

INTEGRATION OF REAL TIME DATA TO VISUALIZE PHYSICAL ENVIRONMENTS IN
XR

A Thesis

Presented

to the Faculty of

California State University, Dominguez Hills

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Muhammad Zaid Kamil

Fall 2025

THESIS : INTEGRATION OF REAL TIME DATA TO VISUALIZE PHYSICAL
ENVIRONMENTS IN XR

AUTHOR: MUHAMMAD ZAID KAMIL

APPROVED:

Ali Jalooli, Ph.D.
Thesis Committee Chair

Horace Crogman, Ph.D.
Committee Member

Alireza Izadoost, Ph.D.
Committee Member

Copyright by
MUHAMMAD ZAID KAMIL

2025

All Rights Reserved

ACKNOWLEDGEMENTS

I would like to express appreciation to the committee members, for their invaluable patience, expertise and feedback throughout the thesis process who supported the work throughout the years.

This Thesis was a long process which involved 2 years, supported by different committee chairs and members whose contributions were essential at various stages. I would like to thank the committee chair, Dr. Jalooli for his expertise in IoT technologies, and his constant feedback to understand the research goals. I would like to thank Committee Member Dr. Crogman for his expertise to develop the XR environments, as well as the financial support for acquiring the Meta Quest HMD essential for this research, for being with my committee member for the entire process. Special thanks to Dr. Alireza, whose provided the clarity and quality of the thesis paper. I also want to express my appreciation to the CSUDH Innovation Incubator team, provided their expertise on the marketing and business side to pitch this thesis to potential investors in CSU Sunstone competition I would like to thank my family, my brother in law for providing access to Cisco IoT technologies to use for the Thesis, my parents and siblings for their support and motivation for the thesis.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES.....	viii
LIST OF FIGURES	ix
ABSTRACT.....	ix
1. INTRODUCTION	1
Objective(s)	2
Limitation of Our Work	3
Current Status	5
2. RELATED WORK.....	7
Current Status	7
3. SYSTEM ARCHITECTURE	22
4. THREE-DIMENSIONAL MODELLING.....	23
3D Modeling Workflow and Tool Evaluation	23
5. IOT SENSOR DEPLOYMENT	32
Sensor Selection Criteria	32
IoT Dashboard.....	34
6. AZURE IOT PROCESSING	36
Azure Resource Utilization and SQL Database.....	37
7. REAL TIME IOT DATA INTEGRATION	39
IoT Integration in Unity.....	39
API Integration	43
8. REAL TIME TRAFFIC AND WEATHER DATA VISUALIZATION.....	55
Traffic Visualization Case Studies	55
Real Time Weather Data	59
9. META QUEST VIEW	61

Teleportation	61
Networking.....	66
10. ARTIFICIAL INTELLIGENCE WITH XR.....	71
AI-driven Virtual Avatars	71
11. CONCLUSION.....	75
REFERENCES	76
APPENDIX: SCRIPTS	80

LIST OF TABLES

1. Comparison Overview of Related Lecture Review	21
2. Comparison of realistic 3D with Unity 3D	29
3. Sensor device comparison.....	32
4. API Comparison Table.....	50

LIST OF FIGURES

1. The integration of various adaptive technologies for classroom enrichment	8
2. Visual summary of XR and real-world physics used in education	9
3. IoT Wireless Technologies for Water Quality Monitoring.....	10
4. Comparison between traditional MOT tracking and the 3D reconstruction.....	14
5. Function menu of multi-scale maps	15
6. Comparison between traditional MOT tracking and the 3D reconstruction.....	17
7. Architecture for Real-Time 3D Modeling and XR Environment Creation	22
8. Coordinates of Open Street Map.....	24
9. Utilizing BloSM Coordinates.....	25
10. 3D Real Time Model of CSUDH Campus utilizing Google 3D Tiles	25
11. Real Time 3D Models.....	26
12. 3D Map using Photorealistic Tiles.....	27
13. ArcGIS SDK configuration.....	27
14. Unity's real-time rendering of downtown SF buildings using ArcGIS SDK.....	28
15. API Key generation and Inspector view of ArcGIS	28
16. Teleportation Environment and Landing Plane.....	31
17. Campus and Desalination in XR Environment.....	31
18. AV201 sensor device and Cisco IoT Network Gateway	33
19. Cisco IoT LoRa WAN Architecture.....	34
20. Cisco IoT Dashboard Landing Page	35
21. Cisco IoT sensor inventory and devices	35
22. Azure IoT Integration for SQL based data retrieval pipeline	36
23. Azure Resources	37
24. Azure SQL data processing workflow.....	38

25. Structured format of temperature and humidity with readable timestamp	40
26. CSUDH virtual map for sensor data integration in Unity.....	41
27. Desalination map for sensor data integration in Unity	42
28. Multiplayer environment for sensor data integration in Unity	42
29. Utilizing Azure functions for REST API integration	44
30. Pseudocode Azure Functions for REST API integration.....	44
31. Successful publication with Azure environment using Linux system.....	45
32. Azure Function Flowchart	46
33. Cost for Azure.....	46
34. Testing the API using Postman.....	47
35. Error fetching data in VR Campus Map built in Meta Quest HMD.....	48
36. Error fetching data in VR Desalination Facility built in Meta Quest HMD.....	48
37. Pseudocode of Fast API app	49
38. Workflow diagram of Fast API.....	50
39. I-10 Freeway VR visualization (Moderate Traffic).....	55
40. I-110 Freeway VR visualization (Heavy Traffic).....	56
41. I-5 Freeway VR visualization (Heavy Traffic).....	56
42. I-210 Freeway VR visualization (Smooth Traffic)	57
43. Google Directions API configuration.....	57
44. Google Directions API usage metrics.	58
45. Condensed pseudocode version of Google Traffic Controller script.....	58
46. Condensed pseudocode version of weather script	59
47. The weather visualization in VR display in clear UI panels.....	60
48. Teleportation to different environments	61
49. Pseudocode of teleportation functionality in Unity	62

50. Landing page environment in Unity and VR Environment.	62
51. Teleportation to Dodgers Stadium.	63
52. Teleportation to Los Angeles downtown environment.....	64
53. Teleportation to Desalination environment.....	65
54. Teleportation to soccer environment.....	65
55. Network architecture.....	66
56. Network manager.....	67
57. VR Multiplayer shared experience	68
58. Avatar with the UI overlay in VR.....	69
59. Menu UI and Avatar selection	69
60. 3D XR landing elements developed for XR visualization.....	70
61. 3D XR environments developed for XR visualization	70
62. Avatar NPC to Player Interaction	71
63. Creation of AI Avatar in Convai.....	72
64. Uploading documents in Convai.....	73
65. Conversations with NPC in XR Environment	74

ABSTRACT

Real-time data integration into AR/VR platforms is an essential approach in addressing global social disconnection by enhancing immersive interaction with physical environments. While current social media platforms effectively bridge user communication through text, posts, and video interactions, they lack the capability to connect users with real-time visualizations of physical environments. In the current market, different types of live data require separate applications; for example, weather data requires a weather app, and traffic data requires a map application. There is a lack of a unified application that integrates all real-time data sources, including traffic, weather, and IoT. This thesis presents a sophisticated XR software ecosystem, {Z}, developed for the Meta Quest headset. It is designed to improve the user's connection to their environment by providing real-time data visualization and interaction across major areas in Los Angeles, including freeways, urban zones, sports venues, academic campuses, and industrial sites. The system utilizes interactive 3D features to visualize live data streams such as weather, freeway traffic, and IoT telemetry within a scalable, multiplayer XR application. This thesis explores a new market which is the integration of Real Time Analytics and XR Market.

CHAPTER 1

INTRODUCTION

Digital communication and online social platforms such as Instagram and TikTok have reshaped human interactions, through virtual connections (texting, video feed, photo sharing). Despite this technological advancement in interpersonal connectivity, a notable gap persists in connecting users to real-world environments, particularly when users want to experience a venue in a different geographical location in real time. As of 2025, the global XR market is valued at approximately \$75.18 billion and is projected to reach \$296.6 billion by 2030, growing at a compound annual growth rate (CAGR) of 31.61% (Mordor Intelligence, 2024). There are key growth factors such as advancements in hardware in XR devices, including lighter, more comfortable, and affordable XR headsets such as Meta Quest, Snap Spectacles, Apple Vision pro that significantly increasing accessibility and adoption rates among both individual consumers and businesses (Reuters, 2024). Additionally, the global deployment of 5G networks has improved XR experiences by enabling faster data transmission speeds and lower latency, crucial for seamless real-time interactions (Mordor Intelligence, 2024; Scoop Market, 2024). Enterprise adoption further accelerates market expansion, as industries such as Niantic Spatial, Snap Lenses increasingly utilize XR technologies for practical applications such as employee training, remote collaboration, and enhanced customer engagement (Data Insights Market, 2024; Core Market Research, 2024; Mordor Intelligence, 2024). The consumer application market is generally concentrated in gaming, education, healthcare, and retail. However, there lacks application that integrates real time data to users. The current real-time data are presented in 2D Dashboards or device panels which are static and non-interactive. In addition, for people to view real-time events such as games in stadiums or visit attractive city venues, users have to purchase a ticket

and physically be in the location. By utilizing the headset, the z-map Software will allow users to travel to different venues and cities in the metaverse to view real-time events and data in an immersive environment.

Objective(s)

Despite advancements in XR and spatial applications, there remains a gap in connecting users to real-time environments. In the current market, different live data requires different applications. For example, the weather data requires a weather application, or the traffic data requires map application. There lacks an application that in-house all the real time data integration includes traffic, weather. Providing the application in Immersive provides the users to utilize the 3D interactive features compared to the current Media that restricts the users to 2D non-interactive static view. This thesis is building a sophisticated software XR ecosystem {Z} designed to connect users to real-time environments to deliver the most lifelike and immersive based environment XR experience. Scalable, multiplayer software in XR that provides access to external real-time data streams from 360 live streaming, weather, traffic of major cities and internal data such as people occupancy. We are currently exploring a new market which is the integration of Real Time Analytics and XR Market. With the global XR market is valued at approximately \$75.18 billion and is projected to reach \$296.6 billion by 2030, and the real time analytics market is valued at \$56.6B projected to reach \$137.38 billion by 2030 (Data Insights Market, 2024). By combining these two markets, there is potential business opportunities for this application.

To improve users' connection to immersive real-world environments, this thesis introduces a build of an immersive application called {Z}, designed specifically to facilitate users for immersive and interactive engagement with real-time environments to view real-time

data and events. For internal data in buildings, the AR/VR is integrated with IoT sensors to capture and collect telemetry, and people occupancy data inside buildings utilizing Cisco IoT, Azure SQL. For the external data streams from Google Maps Platform APIs, data includes real-time traffic, weather, air-quality, 3D street view for real-time visualization.

Our goal is to solve the problem of connecting users to different physical environments by providing access to real time data such as traffic, people occupancy, weather, and a more immersive real-time viewing experience offering 360-degree VR live stream footage for stadium venues, campuses, cities and industrial units in global region. The application allows the user to teleport to different venues and cities simultaneously to view different live events.

Limitation of Our Work

Integration Challenges with IoT Sensors and AR/VR Devices

A critical limitation encountered in this research was the integration of IoT sensors with Augmented Reality/Virtual Reality (AR/VR) Head-Mounted Devices (HMDs), specifically the Meta Quest headset. Due to stringent firewall settings within the Meta Quest environment, establishing direct communication with external IoT sensor data proved problematic. Initially, Azure Functions were utilized to retrieve API-based sensor data from Cisco IoT devices, processed and routed through Unity via the Azure cloud ecosystem. Despite functional initial tests, ongoing firewall restrictions continued to impose connectivity challenges, significantly complicating real-time data integration.

Specialized Licensing and High Cost of Real-Time 3D Model Rendering

The second major limitation pertains to the generation of real-time rendered 3D models from geospatial data sources such as Google 3D Tiles. Effective material and texture baking for

Unity integration require specialized software licenses, notably Simplygon, whose licensing costs are prohibitively expensive for independent developers. This economic constraint significantly limited the scalability and accessibility of the proposed modeling solution, thereby restricting widespread adoption and practical deployment of the developed XR models.

Refresh Data Coroutine Method for Internal IoT Sensors

Another operational limitation involved the data refresh mechanism implemented through the RefreshDataCoroutinemethod for internal IoT sensor data retrieval. While designed to provide periodic updates, this method relied heavily on explicit logging checkpoints initial activation, after completion of waiting intervals (WaitForSeconds(refreshInterval)), and prior to re-query execution (ConnectAndQueryDatabase()). This process highlighted inefficiencies and potential reliability concerns, emphasizing the need for more robust and efficient real-time data retrieval methods within the existing architecture.

Unanticipated and High Costs of Azure API Management

A significant financial limitation arose due to unexpected costs associated with the Azure API Management service. The billing structure of Azure API Management is based on a flat monthly rate calculated hourly, meaning costs accumulate irrespective of actual usage. Between November 25th and December 11th, API Management resource costs unexpectedly escalated to \$2,220.42, even though the API was only provisioned to connect telemetry data from Azure SQL Database to the Meta Quest HMD and was not actively retrieving data. Further exacerbating this issue, the associated billing profile was linked to an Azure Student resource, preventing the use of available Sponsorship credits to offset these costs. This unforeseen financial burden highlights critical administrative and budgetary limitations in using cloud-based API management services for research applications.

Current Status

Developed and deployed a digital twin Virtual Reality (VR) system compatible with the Meta Quest platform, utilizing Unity, C#, Azure cloud services, and Google Cloud API. This system significantly enhances the ability of global users to view, track, and interact with real-time events and venues.

Integrated Cisco Internet of Things (IoT) sensors to collect and transmit telemetry data. Leveraged Azure SQL Database, Azure Functions, and API retrieval mechanisms for real-time data visualization directly within Augmented Reality/Virtual Reality (AR/VR) Head-Mounted Displays (HMDs).

Implemented interactive 3D dashboards featuring real-time traffic and weather data updates. These dashboards are configured to automatically refresh every ten minutes, providing continuous, up-to-date environmental information for major urban areas.

Enabled seamless teleportation functionalities within a single immersive XR application, allowing users to instantaneously navigate between various spatially distinct environments and real-time venues.

Established and validated a robust multiplayer XR architecture, capable of simultaneously supporting over 100 concurrent users within the same virtual reality environment.

In Progress

Integrating live 360-degree video streaming capabilities into the Meta Quest VR application by installing VR-compatible 360-degree cameras at stadium venues. Plans include conducting detailed 3D scans to accurately incorporate realistic and fully rendered 3D stadium models into the immersive experience.

Generating detailed and accurate real-time 3D models of selected buildings and venues to enhance the realism and immersive qualities of the XR application.

Integrating real-time IoT data on people occupancy within specific venues, leveraging existing Wi-Fi network infrastructure to provide accurate and dynamic spatial occupancy visualization.

Collaborating actively with various stadium management personnel to implement and operationalize these technological enhancements across multiple venues.

CHAPTER 2

RELATED WORK

Current Status

This thesis introduces the integration of real-time data, IoT sensor telemetry, and immersive extended reality (XR) functionalities, including teleportation, interactive avatars, and scalable multiplayer environments. However, several studies address related aspects of our research objectives.

Virtual Reality, Augmented Reality, And Mixed Reality in Experiential Learning: Transforming Educational Paradigms (Crogman Et All, 2024)

The authors, Horace T. Crogman, Victor D. Cano, Edlyn Pacheco, Rohan B. Sonawane, and Reza Boroon present an interdisciplinary discussion on how extended reality (XR) tools can address limitations in traditional education by offering immersive, learner-centered environments. Instead of offering a systematic review of past studies, the paper focuses on applying XR in physics education, biological sciences, and environmental modeling using platforms like Unity and custom-developed XR apps. They particularly emphasize real-time interaction, visualization of abstract phenomena, and cross-device deployment (e.g., phones, tablets, VR headsets).

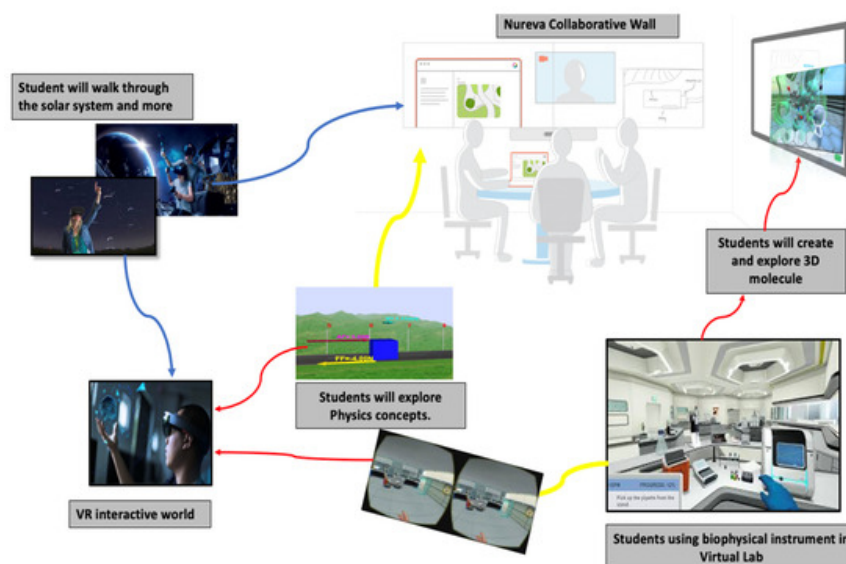
Crogman et al argue that XR creates opportunities for “learning by doing” in environments that are otherwise impossible, dangerous, or costly to replicate physically. This is particularly useful for visualizing atomic and molecular models, electric field dynamics, and climate and earth system models.

Various adaptive and immersive technologies can be integrated into classroom environments to enhance experiential learning (see Figure 2) which shows screenshots of XR

apps simulating electric field lines, displaying real-time manipulation of charge values. This highlights the value of immediate visual feedback, allowing students to explore physical phenomena intuitively.

Figure 1

The integration of various adaptive technologies for classroom enrichment



Note: A workflow diagram connecting interactive VR content, scientific simulations, and collaborative tools. It shows how different educational modules are interconnected through XR. Demonstrates mixed-reality molecular interaction on a tablet, with students rotating molecules and adjusting bond angles via touch interface. Adapted From Virtual Reality, Augmented Reality, And Mixed Reality in Experiential Learning: Transforming Educational Paradigms, by Crogman Et All, 2024.

This underlines tangible manipulation of data, even without VR headsets. Notably, the paper introduces case studies and system design elements developed at their institution, such as:

3D Molecular Visualization

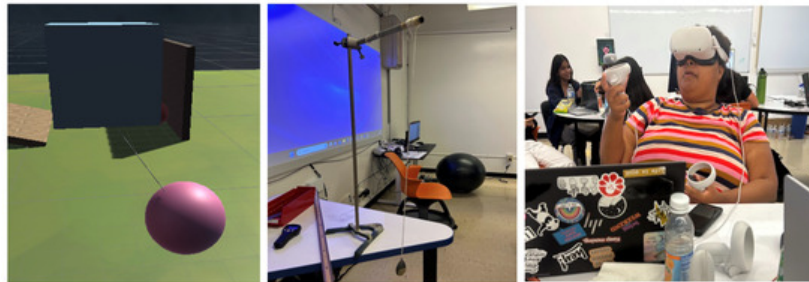
Physics Simulations with Real-Time Variables

Geospatial and Earth Science VR Environments

feasibility and These implementations are more project-driven than review-based and are intended to demonstrate highlight pedagogical value rather than synthesize existing XR education literature comprehensively.

Figure 2

The integration of various adaptive technologies for classroom enrichment.



