final classifier
        x = self.conv4(x)
        x = x.view(x.size(0), -1)  # Flatten
        x = torch.relu(self.fc1(x))
```

```
        x = torch.relu(self.fc2(x))
        outputs.append(self.fc3(x))  # Final Output

        return outputs
```

- Convolutional layers, extracting hierarchical features from the input image.
- Early exits, which produce intermediate outputs for simpler gestures.
- The final classifier, which consolidates high-level features to provide the most accurate predictions for complex gestures.