Note: It is a visual summary of how immersive XR (Extended Reality), and real-world physics apparatuses are being used together in education. Adapted From Virtual Reality, Augmented Reality, And Mixed Reality in Experiential Learning: Transforming Educational Paradigms, by Crogman Et All, 2024.

This paper reinforces the academic and pedagogical justification for XR in STEM education. It validates that students benefit from interactive 3D models and visual-spatial feedback, especially in domains like physics and molecular biology. Their system is primarily simulation-based.

Advances In Machine Learning and IoT For Water Quality Monitoring (Ismail Et All 2024)

This paper explores the integration of IoT and ML techniques to analyze to make accurate predictions for water quality monitoring. In the paper, ML algorithms help predict trends and detect anomalies in water quality, such as changes in turbidity, pH, and dissolved oxygen. It explores different wireless IoT technologies like Wi-Fi, Zigbee, RFID, LoRa WAN, and Cellular Networks integrated with supervised and unsupervised ML algorithms for real-time

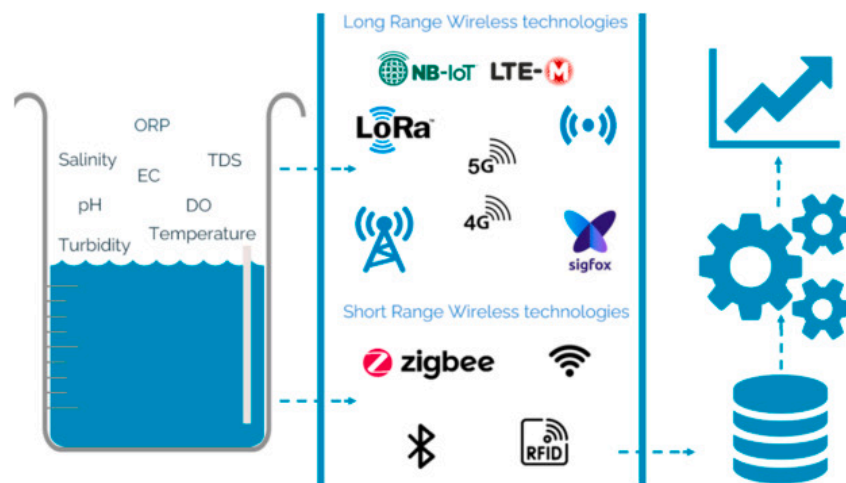
data collection, analysis, and decision-making to improve water quality monitoring. The key insights from the paper are the following:

It explores the use of IoT technologies for remote sensing and real-time monitoring of water quality. The focus is on how devices communicate through various wireless technologies to collect data continuously without human intervention, enabling early anomaly detection. The integration of ML algorithms helps in analyzing vast amounts of real-time sensor data. The paper specifically focuses on supervised learning for predicting water quality parameters and unsupervised learning for anomaly detection, pattern recognition, and clustering.

The paper highlights various practical applications across sectors like wastewater treatment, agriculture, and water distribution, showing how IoT + ML technology can enhance the monitoring process and decision-making. The below Figure 1 presents a variety of IoT wireless technologies classified into two categories: long-range and short-range technologies. It shows how different technologies are used for water quality monitoring (WQM) and the types of data they handle.

Figure 3

IoT Wireless Technologies for Water Quality Monitoring



Note: Advances In Machine Learning and IoT for Water Quality Monitoring, by Ismail Et All, 2024

NB-IoT, LTE-M, LoRa, 5G, 4G: These technologies support large-scale deployments, offering low power consumption and extended data range. For example, LoRa is well-suited for rural areas where long-range communication is essential, and 5G is optimal for areas requiring high-speed data transmission with low latency.

Zigbee, Wi-Fi, Bluetooth, RFID: These are generally used for localized data transmission, making them suitable for monitoring systems where sensors are closely located to one another. For example, Wi-Fi is common in urban environments, and Bluetooth can be used in smart water applications where short-range communication is sufficient.

This thesis has the same the approach of utilizing wireless technologies like LoRa for sensor data collection in real-time. This diagram is helpful as it showcases the spectrum of wireless technologies and helps align the thesis's real-time XR solution, which can utilize similar technologies to provide interactive, immersive experiences.

IoT Integration: Both your research and the paper focus on real-time sensor data integration; however, your thesis elevates this concept by visualizing these data streams inside immersive XR environments (Meta Quest VR), while the paper primarily uses IoT for monitoring and alerting. Your system integrates Unity-based VR environments, turning real-time IoT data into interactive, immersive experiences rather than just analytical dashboards. The key insights from the paper are the following:

Artificial Intelligence as a Service in Distributed Multi-Access Edge Computing on 5G, Extracting Data Using IoT (Ismail et al 2024)

This paper delves into the role of Artificial Intelligence (AI), Internet of Things (IoT), and Augmented Reality/Virtual Reality (AR/VR) in shaping the future of 5G networks using Multi-Access Edge Computing (MEC). The main goal of the paper is to explore how MEC can

provide low-latency and high-bandwidth services in the 5G environment, particularly for IoT applications, AR/VR experiences, and real-time reporting. the paper lays the groundwork for integrating IoT, 5G, and AR/VR through MEC and AI.

The paper highlights that IoT devices are growing rapidly, and to handle the huge amount of real-time data they generate, we need networks like 5G. 5G's low latency and high-speed capabilities make it an ideal solution for transmitting this real-time data, particularly for applications in smart cities or remote monitoring. The paper argues that MEC will be the backbone of such systems, providing fast processing of IoT data at the network edge, which is crucial for applications like self-driving cars or healthcare monitoring systems.

MEC brings computing capabilities closer to users, reducing the need for long data trips to the cloud and enabling faster decision-making. This is especially important for AR/VR applications, which require instantaneous data processing to deliver smooth and immersive experiences. The paper emphasizes that 5G and MEC working together can support real-time AR/VR environments, where high bandwidth and low latency are needed to deliver interactive content without lag.

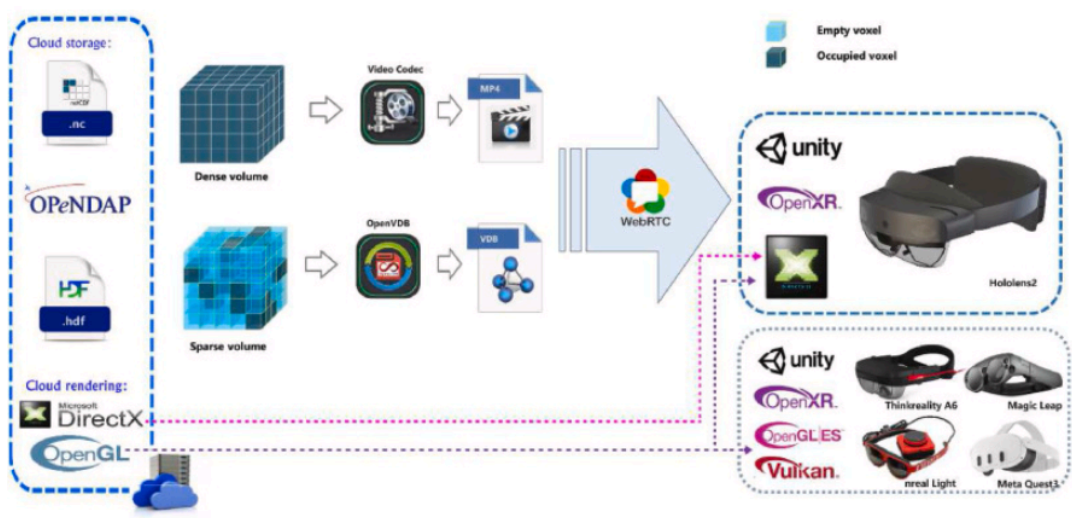
The paper introduces AI as a Service (AIaaS) at the edge, which means that AI tools and infrastructure can be used on demand to process data closer to where it's generated. This enables faster decision-making and can be particularly useful in industries like healthcare or smart cities where real-time data analysis is essential.

XR-based Interactive Visualization Platform for Real-Time Exploring Dynamic Earth Science Data

The paper titled "XR-based interactive visualization platform for real-time exploring dynamic Earth science data" by Xuele Zhang et al. presents an advanced platform, MetIVA, that

leverages Extended Reality (XR) and cloud rendering technology for real-time data exploration. The paper addresses the need for effective tools to visualize complex multi-dimensional datasets, such as meteorological data and environmental variables, in an interactive, immersive way. The transition from 2D planar displays to immersive holographic 3D environments has brought advancements in visualization technology.

The paper explores the integration of XR technologies, specifically Mixed Reality (MR), with cloud-based rendering for enhanced data visualization in Earth science applications. This allows users to interact with large datasets in 3D using immersive technologies such as HoloLens 2. The platform, MetIVA, is designed to handle multi-dimensional data (such as weather simulations, air quality data, and geospatial datasets) at various scales, from global to street-level visualization. The data is processed on a cloud server and then rendered on MR devices like HoloLens 2 using Unity3D and WebRTC for low-latency interaction.

Figure 4*Mixed Reality Device and Platform Composition*

Note: illustrates the platform structure of MetIVA, showing the connection between cloud storage, data processing, and visualization tools like Unity. It emphasizes the use of HoloLens 2, ThinkReality A6, Magic Leap, and Meta Quest for rendering environmental data in XR. Adapted from XR-based Interactive Visualization Platform for Real-Time Exploring Dynamic Earth Science Data,

The interaction module allows users to control 3D data through hand tracking and gaze in MR environments, like the XR interaction in your thesis. Since the output data from meteorological or air quality models can be several gigabytes in size, all the source data are acquired, stored, processed and rendered on the cloud computing platform of AWS EC2.

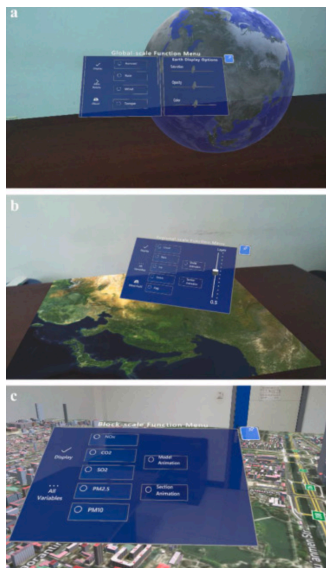
MetIVA incorporates intuitive user interactions via hand tracking and gaze control to manipulate visualized data. These features enable users to explore large-scale environmental data in an immersive, real-time environment while gaining an enhanced understanding of the data patterns and trends.

To make handling large datasets feasible, the platform utilizes cloud-based GPU rendering and data compression techniques (such as video codecs and OpenVDB) to efficiently process and visualize data, reducing the computational load on the local device.

The paper presents a robust multi-scale mapping system within MetIVA, which allows users to switch between global, regional, and block-level views of the data. This feature is critical for environmental monitoring, as it provides flexible zooming and detailed exploration of specific regions or phenomena.

Figure 5

Function menu of muti-scale maps



Note: Shows how MetIVA handles different data visualizations at varying scales (global, regional, street-level) and how users can interact with the function menus to control the 3D environment.

Data-driven Online Traffic Reconstructions in Virtual Reality (Wang et al, 2022)

The paper "Data-driven Online Traffic Reconstructions: Interactively Optimizing in Virtual Reality" presents an innovative method to generate 3D traffic reconstructions in real-time from video data, utilizing Multiple Object Tracking (MOT) and traffic simulations. The method enhances existing online MOT systems by tackling key issues such as identity switches and incomplete vehicle trajectories, which have been limitations in previous 3D traffic reconstruction

approaches. This work is especially notable in its use of virtual reality (VR) for interactive traffic analysis.

This approach addresses a major issue in traditional MOT methods, where vehicles can be lost during tracking (due to occlusion or identity switches) something especially relevant in dense traffic scenarios. Their method offers a solution by reconstructing these trajectories and enabling real-time user interaction in VR. The paper outlines a two-step method for 3D traffic reconstruction. The first step involves extracting vehicle trajectories from video data using online MOT techniques. This is particularly important because MOT traditionally works in 2D space, but this method transforms those coordinates into geodetic coordinates for 3D reconstruction. The second step optimizes these trajectories using traffic simulations and a Kalman filter, which helps correct incomplete trajectories and ensure a smooth flow.

This thesis shares several conceptual similarities with the methods outlined in this paper, particularly in the use of XR for visualizing real-time data.

Real-time Data Integration: Both the paper and this thesis focus on real-time data collection, but while the paper is primarily concerned with video tracking and vehicle trajectories, your thesis involves integrating real-time IoT data (like weather, temperature, and occupancy sensors) into immersive XR environments. Your work focuses more on sensor-driven interaction in XR, whereas the paper uses video feeds for vehicle tracking.

While the paper uses VR for interactive traffic data exploration, this thesis goes a step further by creating an interactive 3D space where users not only view but also manipulate live environmental data. Your work could benefit from the optimization techniques presented in the paper, particularly the Kalman filter and traffic simulations, which could be applied to sensor data in XR.

Traffic Simulation and Predictive Models: One of the key strengths of the paper is its ability to predict missing data in traffic scenarios. Similarly, this thesis could incorporate predictive models for real-time environmental data, such as predicting weather conditions or traffic flow patterns based on historical data and sensor readings. Both works share the concept of using simulation-based predictions to enhance real-time decision-making.

Cloud and Edge Computing: While the paper leverages cloud and edge computing for real-time rendering of traffic data, this thesis also works with cloud technologies (e.g., Azure IoT for sensor data collection). Both systems utilize cloud computing to process large datasets, but your focus is on XR visualization, while the paper uses it for traffic modeling.

Figure 6

This set of images shows the comparison between traditional MOT tracking and the 3D reconstruction method proposed in the paper

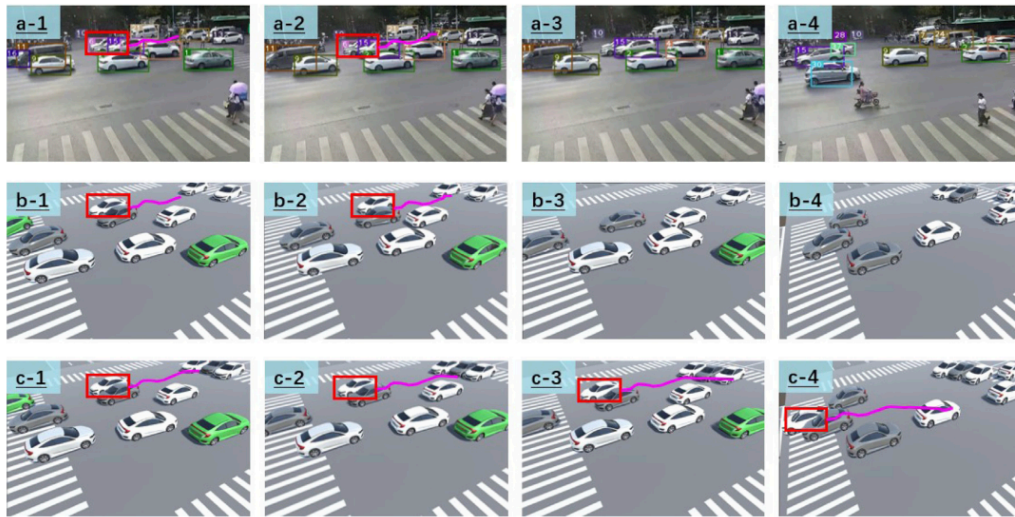


Fig. 3. Some snapshots of the tracking results in the video (top), the 3D reconstruction (middle) using the results of MOT method, and the 3D reconstruction using the results of our method in the virtual reality (bottom) when the vehicle that is tracked in its first half life span and the second half of its life span is completely lost.

Note: The top row (a-1 to a-4) shows the original video footage with tracking results using MOT, including the tracking IDs of each vehicle. The middle row (b-1 to b-4) shows how this data is transformed into a 3D reconstruction. The bottom row (c-1 to c-4) demonstrates the vehicle's trajectory is visualized in a VR environment

This shows the interactive nature of the reconstruction, where users can see how the data is corrected in real-time. The paper presents a significant step in improving the accuracy and realism of 3D traffic reconstruction from video data. It integrates video tracking, traffic simulation, and VR optimization to deliver a robust method for visualizing and interacting with real-time traffic data. Your thesis shares many conceptual similarities but focuses on real-time environmental data in immersive XR, providing a valuable comparison for the practical applications of real-time data visualization in XR environments.

AR/VR and IoT Integration

Many research efforts have explored the integration of AR/VR with IoT technologies, particularly for monitoring and management of industrial environments, infrastructure maintenance, and interactive data visualization.

Ghazali et al. (2020) introduced an educational mixed reality (MR) game to facilitate teaching about water desalination processes. Their system integrated MR visualization using the Unity engine but lacked the integration of real-time IoT sensor data, focusing instead on static educational scenarios.

Similarly, Mól et al. (2008) developed VR simulations for nuclear plant operations using real-time radiation sensor data. This approach utilized the Unreal Engine to visualize predefined sensor scenarios, demonstrating the capability to present IoT sensor data within virtual environments. However, it focused primarily on static, predetermined datasets rather than dynamic, live data integration.

In contrast, Pfetter et al. (2015) explored the use of immersive VR for real-time monitoring and maintenance in industrial operations. Their approach incorporated dynamic data streams from industrial plants, supporting maintenance and troubleshooting tasks. Despite the

interactive capabilities provided, their system primarily focused on historical playback of sensor data, lacking continuous real-time data integration and robust scalability.

Kuvshinnikov et al. (2024) developed a VR-based simulator specifically for non-destructive testing (NDT) in radiographic inspections. This approach utilized digital twins to simulate realistic inspection scenarios, highlighting the utility of immersive training systems. However, the system lacked direct, continuous integration of IoT sensor data and did not incorporate dynamic user interactions or multiplayer functionalities.

In their comprehensive overview, You et al. (2018) proposed a VR space leveraging IoT infrastructures for real-world visualization. They introduced a framework utilizing IoT-driven metadata for accurate VR reconstructions, emphasizing efficient real-time processing and immersive VR experiences. Their proposed end-to-end architecture highlights IoT infrastructure as essential for real-time VR space reconstruction, yet lacks specifics on integrating dynamic multiplayer scenarios and advanced AI-driven interactions.

Real-Time Visualization and Telemetry Integration

Several works addressed real-time data visualization within XR environments, crucial to our thesis's central goal.

Kolomvatsos et al. (2018) surveyed data stream analytics techniques for IoT applications, highlighting methods for handling continuous data streams. Li et al. (2020) further explored cloud computing architectures for IoT data, underscoring the importance of effective cloud-based infrastructure for real-time processing. Lockner and Atzori (2021) evaluated IoT data storage and analytics performance within the Azure cloud environment, offering insights into scalable cloud management practices. Nonetheless, these works did not explicitly address the visualization of IoT data within immersive VR contexts.

This thesis significantly advances existing research due to several unique contributions:

Immersive Teleportation: Unlike previous literature, which primarily focused on fixed or limited environments, our platform uniquely enables real-time teleportation between diverse virtual scenarios, significantly enhancing user engagement and flexibility.

Advanced AI Integration: The inclusion of interactive, AI-driven NPC avatars leveraging large language models (LLMs) significantly increases interactivity and educational effectiveness, improving knowledge retention substantially beyond static or simple interactive models prevalent in earlier studies.

Scalable Multiplayer Architecture: Our system supports extensive concurrent user interactions, accommodating over 100 simultaneous participants, which notably exceeds existing single-user or limited multi-user scenarios described in prior literature.

Real-Time Data Integration: By incorporating diverse real-time data streams—including traffic, weather, and occupancy—through advanced dashboards interfacing with Azure SQL and REST APIs, our research provides robust, comprehensive data integration superior to isolated or limited data integrations found in current works.

Dynamic 3D Environment Generation: The capability to dynamically generate and integrate real-time 3D models from live scans into XR pipelines represents an advancement over static, predefined environments typically employed in existing XR and IoT literature.

Table 1

Comparison Overview of Related Literature Review

Reference	IoT Integration	Real- Time Data	Immersive Visualization	Teleportation	Multiplayer	Advanced AI Integration
Ghazali et al. (2020)	Limited	No	Yes	No	No	No
Mól et al. (2008)	Yes	Partial	Yes	No	No	No
Pfetter et al. (2015)	Yes	Partial	Yes	No	Limited	No
Kuvshinnikov et al. (2024)	Limited	No	Yes	No	No	No
Hong et al. (2020)	Yes	Yes	No	No	No	No
Kolomvatsos et al. (2018)	Yes	Yes	No	No	No	No
Our thesis	Yes	Yes	Yes	Yes	Yes	Yes

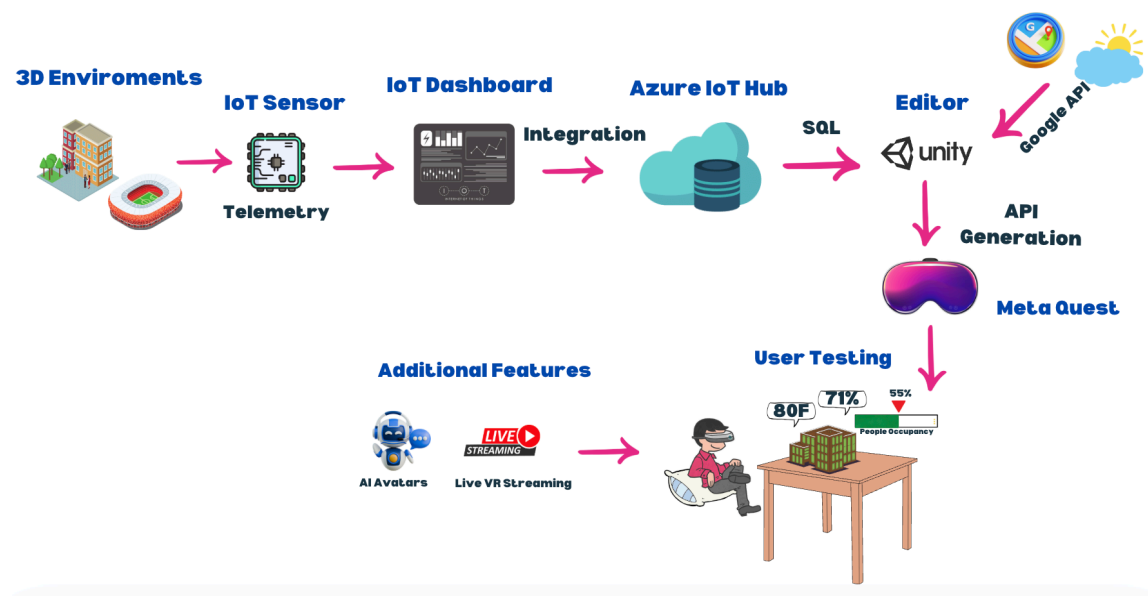
CHAPTER 3

SYSTEM ARCHITECTURE

The chapter outlines the methodology employed in integrating real-time data into an Extended Reality (XR) application. The integration involved several key phases: developing detailed 3D models, deploying IoT sensor infrastructure, managing data through cloud processing and APIs, and deploying the resulting interactive visualization on XR headsets. The methodology involved evaluating various approaches, selecting suitable tools based on performance, accuracy, and budget considerations, and optimizing models for effective visualization and interaction within Unity's XR environment. The overall process followed a structured workflow beginning with initial model generation, texture optimization, and integration into Unity for interactive visualization. Figure 7 provides an architectural overview illustrating the end-to-end integration process.

Figure 7

Architecture for Real-Time 3D Modeling and XR Environment Creation



CHAPTER 4

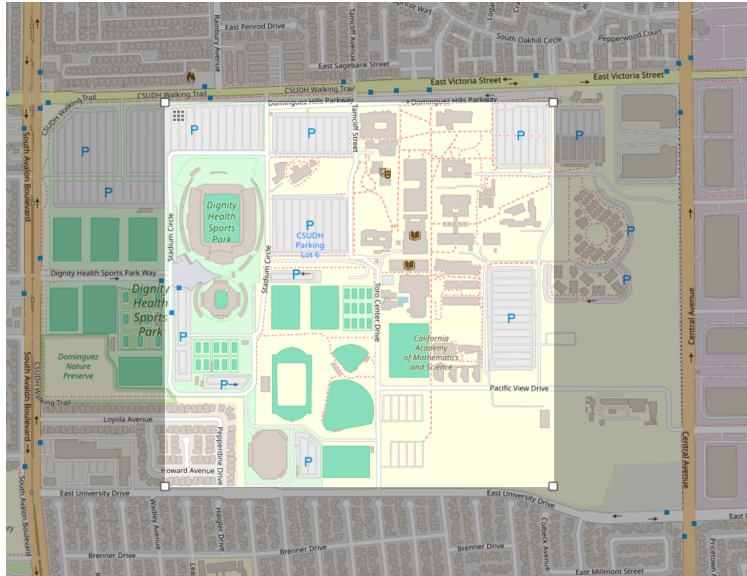
THREE-DIMENSIONAL MODELLING

3D Modeling Workflow and Tool Evaluation

Several methods were critically examined to generate realistic 3D representations, leveraging geospatial data and ensuring suitability for real-time rendering. Initially, Blender was adopted using the Blosm plugin, a tool designed to import Google's photorealistic 3D Tiles based on geographic coordinates from OpenStreetMap (OSM). The primary objective was to create geo-spatially accurate models of the California State University, Dominguez Hills (CSUDH) campus. The Blosm plugin effectively generated detailed urban-scale visualizations; however, significant limitations were encountered in the texture baking stage—an essential step for exporting visually realistic materials to Unity (Akenine-Möller et al., 2019). This stage required supplementary texture-baking software like Simplygon, which proved financially restrictive, leading to its exclusion from further consideration.

3D Realistic Models using Blosm Plugin in Blender

To generate a 3D model, we utilize the geographic coordinates using OpenStreetMap (OSM). For example, the below Figure 3.2 provides the screenshot of the geographic coordinates of the CSUDH campus. Specifically, the rectangular boundary around the campus area was selected, and the minimum and maximum latitude-longitude coordinates were extracted from OSM.

Figure 8*Coordinates of Open Street Map*

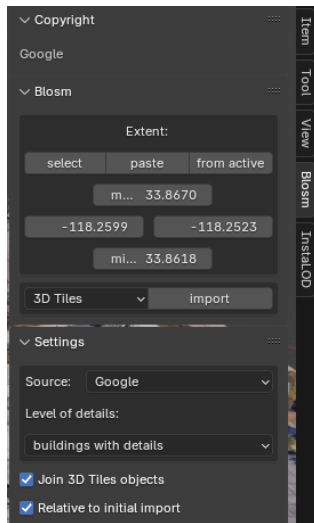
These geographic coordinates were then inserted into the Blosm plugin in Blender, an addon that directly interfaces with Google's photorealistic 3D Tiles service. Within Blosm, the specific geographic extent was defined, and parameters were set to retrieve detailed models. The below Figure 3.3 shows key settings included:

Source: Google 3D Tiles

Level of details: Buildings with details

Join 3D Tiles objects: Enabled for cohesive mesh management

Relative to initial import: Enabled to simplify spatial management within Blender

Figure 9*Utilizing Blosm Coordinates*

Upon configuration, the "Import" function was activated within Blosm. This triggered the download of the 3D tile data from Google Maps' server, resulting in a highly detailed, geo-referenced 3D model of the CSUDH campus.

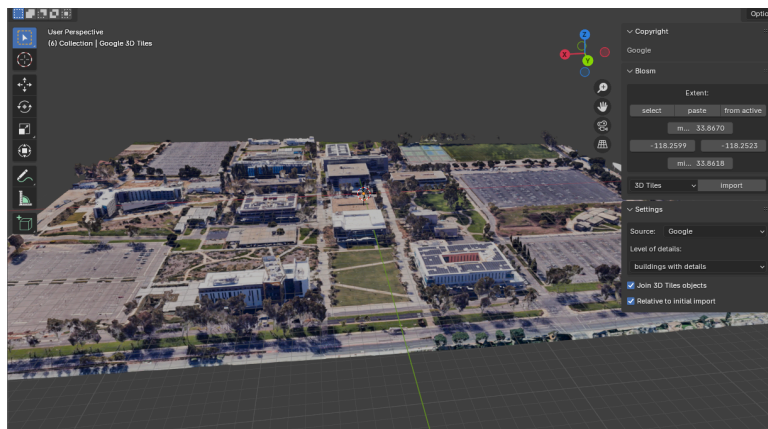
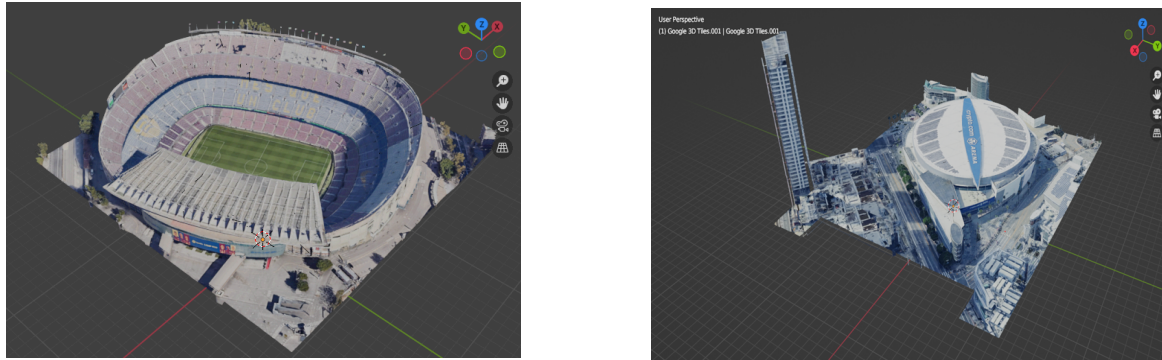
Figure 10*3D Real Time Model of CSUDH Campus utilizing Google 3D Tiles*

Figure 11**Real Time 3D Models of Sport Venues****3D Realistic Models using Esri ArcGIS**

An additional approach employed was the integration of Esri's ArcGIS SDK within Unity, recommended during discussions at a professional AR/VR conference. ArcGIS offered robust geographic visualization capabilities, allowing seamless integration of geospatial data directly into Unity's real-time environment. Integration involved configuring an API key within Unity's Sample Switcher script (Figure 1), defining geographical boundaries using latitude-longitude coordinates, and selecting the local map type to optimize visualization detail (Figure 2). The ArcGIS SDK efficiently loaded high-resolution photogrammetric data, providing superior visual fidelity with aerial imagery resolutions as detailed as 15 centimeters (Figure 3). Despite its strengths in visualization, ArcGIS faced limitations in extracting precise, editable 3D models, restricting practical usage where detailed modification of individual buildings was required (Jones et al., 2020).

Figure 12*3D Map using Photorealistic Tiles*

Create immersive 3D map experiences with
Photorealistic 3D Tiles

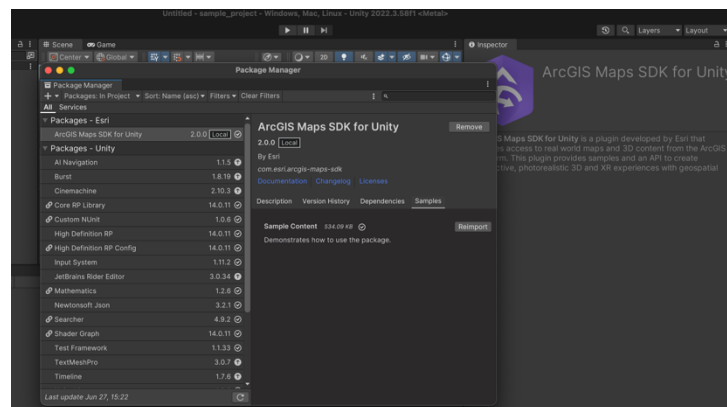
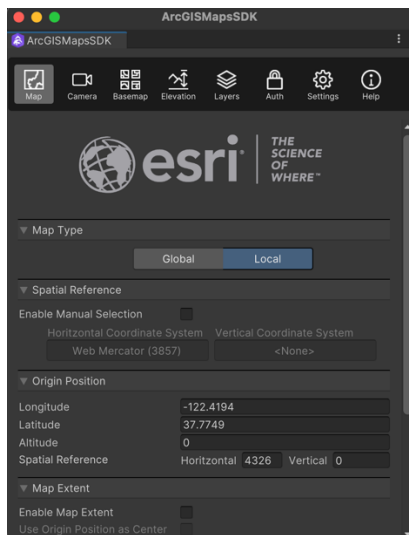
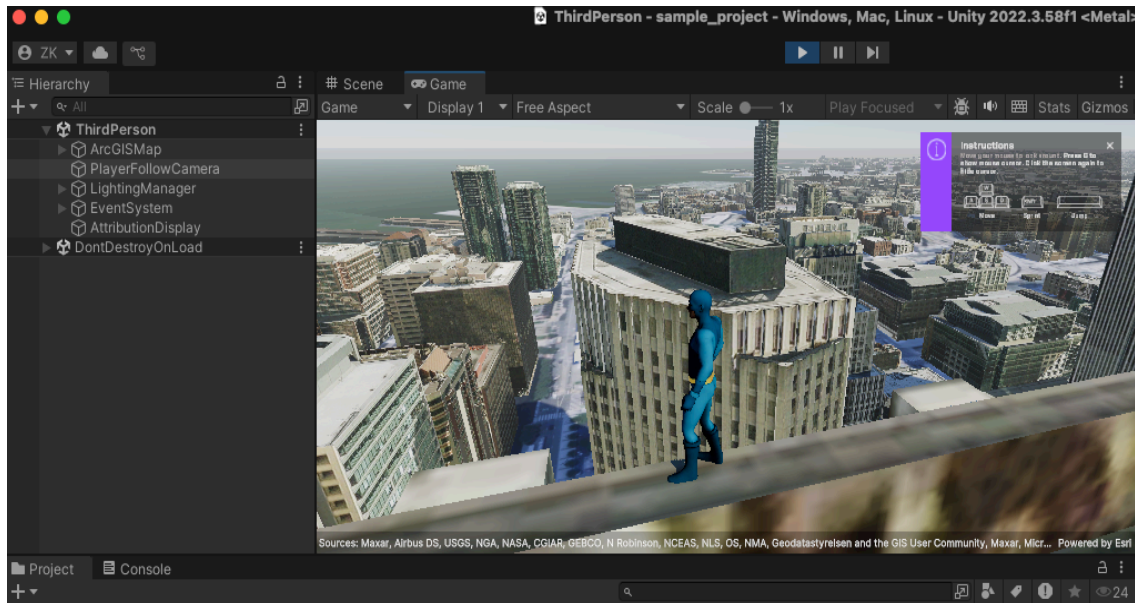
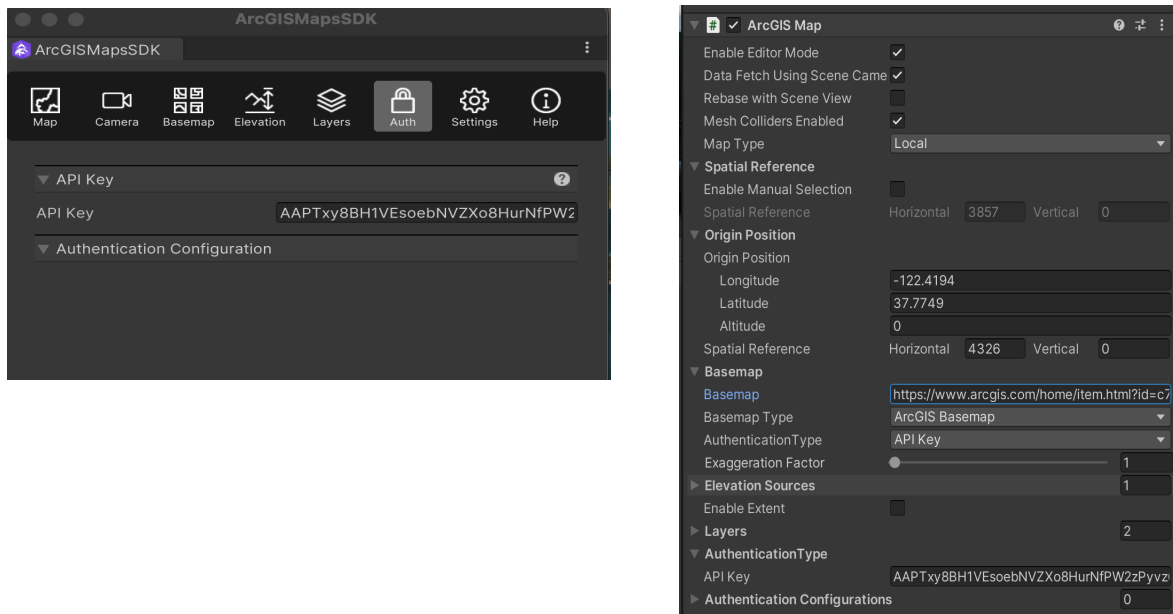
**Figure 13***ArcGIS SDK configuration showing coordinate setup for local map visualization*

Figure 14

Unity's real-time rendering of downtown SG buildings generated using ArcGIS SDK

**Figure 15**

API Key generation and Inspector view of ArcGIS



3D Models using Open-Source 3D Libraries

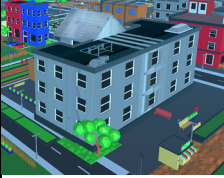
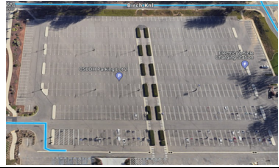
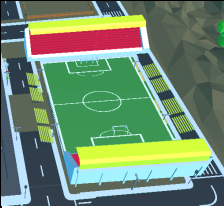
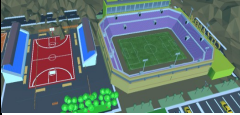
Building models were sourced from reputable online repositories (CGT Trader, Unity Asset Store) providing assets in Unity-compatible FBX format. Roadways, parking lots, and pedestrian pathways were designed carefully, emphasizing connectivity and intuitive layout to ensure smooth navigation for end-users familiar with the campus layout. Textural optimization employed tiling textures, balancing realism and rendering performance efficiently. Directional lighting and street lighting were integrated to further enhance realism, dynamically simulating various times of day to validate the visual effectiveness under different conditions.

Table 2 below provides a comparative evaluation between the realistic campus buildings and their Unity-based virtual counterparts using open-source 3D models using the use case of CSUDH campus, highlighting the accuracy of the modeling process:

Table 2

Comparison of realistic 3D campus with Unity 3D

Building Name	Realistic Building Images	3D Generated Models
I&I		
Welch Hall		

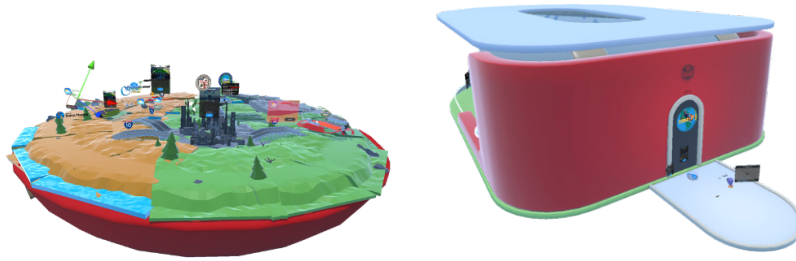
LSU		
Library		
Parking		
Stadium		
Dignity		

3D Models generated in XR Environment

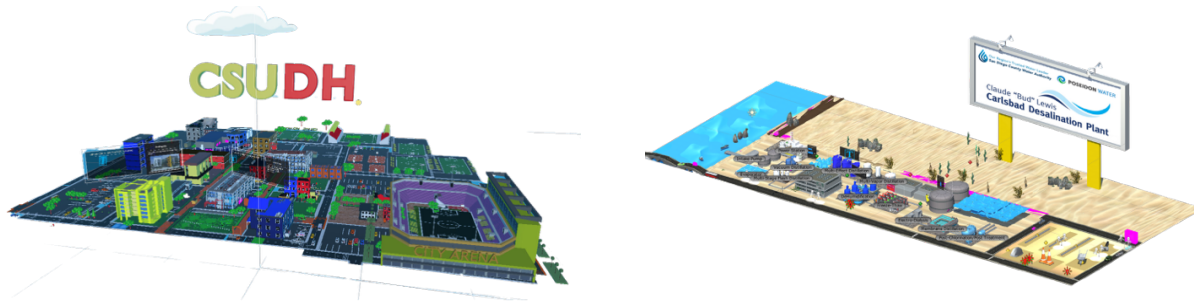
Figure below showcases the integrated 3D campus map developed for immersive XR visualization, effectively demonstrating the outcome of the refined modeling workflow and the integration of optimized, interactive elements.

Figure 16

Teleportation Environment and Landing Plane

**Figure 17**

Campus and Desalination XR Environment



This comparative assessment aligns with contemporary literature emphasizing the necessity of balancing model accuracy, computational efficiency, and financial feasibility in real-time graphics applications (Schwarz & Müller, 2022; Hu et al., 2023). Thus, the methodology ultimately adopted was a hybridized approach, primarily leveraging Open-Source 3D Models for initial captures, manual refinement in Blender, and final integration within Unity. This pipeline successfully enabled the development of interactive and immersive XR environments suitable for practical and educational applications.

CHAPTER 5

IOT SENSOR DEPLOYMENT

Sensor Selection Criteria

Effective integration of real-time data into AR/VR applications depends on the careful selection of IoT sensor devices. Critical factors influencing the selection include portability, accuracy, data acquisition rate, and integration capability. Table 3 below summarizes a comparative analysis between Raspberry Pi-based sensors and LoRa WAN sensor devices.

Table 3

Sensor Device Comparison

Criteria	Raspberry Pi-based Sensors	LoRa WAN Sensor Devices
Connectivity	Wi-Fi/Ethernet/Bluetooth	LoRa WAN (long-range, low-power wireless)
Power Consumption	High (requires power supply or battery management)	Low (optimized for battery operation)
Range	Limited (tens of meters)	Long-range (~15 km rural areas)
Data Frequency	High-frequency updates possible	Low frequency (~15 min intervals)
Cost	Lower base cost, additional modules needed	Higher initial cost (integrated gateway)
Setup Complexity	Moderate-High	Low-Moderate
Scalability	Moderate complexity	Highly scalable
Data Security	Variable security implementation	Built-in encryption and secure protocols

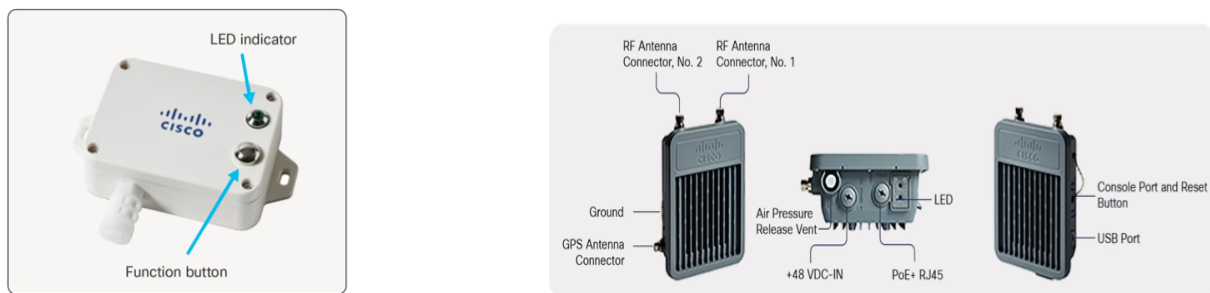
Community Support	Extensive	Growing, particularly in IoT
--------------------------	-----------	------------------------------

Data acquisition involved strategically placing Cisco AV201 LoRa WAN sensors within target environments to capture essential real-time environmental metrics such as temperature, humidity, and occupancy levels. LoRa WAN technology was specifically selected due to its long-range capabilities, low power consumption, and inherent scalability, making it ideal for integration into geographically extensive and complex facilities. Data transmission was efficiently handled by Cisco LoRa WAN gateways, ensuring reliable and uninterrupted data streams even across extended distances of up to 10 kilometers.

Additionally, the telemetry data from sensors was enriched by integrating external API sources, including Google Maps for geographical contextualization and the Meteosource API for weather forecasting and updates. This enriched data provided comprehensive contextual information to enhance user experiences in the XR environment, enabling nuanced and informed interactions.

Figure 18

AV201 sensor device and Cisco IoT Network Gateway

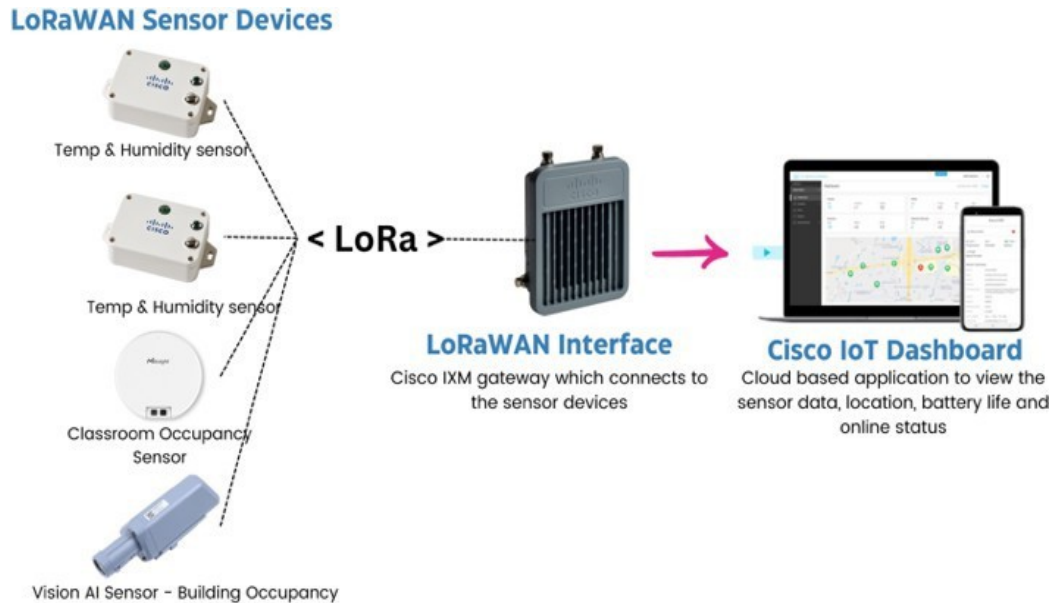


IoT Dashboard

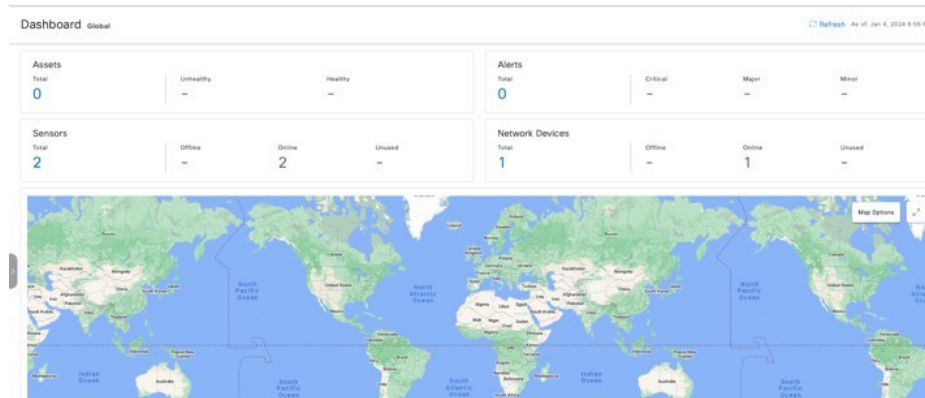
The Cisco IoT LoRa WAN architecture consists of AV201 sensors, gateways, and an IoT Operations Dashboard. The AV201 sensors capture environmental metrics, and the gateway devices manage the long-range wireless data transmission. Figure 19 illustrates this IoT architecture, highlighting data flow from sensors through gateways to the cloud-based dashboard.

Figure 19

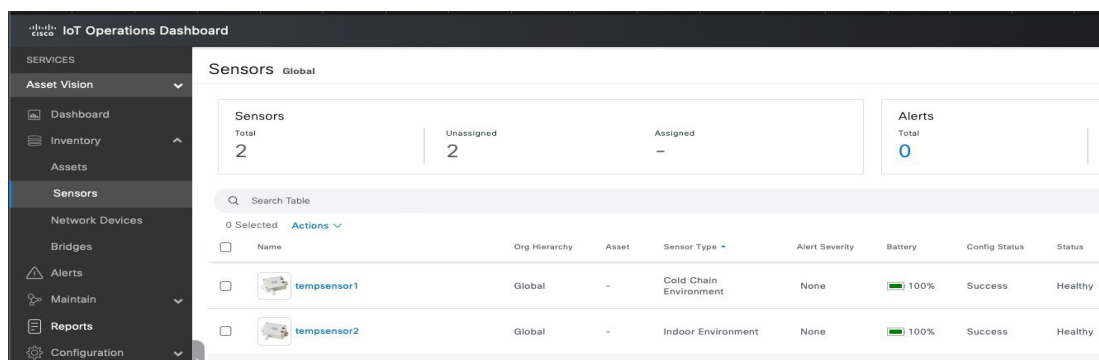
Cisco IoT LoRa WAN Architecture



To provide a centralized, user-friendly interface for managing and monitoring deployed IoT devices, a comprehensive IoT Operations Dashboard was employed. Figure below illustrates the landing page of this dashboard, displaying a map-based visualization of sensor and network device locations. The dashboard offers a holistic overview of the sensors installed across various environments, enabling operators to ascertain sensor status, coverage areas, and signal strength briefly.

Figure 20*Cisco IoT Dashboard Landing Page*

Telemetry data collected from Cisco IoT sensors is initially managed via the Cisco IoT Dashboard, then seamlessly integrated with Microsoft Azure IoT Hub. This integration facilitates streamlined data ingestion and processing. The Cisco IoT Operations Dashboard provides a centralized visualization interface to manage sensors, displaying real-time status, location, and alerts. This approach simplifies operational monitoring and decision-making, crucial for real-time AR/VR visualizations (Hunkeler et al., 2008).

Figure 21*Cisco IoT sensor inventory and devices*

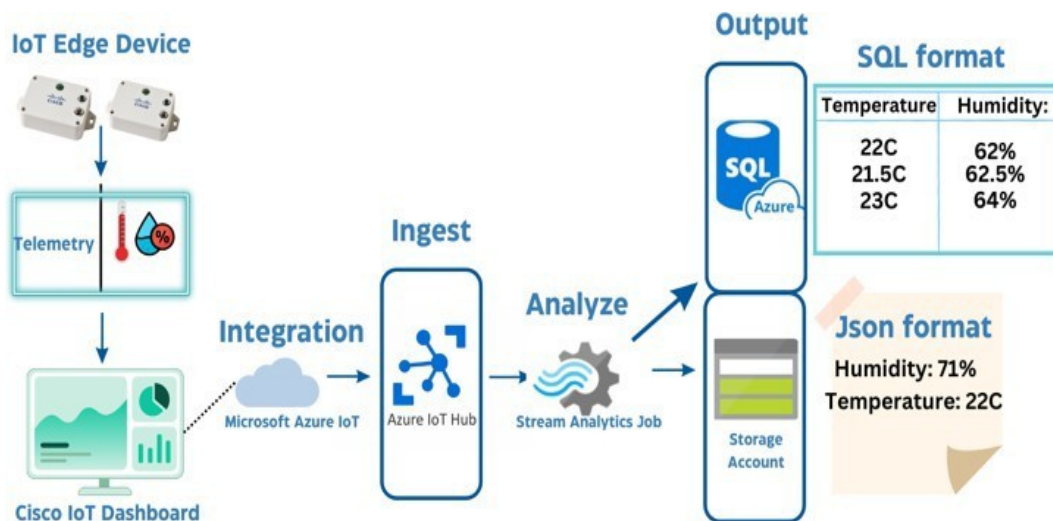
CHAPTER 6

AZURE IOT PROCESSING

Once collected, sensor data was transmitted to Microsoft Azure IoT Hub, where Azure Stream Analytics facilitated real-time processing. Raw telemetry data, initially captured in JSON format, was transformed into structured SQL databases, providing efficient querying and analytical capabilities. This process supported both detailed historical analysis and immediate, real-time updates essential for operational responsiveness.

Figure 22

Azure IoT Integration for SQL based data retrieval pipeline



Initially, direct SQL integration with Unity was employed for immediate visualization of sensor data, allowing quick validation within the Unity Editor. However, due to security and operational limitations especially when deploying to standalone XR devices such as Meta Quest, an API-based middleware solution utilizing Azure Functions was adopted. Azure Functions provided a secure, scalable RESTful API that served structured telemetry data directly to XR

applications. This transition aligned closely with best practices in IoT and XR integration, ensuring data integrity, scalability, and robust security.

Azure Resource Utilization and SQL Database

Azure resources including Stream Analytics and SQL databases are monitored and optimized for performance and scalability. The SQL database enables structured storage of sensor data, facilitating efficient retrieval and integration into AR/VR visualization platforms.

Figure 23

Azure Resources

Resources

Recent Favorite

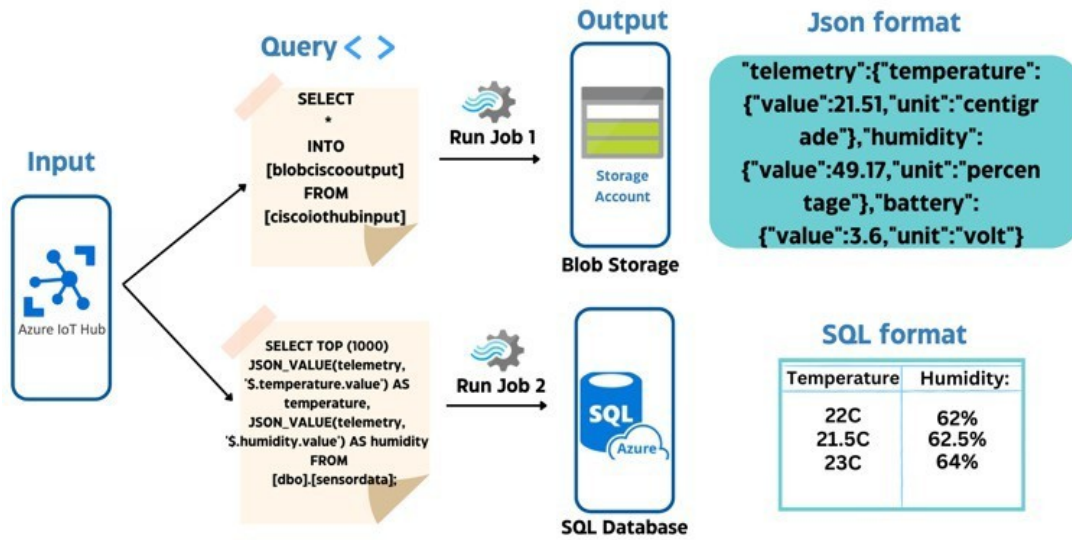
Name	Type	Last Viewed
 ciscoiothub	IoT Hub	a few seconds ago
 zmap	Function App	3 days ago
 zmap	Application Insights	5 days ago
 undefined	Stream Analytics job	5 days ago
 IoTsensordata	SQL database	2 weeks ago
 zmap-apim	API Management service	2 weeks ago
 Microsoft Azure Sponsorship	Subscription	2 weeks ago
 zmap	Resource group	3 weeks ago
 ResourceZaid	Resource group	3 weeks ago
 Azure for Students	Subscription	3 weeks ago
 Azure subscription 1	Subscription	11 months ago
 cs410032001992d24fc	Storage account	11 months ago

[See all](#)

The Azure Stream Analytics Job processes incoming data in real-time, transforming JSON-based telemetry into structured format (SQL tables) for improved querying and reporting. Figure 3.6 illustrates the data processing workflow, wherein temperature and humidity values are extracted from the telemetry, stored in a SQL Database, and made available in JSON format for broader application compatibility.

Figure 24

Azure SQL data processing workflow



CHAPTER 7

REAL TIME IOT DATA INTEGRATION

IoT Integration in Unity

A critical component of the data processing pipeline involved the utilization of an Azure SQL Database instance to store, query, and analyze sensor-generated telemetry. To view the data in SQL format, we created the SQL Database resource. By leveraging a relational database management system, incoming raw data from IoT endpoints such as temperature and humidity readings undergoes structured transformation to facilitate comprehensive analytical workflows. Based on the below query we can view the telemetry data. The data received in the cloud is in JSON format. Therefore, we only need to retrieve the temperature and humidity value (refer Figure below).

```
SELECT TOP (2)
```

```
JSON_VALUE (telemetry, '$. temperature.value') AS temperature, JSON_VALUE (telemetry, '$.humidity.value') AS humidity
```

```
DATEADD (SECOND, CAST (timestamp AS BIGINT)/1000, '1970-01-01')
```

```
AS
```

```
readable_timestamp, FROM
```

```
[dbo].[sensordata] ORDER BY
```

```
readable_timestamp DESC;
```

Figure 25

Structured Format of temperature and humidity with readable timestamp

Query 1 ×

Run ☐ Cancel query

```

1 SELECT TOP (2)
2     JSON_VALUE(telemetry, '$.temperature.value') AS temperature,
3     JSON_VALUE(telemetry, '$.humidity.value') AS humidity,
4     DATEADD(SECOND, CAST(timestamp AS BIGINT)/1000, '1970-01-01') AS readable_timestamp
5 FROM [dbo].[sensordata]
6 ORDER BY readable_timestamp DESC;

```

Results Messages

Search to filter items...

temperature	humidity	readable_timestamp
18.83	51.17	2024-12-08T18:07:28.0000000
18.39	53.41	2024-12-08T18:06:16.0000000

The below pseudocode presents a sample SQL query executed via the Azure Portal's Query Editor.

Using the JSON_VALUE function, the query parses embedded fields (\$.temperature.value and \$.humidity.value) from the JSON payload, converting them into structured columns suitable for downstream analyses. Under the Query editor in the SQL Database, we run the following SQL command as shown below

In the initial testing phase, the Unity Editor was configured to run direct SQL queries against the Azure SQL Database. By employing standard SQL commands, the most recent sensor data entries could be retrieved, including human-readable timestamps and structured JSON fields converted into discrete temperature and humidity metrics.

Unity Editor viewpoint. As we can see from the below Figure, the telemetry data displays Timestamp, temperature and humidity data in the VR campus map. Therefore, we were able to achieve integration of three different programs: from IoT Dashboard to Azure Environment using SSL certificate generation then to Unity Editor using direct SQL database connection. The Azure SQL Connector script pseudocode can be shown utilizes direct SQL database connection. Which does not work when building the application to the XR Headset (Meta Quest Headset). The XR Headset requires API URL which will be used to utilize the connection.

Figure 26

CSUDH virtual map in Unity for sensor data integration Unity



Figure 27

Desalination Map for sensor data integration with Unity

**Figure 28**

Multiplayer environment for sensor data integration in Unity



While effective for local testing, direct SQL database connections from a client-side application, such as Unity, raised operational and security concerns particularly for deployment on standalone devices like the Meta Quest Head-Mounted Display (HMD). Notably, research in IoT and XR integration underscores the importance of secure and scalable data delivery methods, advocating for middleware architectures that abstract database operations behind secure APIs (Xie et al., 2020; Rathore & Son, 2021). These best practices align with well-established guidelines for mobile and VR platforms, where limited system resources and restrictive sandboxed environments often impede direct database connections (Polys et al., 2021).

API Integration

Unity integration with RESTful APIs significantly improved system performance, data security, and compatibility with XR hardware. Real-time telemetry data was effectively visualized within immersive virtual environments, demonstrating smooth interaction and minimal latency. Deployments to Oculus Quest Pro and Apple Vision Pro headsets validated the system's effectiveness, showcasing engaging, intuitive, and responsive user interactions. Teleportation functionality enhanced user experience by providing immediate, realistic transitions between diverse XR environments, significantly improving spatial orientation and interaction quality. By transitioning from direct SQL queries in the Unity Editor to a REST-based API integration via Azure Functions (see Figure x), the solution adheres to established best practices in IoT, cloud, and XR integration. The resulting architecture delivers reliable, secure, and scalable access to real-time telemetry data within the XR environment on the Meta Quest HMD. This step aligns with broader research emphasizing end-to-end security, dynamic scalability, and modular system design in IoT-driven VR/AR applications (Ranjan, 2014; Barbierato et al., 2018; Rathore & Son, 2021).

Figure 29

Utilizing Azure Functions for REST API Integration



The pseudocode shown below outlines a serverless approach to retrieving sensor telemetry data from a cloud-hosted SQL database and delivering it via a secure and scalable endpoint. In this logic, the system first fetches a database connection string from environment variables, then executes a SQL query that extracts the top two most recent temperature and humidity readings, along with human-readable timestamps derived from UNIX epoch time. After successfully reading these values, the pseudocode returns them as structured JSON objects in an HTTP 200 (OK) response. If an error is encountered such as a database connectivity issue or unexpected exception, an HTTP 400 (Bad Request) status is returned, accompanied by a concise error message

Figure 30

Pseudocode Azure Functions for REST API Integration

```
BEGIN function GetTelemetryData (HTTP GET):
    Initialize empty list "data"

    Try:
        Retrieve DB connection string from environment
        Define SQL query to select top 2 recent records (temperature,
            humidity, readable timestamp)
        Open SQL connection
        Execute query and read data:
            For each record:
                Extract temperature, humidity, timestamp
                Add these values to "data" list
        Close SQL connection
        Return HTTP 200 response with "data"

    Catch exceptions:
        Log error message
        Return HTTP error response

END
```

Function Name: The [FunctionName("GetTelemetryData")] attribute assigns a name, "GetTelemetryData," to the Azure Function. This name is used by the Azure Functions runtime to identify and route HTTP requests to this function.

Connection String: The connection String variable retrieves a database connection string from environment variables (e.g., MyDBConnectionString) we created on Azure Function. This connection string contains the necessary details such as server address, database name, and authentication credentials to establish a secure and successful connection to the SQL database.

SQL Query - The query string stores the SQL command that will be executed against the database. In this case, it selects the top two most recent telemetry records from the [dbo].[sensordata] table

Catch Exception Error: We have to publish the Azure Function in Visual Studio with the Azure environment.

Figure 31
Successful publication with the Azure environment using Linux System

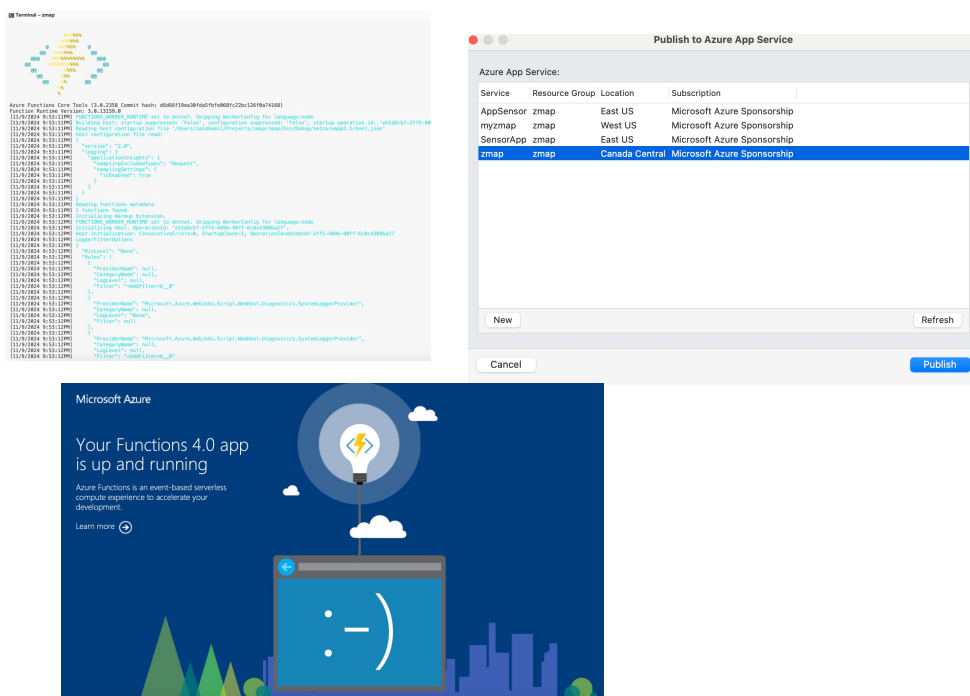
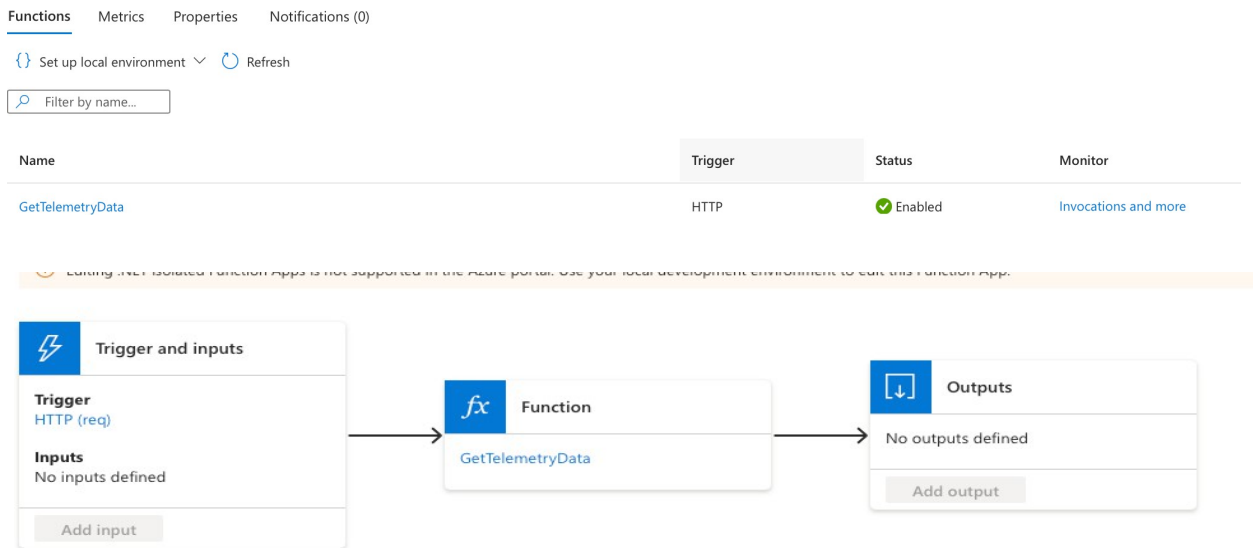
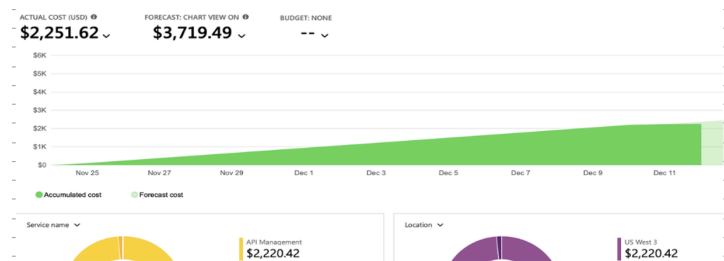


Figure 32*Azure Functions Flowchart*

The HTTP function named "GetTelemetryData" is formed in the Azure Portal interface, the function is shown in a "Running" state with an accessible HTTP endpoint, illustrating how the cloud environment manages both the code and its runtime settings. The event-driven function execution model that simplifies database access, improves maintainability, and supports secure, real-time data retrieval in modern cloud-based IoT solutions

Figure 33

Cost for Azure. \$2,251.62 forecasted monthly for moderate telemetry load

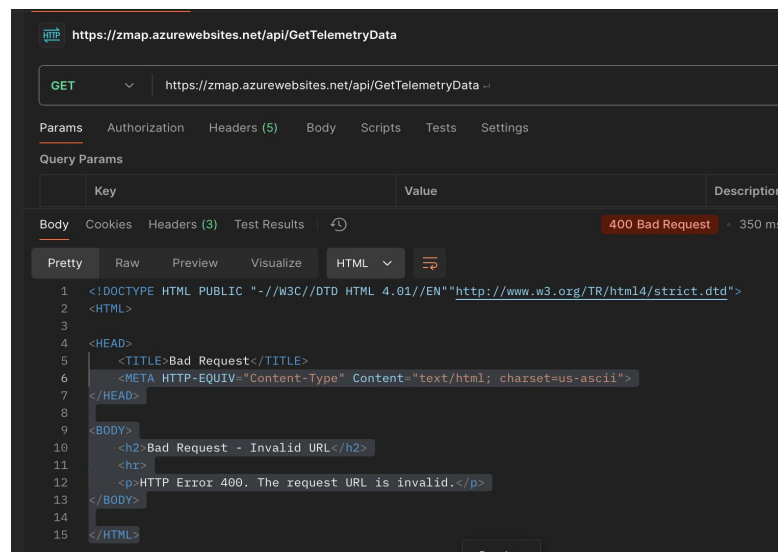


Utilizing Postman Tool

We used Postman, a tool for testing and validating Application Programming Interfaces (APIs), to verify the operational status of the deployed Azure Function endpoint. We created an API using the Azure Function, tested the API on postman, however, the request is unable to connect to the SQL Database. By using Postman as a test client, these issues are surfaced early, providing an opportunity to troubleshoot and refine the API's configuration. Corrective measures might include verifying that the route attribute in the function matches the invoked URL, ensuring that environment variables and connection strings are present and properly configured, reviewing the deployment logs and runtime settings, and confirming that the function indeed returns the expected JSON data under normal conditions.

Figure 34

Testing the API using Postman



`https://zmap.azurewebsites.net/api/GetTelemetryData` was queried via a GET request in Postman, and the response returned a 400 Bad Request error, featuring an HTML-based error message indicating an invalid URL. From the below Figure In the context of Azure Functions, a 400 Bad Request response could signify several potential issues.

Showcasing on XR HMD

After successfully publishing the Azure Function, the REST API URL retrieved for the function was <https://zmap.azurewebsites.net/api/GetTelemetryData>. We used the Function key to make the Rest API calls. However, from testing the API from Postman and utilizing that API URL in Unity to build to the Meta Quest Headset there was Error fetching Data.

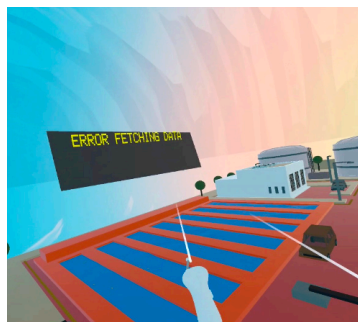
Figure 35

Error fetching data in VR Campus Map Environment built on the Meta Quest HMD



Figure 36

Error fetching data in VR Desalination Environment built on the Meta Quest HMD



Fast API (Local Python Server with Caching)

Given the real-time dependency on telemetry data, a major design concern was the potential failure of upstream providers, such as Error Fetching Data

To address this, a caching mechanism can be introduced within the backend Fast API service to store the last known valid telemetry data. The Fast API logic was updated with a conditional fallback if a live request fails, the last cached result is returned (Figure 36 shows Fast API update pseudocode).

Figure 37

Pseudocode of Fast API app

```
BEGIN FastAPI App

  Allow Cross-Origin Requests (CORS)

  Set Azure SQL DB connection details

  DEFINE HTTP GET endpoint '/telemetry/latest':

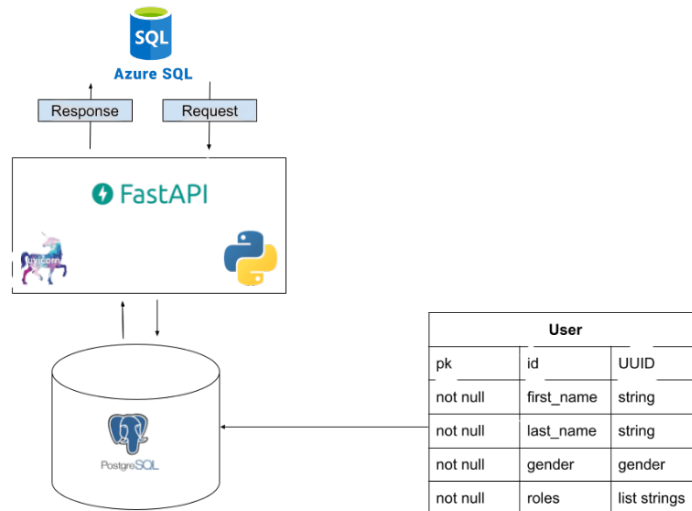
    Connect to DB
    Execute SQL query for latest telemetry (timestamp,
    temperature, humidity)

    IF data exists:
      Return telemetry data as JSON
    ELSE:
      Return "No data found" message

    HANDLE exceptions:
      Return error message as JSON

END FastAPI App
```

In addition, to overcome Azure cost constraints (e.g., \$2,251.62 forecasted monthly for moderate telemetry loads, Figure 5), a Fast API server was deployed on a local network. This allowed full control over routing, caching, and fallback logic.

Figure 38*Workflow diagram of Fast API*

Note: Unity queries a local endpoint /telemetry/latest. Fast API connects to Azure SQL and fetches the most recent readings. JSON output is returned, or a cached value is served if DB is unreachable. Errors are logged locally and returned as structured JSON.

Key Feature: Caching layer added using in-memory storage (e.g., `TTLCache`) to reduce query frequency and ensure resilience during SQL outages or API failures.

Benefits: Extremely fast response time in XR, fully offline-compatible with caching, No recurring cloud fees, easily extensible for multiple endpoints

Challenges: Requires hosting and SSL handling, slightly increased backend complexit

Table 4*API Comparison Table*

Integration Method	Cost	Performance	Fallback Support	Suitable for Production?	Notes
Azure Function	Medium–High	Moderate	✗	✓ with caution	Best for scalable cloud use

Integration Method	Cost	Performance	Fallback Support	Suitable for Production?	Notes
Fast API (Local Server)	Low	High	✓	✓	Ideal for local or hybrid use
Direct SQL (Unity)	Free	High (local)	✗	✗	

Overcoming Firewall Restrictions on Meta Quest XR Devices

While designing immersive, data-driven XR simulations, one critical technical constraint encountered was the inability of the Meta Quest headset to directly access IoT cloud APIs hosted on local servers or behind institutional firewalls. Unlike typical desktop environments, the Meta Quest operates within a mobile-derived Android-based operating system that imposes strict limitations on outbound HTTP requests—particularly to local IP addresses, self-signed certificates, or non-standard ports. This restriction severely impedes real-time data flow from edge or hybrid IoT platforms when not properly routed or hosted.

This project initially relied on a Fast API server running locally, which functioned well within a development LAN but failed when deployed to a standalone Quest device due to these networking constraints. The firewall policies, DNS resolution limitations, and absence of developer proxy settings collectively contributed to blocked API requests and failed telemetry visualizations in standalone XR mode.

These issues are consistent with known architectural barriers in mobile XR development, where devices such as the Meta Quest are unable to resolve private network addresses or access non-public endpoints without tunneling, proxying, or elevated device permissions. Research by Rathore and Son (2021) has emphasized that real-time data exchange in IoT and XR systems is

most effective when hosted over standard web protocols (e.g., HTTPS over port 443), secured by TLS, and delivered via mobile-friendly cloud services.

These firewall limitations can block outbound HTTPS requests or disallow DNS resolution for non-whitelisted domains, effectively severing real-time connectivity to telemetry APIs. Furthermore, Android Web Request classes used in Unity on the Quest cannot resolve localhost or private IPs, ruling out traditional internal API testing approaches. Similar issues have been discussed in edge-computing studies, where secure and scalable IoT-VR communication requires architectural decoupling and edge-aware routing (Rathore & Son, 2021; Polys et al., 2021).

To mitigate these constraints and ensure reliable cloud-to-XR connectivity, several implementation strategies were evaluated:

Port Forwarding with Static IP (Short-Term Workaround)

A temporary solution involved setting up port forwarding on the local router to map a public-facing port (e.g., 443) to the Fast API server's internal IP. This was combined with tunneling services like grok or LocalTunnel, which exposed the server to the public internet using a secure HTTPS URL. Unity was then configured to query the external URL from within the Quest environment. While this enabled data transmission, it introduced latency, dependency on external tunneling services, and frequent disruptions due to dynamic IP address changes. Limitation: Suitable for debugging or classroom demos but not scalable for production due to fragility and security concerns.

Deployment to Cloud Providers

A more robust method involved deploying the API backend (e.g., Fast API) to a public cloud provider such as Azure App Service, Render.com, or Vencel. These platforms offer

HTTPS endpoints by default, overcoming certificate and DNS limitations on Meta Quest. By enabling CORS policies, whitelisting client domains, and applying API tokens, this approach offered a scalable, secure, and universally accessible architecture. Unity clients could fetch real-time telemetry data without being blocked by device-level firewalls or network policies.

Firebase Realtime Database

Another viable route was leveraging Firebase, which offers native SDKs for Unity and integrates well with the Android-based Meta Quest. Firebase's real-time database or Fire store enables bi-directional sync and is designed to bypass many network constraints common to enterprise firewalls. Using Google Play Services for Unity, data can be pushed or pulled directly into XR applications in near real-time. This is particularly effective for mobile-first deployments or educational environments already using Google platforms.

Consideration: Requires migration from SQL schemas or use of connectors such as Firebase Cloud Functions.

MQTT over WebSocket's (IoT-Specific Protocol)

For use cases involving frequent telemetry updates, MQTT—a lightweight publish-subscribe messaging protocol—offers a firewall-friendly option. MQTT brokers like Eclipse Mosquitto can be configured to use WebSocket's over port 443, which is typically allowed by mobile and enterprise firewalls. Unity plugins such as M2MQTTUnity or UnityMQTT support this integration. While MQTT is not a full replacement for REST APIs, it provides efficient, real-time delivery of telemetry data with minimal overhead.

Academic Context

In XR contexts, where user immersion is tightly coupled to data freshness and interface responsiveness, these limitations are even more pronounced. Rathore and Son (2021) further recommend solutions such as mobile-first backend hosting, encrypted transport protocols, and

resilient edge caching mechanisms. This thesis adopts a layered integration model aligned with these best practices to maximize compatibility with Meta Quest and similar XR platforms. Additional approaches include routing traffic through API gateways or reverse proxies such as NGINX or Azure API Management, which provide consistent HTTPS endpoints while abstracting internal logic and retry mechanisms. For environments with persistent restrictions, WebSocket-based communication was explored as a real-time alternative to REST APIs, offering lower latency and persistent connections suitable for XR telemetry (Barbierato et al., 2018). In future iterations, the use of edge computing nodes or Firebase Cloud Messaging (FCM) may offer more robust alternatives for delivering IoT data to constrained XR environments. Collectively, these methods ensure resilient API communication despite the challenges posed by Meta Quest's networking restrictions and institutional firewalls.

CHAPTER 8

REAL TIME TRAFFIC AND WEATHER DATA VISUALIZATION

Integration with external APIs such as Google Directions and Meteosource significantly enhanced contextual realism and usability of the XR applications. Google's Directions API successfully provided detailed real-time traffic visualization across major Southern California freeways. Traffic conditions (smooth, moderate, heavy) were clearly visualized through color-coded 3D models (Figures 39 -42). Similarly, Meteosource's weather data was seamlessly integrated, displaying live weather conditions for multiple locations including Carson, Pasadena, and Anaheim, significantly enhancing environmental contextualization and user awareness.

Traffic Visualization Case Studies

Figure 39

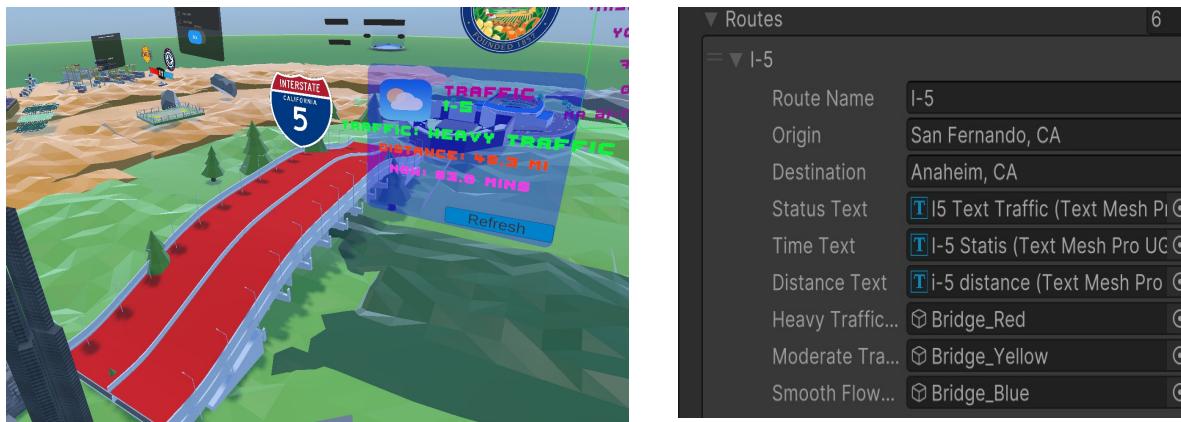
I-10 Freeway VR visualization (Moderate Traffic)



Note: Illustrates moderate traffic conditions on Interstate 10 (I-10) from Santa Monica to Los Angeles. Utilizing Unity's Inspector view (Figure 5.2), developers configure route information, such as origin, destination, and UI elements for displaying traffic status, distance, and travel time. The smooth traffic condition is visualized using a blue-colored 3D model.

Figure 40*I-110 Freeway VR visualization Heavy Traffic*

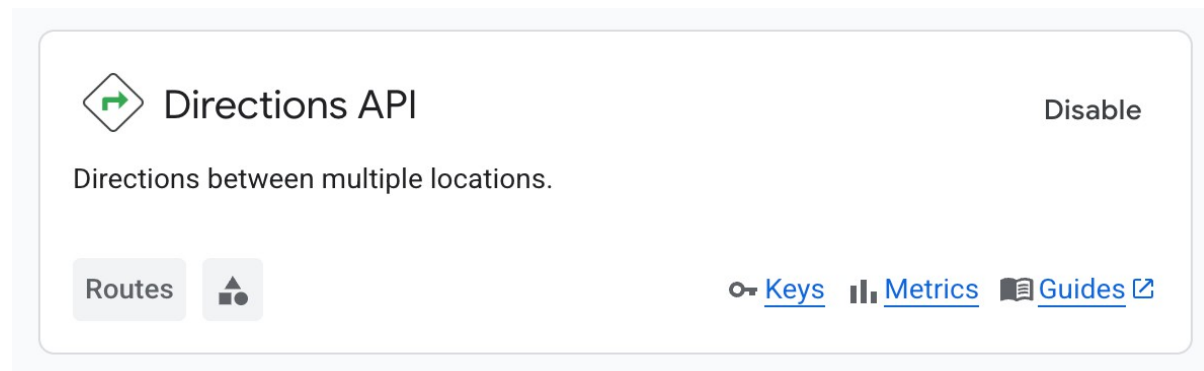
Note: shows heavy traffic conditions from Carson to Los Angeles, depicted with a red-colored 3D bridge model. Figure 5.4 details the inspector view used for configuring origin, destination, and related UI elements. This integration provides users with an immediate and intuitive understanding of severe congestion.

Figure 41*I-5 Freeway VR visualization (Heavy Traffic)*

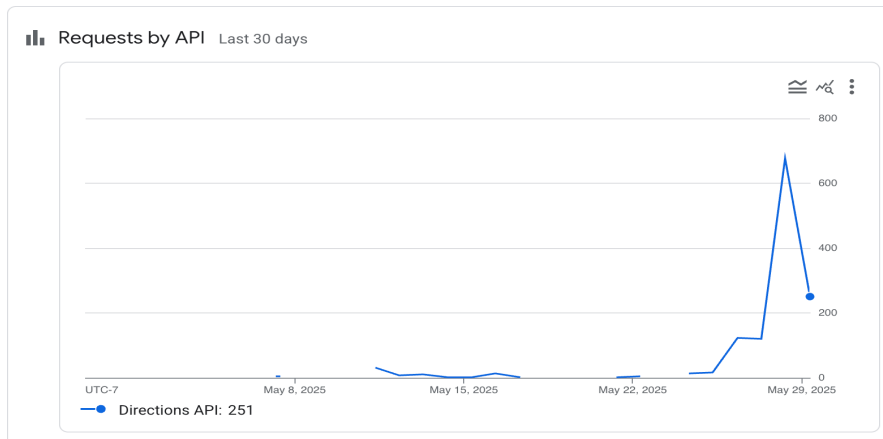
Note: Figure 41 visualizes heavy traffic on Interstate 5 (I-5) from San Fernando to Anaheim. The Unity inspector configuration (Figure 5.6) allows defining the route details and linking visual indicators for real-time updates. Heavy traffic is indicated by red, alerting users to severe delays.

Figure 42*I-210 Freeway VR visualization (Smooth Traffic)*

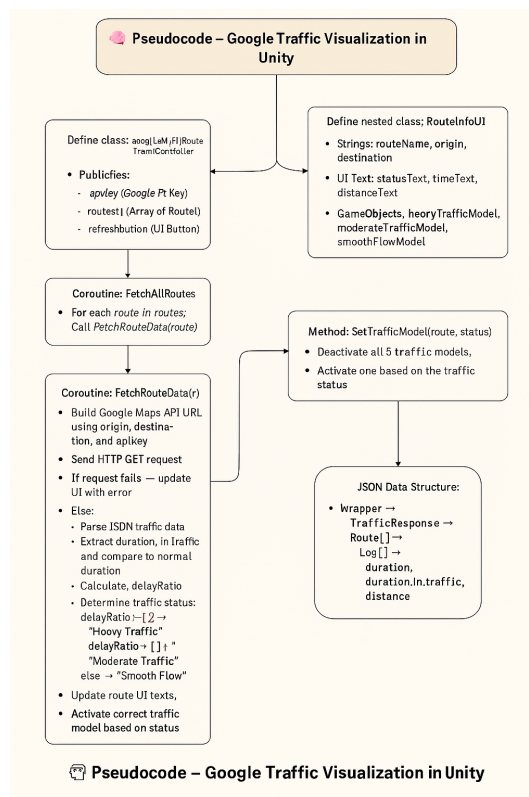
Note: Smooth traffic conditions on Interstate 210 (I-210) from Los Angeles to Pasadena are represented in Figure. This segment is indicated by a blue-colored bridge model, highlighting optimal travel conditions. Inspector configurations allow effortless updates and scalability.

Figure 43*Google Directions API configuration*

Note: Traffic data integration utilizes Google Directions API, configured in the Google Cloud Console. Parameters such as origin, destination, departure time, and traffic model are used to fetch real-time congestion data.

Figure 44*Google Directions API usage metrics*

Note: The API request metrics from Google Cloud Console demonstrate the application's traffic polling activities over the past 30 days, highlighting increased API interactions during system demonstrations and testing periods.

Figure 45*Condensed pseudocode version of Google Traffic Controller script*

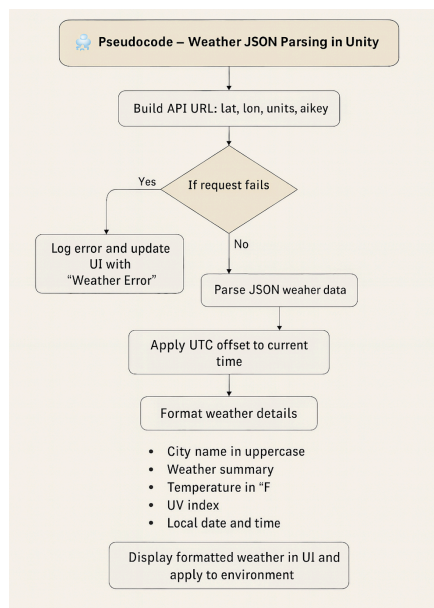
The script handles the core logic of fetching, parsing, and visualizing traffic data. Constructs HTTP GET requests to Google's Directions API Parses JSON responses into structured Unity objects. Classifies traffic conditions into Smooth, Moderate, or Heavy traffic based on delay ratios. Updates Unity UI components and activates appropriate 3D models (blue for smooth, red for heavy traffic).

Real Time Weather Data

We selected the free tier of Meteosource's Point API due to its simplicity, reliable JSON responses, and useful icon assets. Meteosource provides concise JSON weather summaries which are easily integrated into Unity via web requests. An example of a JSON response structure for Carson is provided:

Figure 46

Condensed pseudocode version of weather script



Provide real-time weather data for multiple predefined locations, such as Carson, Miami, Anaheim, and Pasadena. Constructs and sends HTTP requests to the Meteosource API. Parses

JSON responses to extract relevant weather information. Updates the Unity UI elements with real-time data, including weather summary, temperature, UV index, and local timestamps.

Weather Case Studies

Figure 47

The weather visualization in VR display in clear UI panels



Note: The panels indicate the weather summary of Carson, Miami, Anaheim, and Pasadena. (e.g., cloudy or mostly cloudy), temperature, UV index, and timestamp clearly, enhancing user interaction and environmental awareness. These examples validate the accuracy and reliability of the Meteosource API and Unity integration, showing real-time weather conditions effectively.

CHAPTER 9

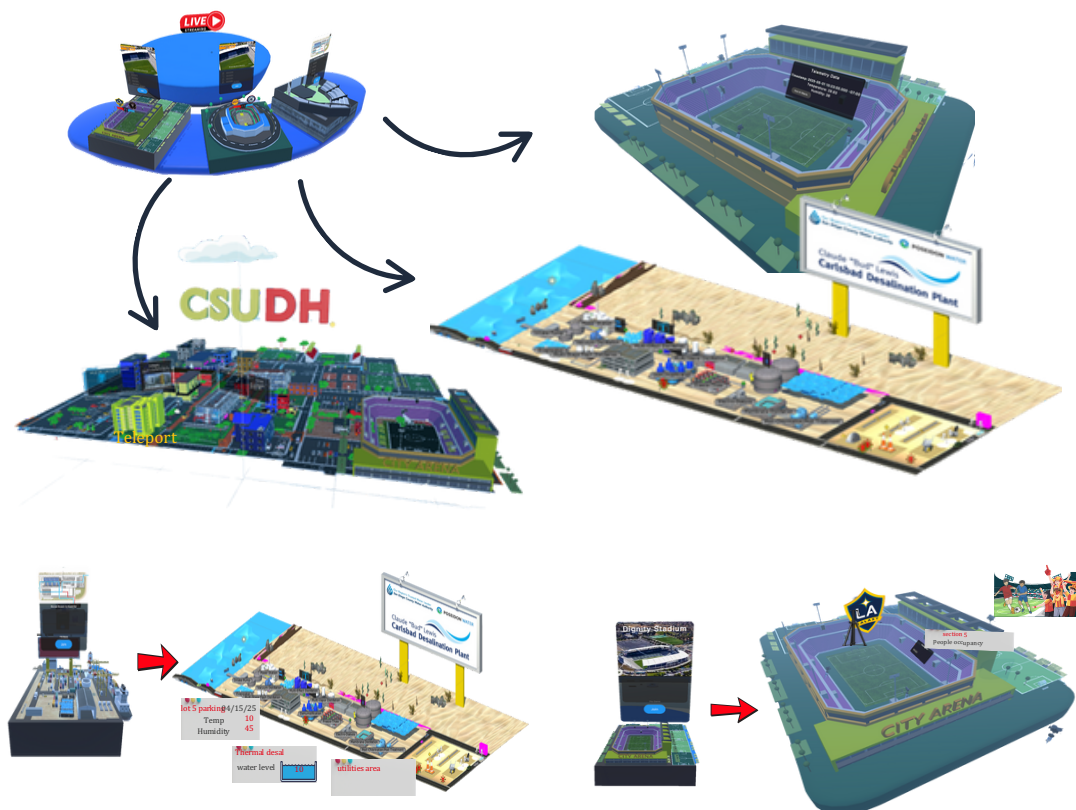
META QUEST VIEW

Teleportation

The Meta Quest view is after the build process from the Unity environment. The application utilizes teleportation functionalities within a single immersive XR application, allowing users to instantaneously navigate between various spatially distinct environments and real-time venues. Providing realistic transitions between diverse XR environments.

Figure 48

Teleportation to different environments.



Note: Allow users to select movement environments zones using ray-based input

Figure 49

Pseudocode of teleportation functionality in Unity

```

BEGIN function RequestTeleport (Trigger teleportation):

    If teleportation anchor transform is set:
        Set teleportation destination position and rotation based on anchor
        Queue teleport request for XR Origin
    Else:
        Return error

END

BEGIN function GenerateTeleportRequestForArea (Generate teleport request for area):

    If raycast hit is valid and not zero:
        Set teleportation destination position from raycast hit point
        Set teleportation destination rotation from area transform
        Return true
    Else:
        Return false

END

```

Figure 50

Landing page environment in Unity and VR Environment

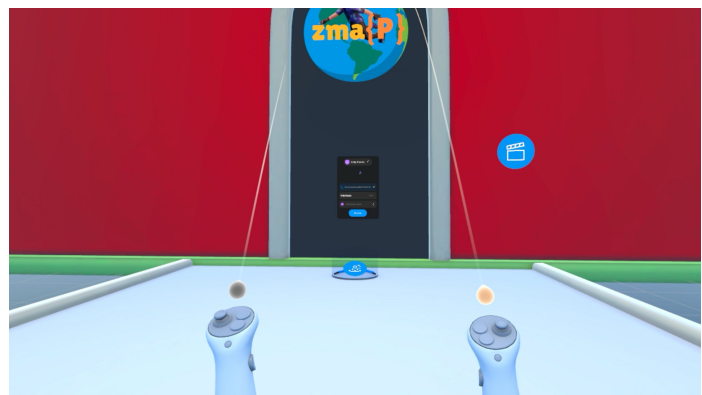


Figure 51

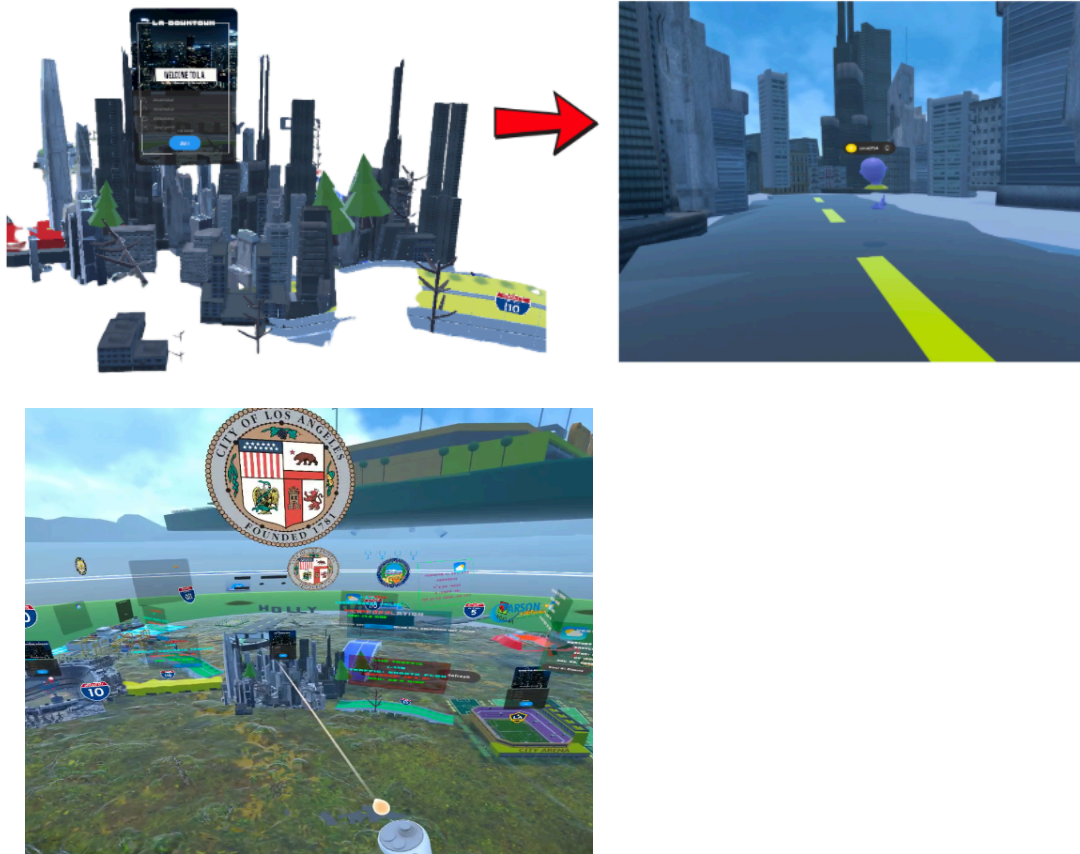
Teleportation to Dodgers Stadium.



Note: Representing real 3D models utilizing real time textures from Open Street Map

Figure 52

Teleportation to Los Angeles downtown environment.



Note: Teleportation allows transition into a detailed virtual representation of the LA Downtown area.

Figure 53

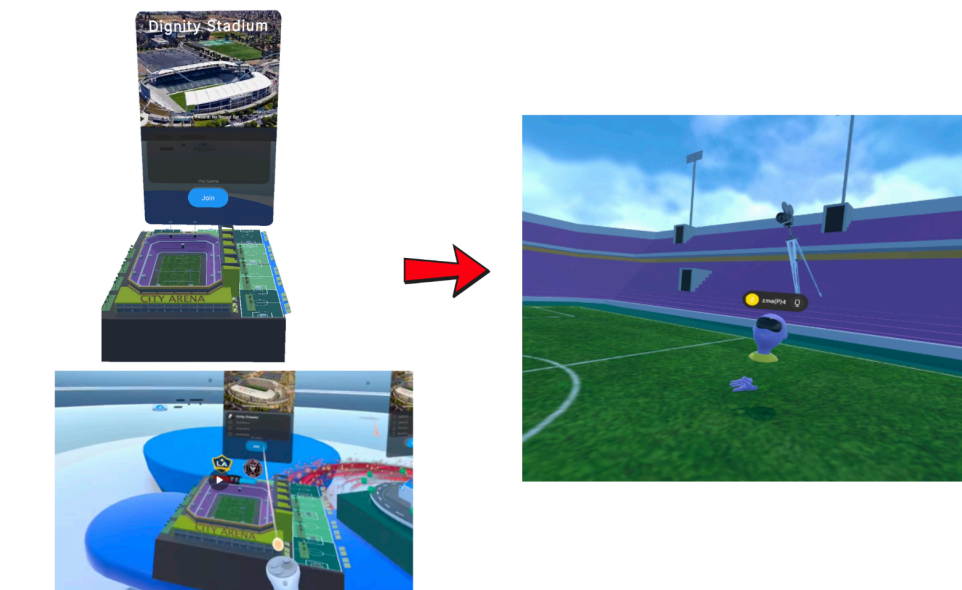
Teleportation to Desalination environment.



Note: Teleportation allows transition into a detailed virtual representation of the Carlsbad Desalination Plant. Upon joining this XR environment, users gain immediate access to real-time telemetry data including temperature, humidity, water levels, and occupancy. This immersive industrial scenario provides critical operational insights essential for facility engineers and managers.

Figure 54

Teleportation to Soccer environment

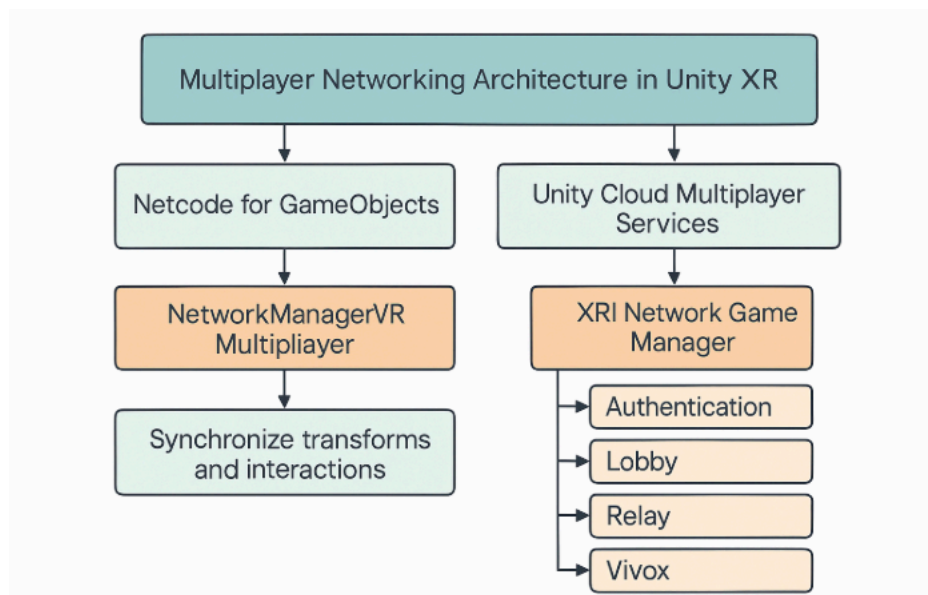


Networking

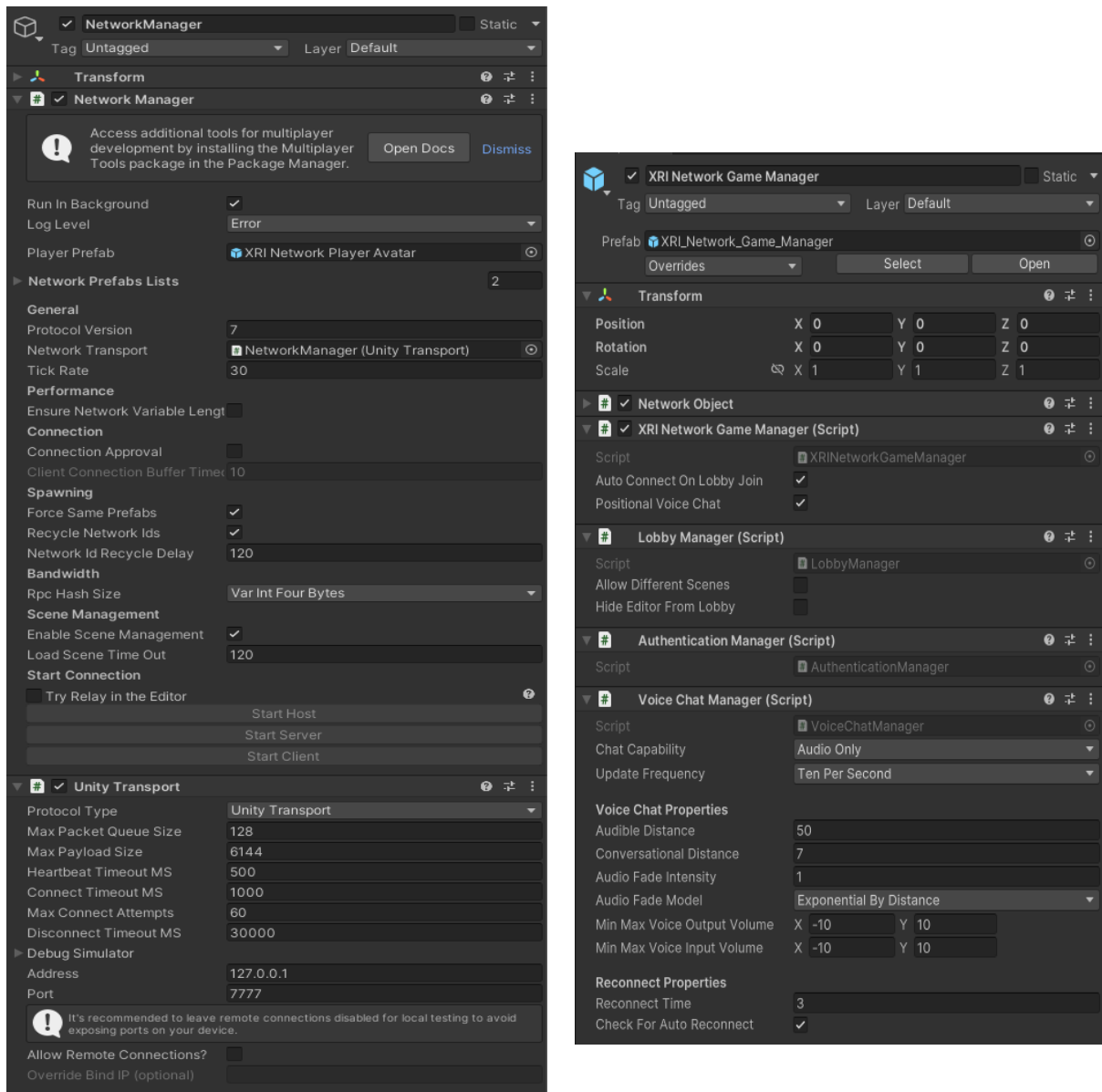
The Unity VR Multiplayer Template addresses this by including a teleportation-based navigation system, preconfigured in its Sample Scene. These mechanics are implemented using components from the XR Interaction Toolkit. In multiplayer sessions, teleportation events are networked using NGO, so that a player's movement is immediately reflected on other clients.

Figure 55

Network Architecture



Note: The networking architecture which includes Networking Stack, Avatar IK Synchronization Flow, Teleportation Trigger System

Figure 56*Network Manager*

Note: It is responsible for tying together the connection functionality between Neet code for Game Objects (NGO) and Unity Cloud's Multiplayer Solutions

This Unity application offers an extensible and production-aligned starting point for,
 Rapid prototyping of multiplayer VR experiments Studying spatial interaction design pattern.
 Evaluating cloud-based networking solutions in immersive contexts

Figure 57

Shared multiplayer experience to view real time data in Los Angeles



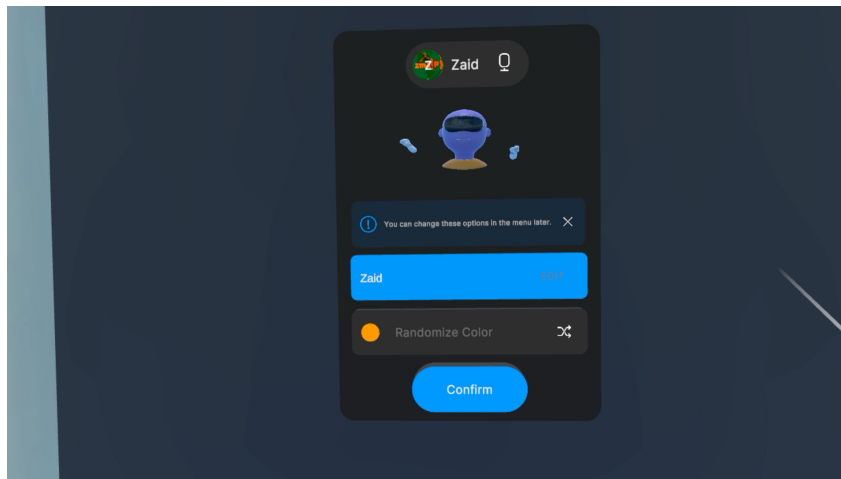
Note: The Multiplayer Synchronization which uses the Client-Server Model: Each user's device (client) sends motion and interaction data to a central server Shows the Avatar based multiplayer system viewing real time traffic and weather data of different environments in the main panel.

User Flow and Room Access

The landing page and avatar customization UI enable users to either create a new room or join an existing one. The system only activates the multiplayer experience once the user has initiated or connected to a room, maintaining a clear flow from UI interaction to active session.

Figure 58

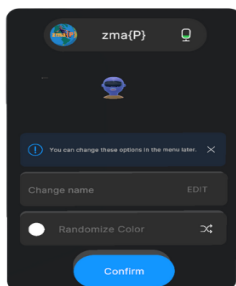
Avatar with the UI overlay in VR



Note: Above shows the critical UI components and avatar control mechanisms that facilitate player customization and interaction within an XR multiplayer environment. The combination of responsive UI elements and IK-driven avatar representation provides a foundation for immersive multiplayer experiences.

Figure 59

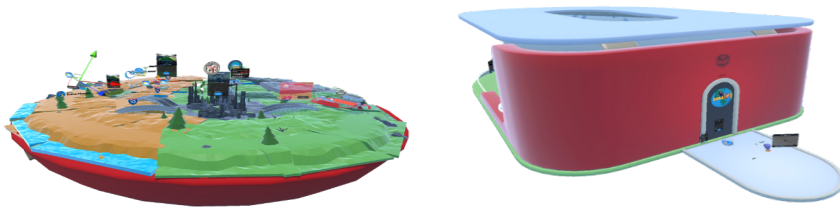
Menu UI and Avatar selection



Note: The Menu UI manages the player's name, color selection, volume indicator, microphone status, and panel toggling between customization and connection. Together, they form the core of the landing page where users engage with the application.

Figure 60

3D XR landing elements developed for XR visualization

**Figure 61**

3D XR environments developed for XR visualization.



CHAPTER 10

ARTIFICIAL INTELLIGENCE WITH XR

AI-driven Virtual Avatars

The integration of AI-driven virtual avatars through the Convai platform substantially improved user interactivity within XR scenarios. Spatially aware avatars demonstrated sophisticated conversational capabilities and dynamic actions, responding effectively to user interactions and environmental cues. Tests conducted within the Meta Quest VR environment confirmed that NPC avatars provided accurate, contextually relevant responses, significantly enhancing realism and user engagement. Unity Inspector configurations allowed detailed customization and robust integration, leading to seamless, natural interactions in the XR space.

Figure 62

Avatar NPC to Player Interaction



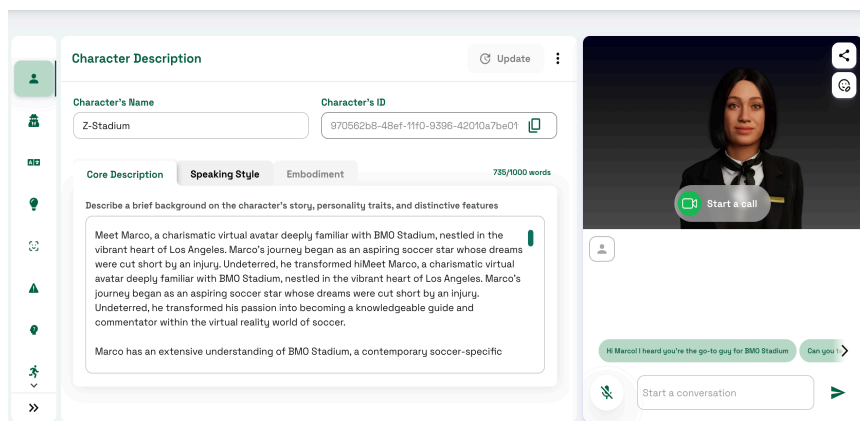
This allows users to communicate naturally with NPC avatars to gain insights about various teleported virtual environments such as cities, campuses, industrial sites, and sports venues, enriching the overall immersive experience

Convai and Knowledge Banks

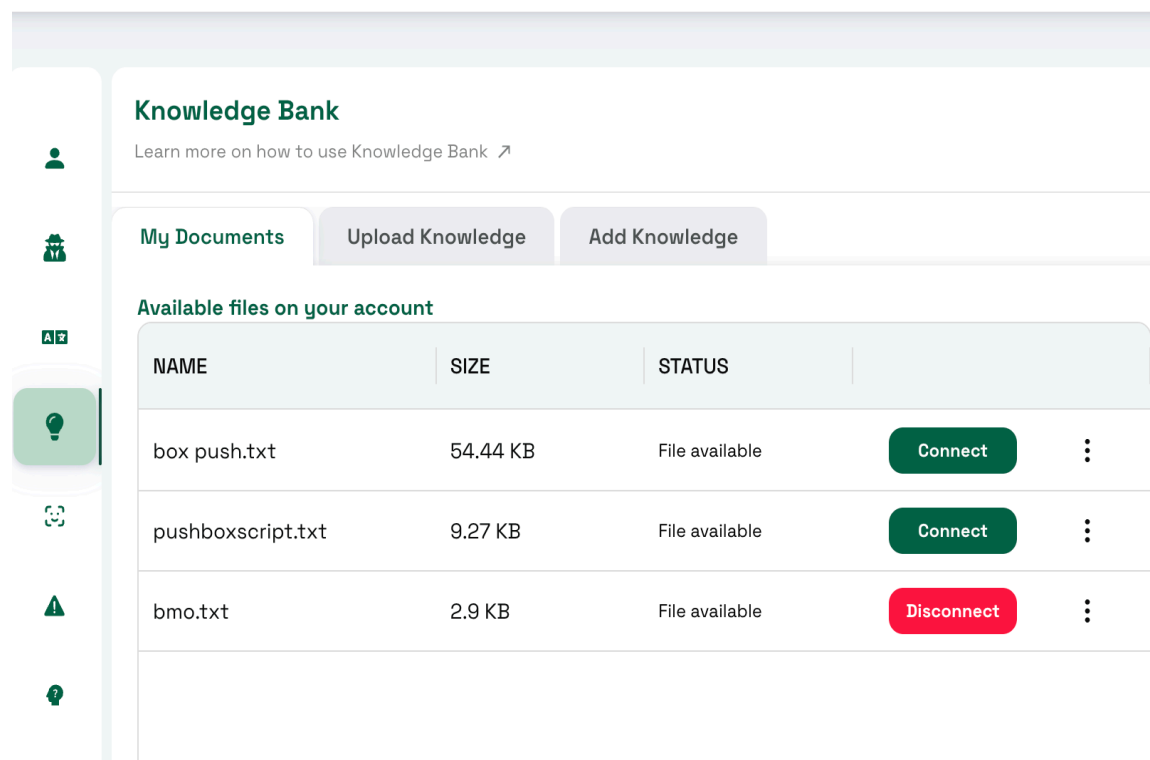
Convai leverages large language models (LLMs) to provide virtual characters with sophisticated conversational capabilities. Despite their advanced text-generation abilities, LLMs often struggle with factual accuracy when queried on specialized topics. Convai's Knowledge Bank addresses this limitation by enabling NPC avatars to draw upon curated, domain-specific repositories of information, significantly enhancing the factual reliability and topical relevance of interactions. A Knowledge Bank acts as a structured repository containing detailed contextual information, including character backgrounds, dialogues, lore, and domain-specific data. This structured knowledge enhances character responses, ensuring coherent and accurate interactions (Convai, 2024).

Figure 63

Creation of AI Avatar in Convai



Note: The virtual character, named Marco, was created on the Convai platform using the Character Creator Tool. Marco was assigned detailed attributes including a defined speaking style, backstory, personality traits, and distinctive features, tailored specifically as a knowledgeable virtual guide for BMO Stadium.

Figure 64*Uploading documents in Convai*

Note: The knowledge bank file was prepared containing information about BMO Stadium, its historical context, venue specifics, and pertinent event details. The curated information was uploaded to Convai's Knowledge Bank as a structured text file.

Knowledge Bank Integration and Verification

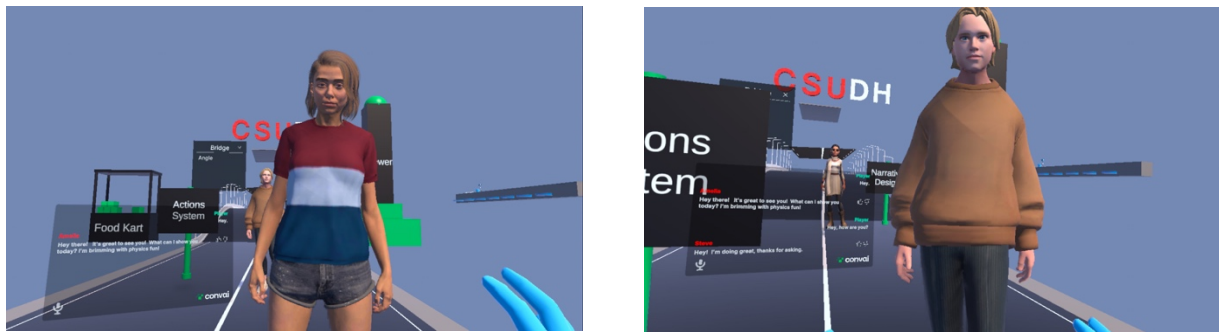
Once uploaded, the knowledge repository was linked directly to Marco. Convai processes the file through a retrieval-augmented generation (RAG) technique, allowing Marco to accurately respond to user queries using information retrieved from the knowledge bank. Subsequent tests confirmed that Marco's responses had significantly improved in accuracy and relevance.

Interaction with AI Avatar in Meta Quest

The implementation of AI-driven NPC avatars using Convai demonstrated significant enhancements in immersive user interactions within the XR environment, as evidenced by practical tests conducted with the Meta Quest VR headset.

Figure 65

Conversations with NPC in XR Environment



Note: Illustrate real-time VR interactions captured directly from the Meta Quest interface. Dialogue logs displayed on-screen provided visual feedback on conversational exchanges, affirming the effectiveness of Convai's knowledge bank and dynamic action integration.

Results and Practical Implementation

Following Convai Knowledge Bank integration, the AI-driven virtual avatar demonstrated marked improvements in conversational accuracy and contextual relevance. Results: AI Avatar Interaction in XR Environment. Users interacting with Marco at various teleported VR locations (campuses, stadiums, cities) could receive reliable, informative responses tailored precisely to the queries presented.

CONCLUSION

This thesis successfully demonstrated the end-to-end system architecture that underlines integration of IoT networks coupled with Google Cloud APIs to provide a scalable pipeline for integrating real-time visualization of data into immersive AR/VR platform. In each stage, conventional and novel element technologies were used that constitute the proposed system architecture. The better solution for acquiring real time data to have the IoT infrastructure in place. The application has potential business revenue, as the manager of BMO stadium is interested to test the application as a pilot using their venue. By leveraging already existing system: such as LoRa WAN sensor technology, 360 cameras, IoT Devices, Azure cloud infrastructure, and Unity-based visualizations, the research addressed critical gaps in the real-time visualization of complex environments, ranging from industrial facilities to academic campuses, sports venues, and urban infrastructures. However, challenges related to the direct integration of IoT data into standalone XR devices through secure RESTful APIs highlighted areas for further improvement. Future research should prioritize refining IoT API connectivity to XR HMD devices.

The next stage of the application is to get user testing and feedback on the application utilizing Meta Quest HMD. After the user feedback, the application can be adopted in Meta Quest store platforms which can be utilized by global users. In addition, the application will be cross built to support Mobile phone view which is accessible to users. Finally, we expect the continued integration of the VR space with IoT real time data will cause a significant transformation of users connection to environments, thereby bringing new business models.

REFERENCES

- Akenine-Möller, T., Haines, E., & Hoffman, N. (2019). *Real-time rendering* (4th ed.). CRC Press.
- Amadeo, M., Campolo, C., Iera, A., & Molinaro, A. (2016). Information-centric networking for the internet of things: Challenges and opportunities. *IEEE Network*, 30(2), 92–100. <https://doi.org/10.1109/MNET.2016.7437030>
- Biljecki, F., Ledoux, H., Stoter, J., & Vosselman, G. (2016). The CityGML standard: A structured overview. *ISPRS International Journal of Geo-Information*, 5(9), 158.
- Chen, J., Liu, B., & Zhou, Z. (2019). Real-time rendering of large-scale urban scenes using GIS-based 3D tiles. *Journal of Computer Graphics Techniques*, 8(3), 29–48.
- Colakovic, A., & Hadzialic, M. (2018). Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues. *Computer Networks*, 144, 17–39. <https://doi.org/10.1016/j.comnet.2018.07.017>
- Core Market Research. (2024). Global XR market trends and forecast. Core Market Research. <http://www.coremarketresearch.com/xr-market-trends>
- Data Insights Market. (2024). Enterprise adoption of XR technologies in training and collaboration. Data Insights Market. <http://www.datainsightsmarket.com/xr-enterprise-adoption>
- Ghazali, A., Abdalla, A. N., & Kamil, A. A. (2020). Educational mixed reality game for teaching water desalination processes. *International Journal of Emerging Technologies in Learning (iJET)*, 15(5), 132–146. <https://doi.org/10.3991/ijet.v15i05.12595>
- Hernandez, M., Bianchi, F., & Urdiales, C. (2022). IoT and cloud-based predictive maintenance using Azure Stream Analytics. *Sensors*, 22(3), 1029. <https://doi.org/10.3390/s22031029>

- Hong, Y., Xue, Y., & Zhao, J. (2020). Integration of IoT sensors and Azure Functions for real-time monitoring applications. *IEEE Internet of Things Journal*, 7(10), 10032–10041. <https://doi.org/10.1109/JIOT.2020.2992008>
- Hu, W., Cheng, H., & Wei, L. (2023). Optimization strategies for texture baking in real-time graphics applications. *ACM Transactions on Graphics*, 42(2), Article 21.
- Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). MQTT-S—A publish/subscribe protocol for wireless sensor networks. *IEEE International Conference on Communication System Software and Middleware*, 791–798. <https://doi.org/10.1109/COMSWA.2008.4554519>
- Jones, C., et al. (2020). Integration of GIS data into game engines for immersive 3D environments. *Journal of Geovisualization and Spatial Analysis*, 4(1), 1–15.
- Kolomvatsos, K., Anagnostopoulos, C., & Hadjiefthymiades, S. (2018). Data stream analytics in the Internet of Things (IoT): A survey. *IEEE Communications Surveys & Tutorials*, 20(4), 2864–2896. <https://doi.org/10.1109/COMST.2018.2848941>
- Kuvshinnikov, I., et al. (2024). Virtual reality training simulator for non-destructive testing in radiographic inspection. *Journal of Industrial Engineering*, 2024, Article ID 564890. <https://doi.org/10.1155/2024/564890>
- Langote, S., Singh, A., & Rathore, S. (2023). Scalable architecture for IoT sensor networks using LoRaWAN technology. *IEEE Transactions on Industrial Informatics*, 19(4), 3460–3469. <https://doi.org/10.1109/TII.2023.3235698>
- Li, X., Liu, Y., & Xia, F. (2020). Cloud computing in the Internet of Things: Architecture, applications, and challenges. *IEEE Communications Surveys & Tutorials*, 22(4), 2460–2485. <https://doi.org/10.1109/COMST.2020.3002605>

- Lockner, Y., & Atzori, L. (2021). IoT data storage and analytics in Azure cloud: Performance and scalability evaluation. *Journal of Cloud Computing*, 10(1), 36.
<https://doi.org/10.1186/s13677-021-00267-0>
- Meteosource API Documentation. (2025). Meteosource. Retrieved May 29, 2025, from
<https://www.meteosource.com/api>
- Mól, A. C. A., Jorge, C. A. F., & Couto, P. M. (2008). Using a game engine for VR simulations in nuclear plants. *Computers & Graphics*, 32(2), 195–202.
<https://doi.org/10.1016/j.cag.2007.12.007>
- Mordor Intelligence. (2024). Virtual, augmented, and mixed reality market – growth, trends, COVID-19 impact, and forecasts (2024–2030). Mordor Intelligence.
<http://www.mordorintelligence.com/industry-reports/virtual-augmented-and-mixed-reality-market>
- Pfetter, M., et al. (2015). Virtual reality enhanced industrial operations for plant monitoring and maintenance. *International Journal of Advanced Manufacturing Technology*, 80(1–4), 485–496. <https://doi.org/10.1007/s00170-015-7052-1>
- Polys, N. F., North, C., & Bowman, D. A. (2021). Immersive analytics: Theory and research agenda. *Frontiers in Virtual Reality*, 2, Article 690821.
<https://doi.org/10.3389/frvir.2021.690821>
- Rathore, S., & Son, H. (2021). RESTful API integration for scalable IoT data management. *Journal of Web Engineering*, 20(6), 1509–1523. <https://doi.org/10.13052/jwe1540-9589.2069>

Schwarz, M., & Müller, T. (2022). Evaluating mesh optimization techniques for real-time rendering. *IEEE Transactions on Visualization and Computer Graphics*, 28(9), 3564–3575.

Scoop Market. (2024). Impact of 5G networks on XR adoption. Scoop Market Intelligence.
<http://www.scoopmarket.com/reports/5g-impact-xr>

Singh, S., Chatterjee, P., & Khemani, D. (2020). Wireless sensor networks for predictive maintenance of mobile industrial assets. *Sensors*, 20(23), 6779.

Unity Technologies. (2024). UnityWebRequest documentation. Retrieved May 29, 2025, from
<https://docs.unity3d.com/Manual/UnityWebRequest.html>

APPENDIX: SCRIPTS

XR Multiplayer.cs

```

using Unity.Netcode;
using UnityEngine;

#if UNITY_EDITOR
using UnityEditor;
#endif

namespace XRMultiplayer
{
    /// <summary>
    /// Manages the network functionality for VR multiplayer.
    /// </summary>
    public class NetworkManagerVRMultiplayer : NetworkManager
    {
        [SerializeField, Tooltip("Set this to control how much logging is generated")]
        LogLevel m_LogLevel;

        [SerializeField, Tooltip("This should almost always be set to true")]
        bool m_RunInBackground = true;

        [SerializeField] NetworkConfig
        m_NetworkConfig;

        ///<inheritdoc/>
        void Awake()
        {
            LogLevel = m_LogLevel;
            RunInBackground = m_RunInBackground;
            NetworkConfig = m_NetworkConfig;
            Utils.s_LogLevel = LogLevel;
        }
    }
}

#if UNITY_EDITOR
[CustomEditor(typeof(NetworkManagerVRMultiplayer))] class
VRMultiplayerTemplateNetworkManagerEditor : Editor

```

```

base.OnInspectorGUI();

if (Application.isPlaying)
{
    switch (XRINetworkGameManager.CurrentConnectionState.Value)
    {
        case XRINetworkGameManager.ConnectionState.None:
            GUILayout.Box("Authenticating");
            break;
        case XRINetworkGameManager.ConnectionState.Authenticating:
            GUILayout.Box("Authenticating");
            break;
        case
            XRINetworkGameManager.ConnectionState.Authenticated
            : if (GUILayout.Button("Connect"))
                {
                    XRINetworkGameManager.Instance.QuickJoinLobby();
                }
            break;
        case XRINetworkGameManager.ConnectionState.Connecting:
            GUILayout.Box("Connecting");
            break;
        case
            XRINetworkGameManager.ConnectionState.Connected
            : if (GUILayout.Button("Disconnect"))
                {
                    XRINetworkGameManager.Instance.Disconnect();
                }
            break;
    }
}
else
{
    GUILayout.Box("Game not running.");
}

```

GoogleMultiRoute.cs

```

using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;
using TMPro;

```

```

using System.Collections;

public class GoogleMultiRouteTrafficController : MonoBehaviour
{
    [System.Serializable]
    public class RouteInfoUI
    {
        public string routeName;
        public string origin;
        public string destination;
        public TMP_Text statusText;
        public TMP_Text timeText;
        public TMP_Text distanceText;

        public GameObject heavyTrafficModel; // Prefab for Heavy Traffic
        public GameObject moderateTrafficModel; // Prefab for Moderate Traffic
        public GameObject smoothFlowModel; // Prefab for Smooth Flow
    }

    [Header("Google API")]
    public string apiKey;

    [Header("Routes to Track")]
    public RouteInfoUI[] routes;

    [Header("Refresh Controls")]
    public Button refreshButton;
    public float refreshIntervalSeconds = 300f; // Every 5 minutes

    void Start()
    {
        if (refreshButton != null)
            refreshButton.onClick.AddListener(() => StartCoroutine(FetchAllRoutes()));

        StartCoroutine(FetchAllRoutes());
        InvokeRepeating(nameof(AutoRefresh), refreshIntervalSeconds,
refreshIntervalSeconds);
    }

    void AutoRefresh()
    {
        StartCoroutine(FetchAllRoutes());
    }
}

```

```

        yield return FetchRouteData(route);
    }
}

IEnumerator FetchRouteData(RouteInfoUI route)
{
    string url =
$"https://maps.googleapis.com/maps/api/directions/json?origin={route.origin}&destination={route.destination}&departure_time=now&traffic_model=best_guess&key={apiKey}";
    UnityWebRequest req = UnityWebRequest.Get(url);
    yield return req.SendWebRequest();

    if (req.result != UnityWebRequest.Result.Success)
    {
        route.statusText.text = $"{route.routeName}\nError: {req.error}";
        route.timeText.text = "";
        route.distanceText.text = "";
        yield break;
    }

    try
    {
        string raw = req.downloadHandler.text;
        TrafficResponse data =
JsonUtility.FromJson<Wrapper>($"{{\"data\":{raw}}}")?.data;
        var leg = data.routes[0].legs[0];

        float minutes = leg.duration_in_traffic.value / 60f;
        string distance = leg.distance.text;

        // Interpret traffic severity
        float delayRatio = (float)leg.duration_in_traffic.value /
leg.duration.value;
        string status = delayRatio > 1.3f ? "Heavy Traffic"
            : delayRatio > 1.1f ? "Moderate Traffic"
            : "Smooth Flow";

        // Update UI
        route.statusText.text = $"{route.routeName}\nTraffic: {status}";
        route.timeText.text = $"Now: {minutes:F1} mins";
        route.distanceText.text = $"Distance: {distance}";

        // Update the active model based on the traffic condition
        SetTrafficModel(route, status);
    }
}

```

```

        catch (System.Exception ex)
        {
            route.statusText.text = $"{route.routeName}\nError: Parse Failed";

            Debug.LogError($"Traffic JSON parse error: {ex.Message}");
        }
    }

void SetTrafficModel(RouteInfoUI route, string status)
{
    // Deactivate all models first
    route.heavyTrafficModel.SetActive(false);
    route.moderateTrafficModel.SetActive(false);
    route.smoothFlowModel.SetActive(false);

    // Activate the model based on the traffic status
    if (status == "Heavy Traffic")
    {
        route.heavyTrafficModel.SetActive(true);
    }
    else if (status == "Moderate Traffic")
    {
        route.moderateTrafficModel.SetActive(true);
    }
    else if (status == "Smooth Flow")
    {
        route.smoothFlowModel.SetActive(true);
    }
}

// JSON Helpers
[System.Serializable] public class Wrapper { public TrafficResponse data; }
[System.Serializable] public class TrafficResponse { public Route[] routes; }
[System.Serializable] public class Route { public Leg[] legs; }
[System.Serializable] public class Leg
{
    public Duration duration;
    public Duration duration_in_traffic;
    public Distance distance;
}

```

MultiWeather.cs

```

using UnityEngine;
using UnityEngine.Networking;

```

```

using UnityEngine.UI;
using TMPro;
using System.Collections;
using System;

public class MultiCityWeatherController : MonoBehaviour
{
    [System.Serializable]
    public class CityWeatherUI
    {
        public string cityName;
        public float latitude;
        public float longitude;
        public TMP_Text weatherText;
        public RawImage weatherIcon;
    }

    public CityWeatherUI[] cities;
    public TMP_Text debugConsole;
    public Button refreshButton;
    public WeatherEnvironmentController envController;

    private string apiKey = "z9y3uvc040njcuxlyi7jxhqicfeqczd47hucv9bv";
    private float refreshInterval = 600f;

    void Start()
    {
        if (refreshButton != null)
            refreshButton.onClick.AddListener(() => StartCoroutine(FetchAllCities()));

        StartCoroutine(FetchAllCities());
        InvokeRepeating(nameof(StartCityLoop), refreshInterval, refreshInterval);
    }

    void StartCityLoop()
    {
        StartCoroutine(FetchAllCities());
    }

    IEnumerator FetchAllCities()
    {
        foreach (var city in cities)
        {
            yield return FetchWeather(city);
        }
    }
}

```

```

{
    string apiUrl =
$"https://www.meteosource.com/api/v1/free/point?lat={city.latitude}&lon={city.longitude}&sections=current&timezone=auto&language=en&units=us&key={apiKey}";
    Log($"Requesting weather for {city.cityName} from: {apiUrl}");

    UnityWebRequest www = UnityWebRequest.Get(apiUrl);
    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        LogError($"Weather API Error for {city.cityName}: {www.error}");
        city.weatherText.text = $"{city.cityName}\nWeather Error";
    }
    else
    {
        try
        {
            string json = www.downloadHandler.text;
            WeatherDataWrapper data =
JsonUtility.FromJson<WeatherDataWrapper>(json);
            var w = data.current;

            // Apply manual UTC offset
            float offset = GetUtcOffset(city.cityName);
            DateTime localTime = DateTime.UtcNow.AddHours(offset);

            string display = $"Weather in {city.cityName.ToUpper()}: \n" +
                $"{w.summary} \n" +
                $"Temp: {w.temperature}°F \n" +
                $"UV Index: {w.uv_index} \n" +
                $"{localTime:MMM dd, yyyy hh:mm tt}";
            city.weatherText.text = display;

            envController?.ApplyWeather(w.summary.ToLower(), localTime.Hour);

            if (!string.IsNullOrEmpty(w.icon))
            {
                string iconUrl =
$"https://www.meteosource.com/static/img/weather_icons/{w.icon}.png";
                StartCoroutine(LoadIcon(iconUrl, city.weatherIcon));
            }
        }
        catch (Exception ex)
    }
}

```



```

        {
            LogError($"JSON Parse Error for {city.cityName}: {ex.Message}");
            city.weatherText.text = $"{city.cityName}\nWeather Error";
        }
    }

float GetUtcOffset(string cityName)
{
    switch (cityName.ToLower())
    {
        case "carson": return -7f;    // PDT
        case "miami":  return -4f;    // EDT
        case "tennessee": return -5f; // CDT
        default: return -7f;
    }
}

IEnumerator LoadIcon(string url, RawImage image)
{
    UnityWebRequest iconReq = UnityWebRequestTexture.GetTexture(url);
    yield return iconReq.SendWebRequest();

    if (iconReq.result == UnityWebRequest.Result.Success)
    {
        Texture2D icon =
            ((DownloadHandlerTexture)iconReq.downloadHandler).texture;
        if (image != null) image.texture = icon;
    }
    else
    {
        LogError("Icon Load Error: " + iconReq.error);
    }
}

void Log(string msg)
{
    Debug.Log(msg);
    if (debugConsole != null) debugConsole.text = msg + "\n" + debugConsole.text;
}

void LogError(string msg)
{
    Debug.LogError(msg);
    if (debugConsole != null) debugConsole.text = "<color=red>" + msg +

```

```
        "</color>\n" + debugConsole.text;
    }

    [System.Serializable]
    public class WeatherDataWrapper
    {
        public CurrentWeather current;
    }

    [System.Serializable]
    public class CurrentWeather
```

SQLConnector.cs

```
// Full corrected C# script

using UnityEngine;
using UnityEngine.Networking;
using TMPro;
using System.Collections;

public class TelemetryAPI : MonoBehaviour
{
    public TMP_Text II_telemetry_Text;
    public TMP_Text LSU_telemetry_Text;
    public TMP_Text DebugConsole; // Optional debug console display in headset

    private string apiUrl;
    private float refreshInterval = 10f; // 10 seconds for quick testing (can set to
    900f = 15 min for production)

    void Start()
    {
#if UNITY_EDITOR
        apiUrl = "http://localhost:8000/telemetry/latest";
#elif UNITY_ANDROID
        apiUrl = "http://192.168.1.120:8000/telemetry/latest"; // Update if IP changes
#else
        apiUrl = "http://192.168.1.120:8000/telemetry/latest";
#endif

        Log("API URL in use: " + apiUrl);
    }
}
```

```

IEnumerator FetchTelemetryLoop()
{
    while (true)
    {
        yield return FetchTelemetry();
        yield return new WaitForSeconds(refreshInterval);
    }
}

IEnumerator FetchTelemetry()
{
    Log("Sending request to: " + apiUrl);
    UnityWebRequest www = UnityWebRequest.Get(apiUrl);
    yield return www.SendWebRequest();

    Log("Request completed. Status: " + www.result);

    if (www.result != UnityWebRequest.Result.Success)
    {
        LogError("API Fetch Error: " + www.error);
        LogError("Response: " + www.downloadHandler.text);
    }
    else
    {
        string json = www.downloadHandler.text;
        Log("Raw JSON Received: " + json);

        try
        {
            TelemetryData[] telemetryArray =
                JsonHelper.FromJson<TelemetryData>(json);

            if (telemetryArray.Length >= 2)
            {
                var telemetry1 = telemetryArray[0];
                var telemetry2 = telemetryArray[1];

                II_telemetry_Text.text = $"Timestamp:
{telemetry1.timestamp}\nTemperature: {telemetry1.temperature}\nHumidity:
{telemetry1.humidity}";

                LSU_telemetry_Text.text = $"Timestamp:
{telemetry2.timestamp}\nTemperature: {telemetry2.temperature}\nHumidity:

```

```

{telemetry2.humidity}";

        Log("Telemetry updated successfully in two UI fields.");
    }
    else
    {
        LogError("Received less than 2 telemetry records.");
    }
}
catch (System.Exception ex)
{
    LogError("JSON Parsing Error: " +
ex.Message); LogError("Failed JSON: "
+ json);
}
}
}

void Log(string message)
{
    Debug.Log(message);
    if (DebugConsole != null)
    {
        DebugConsole.text = message + "\n" +
DebugConsole.text; TrimConsole();
    }
}

void LogError(string message)
{
    Debug
    .LogE
    rror(
    messa
    ge);
    if
    (Debu
    gCons
    ole
    !=
    null)
    {
        DebugConsole.text = "<color=red>" + message +

```