

Xander Gilbert Vanparys

Multimodal Large Language Models and Computer Vision for Open Discovery in Educational Augmented Reality

Master's thesis in Computer Science

Supervisor: Gabriel Hanssen Kiss

Co-supervisor: Pieter Simoens

June 2025



NTNU

Norwegian University of
Science and Technology

Xander Gilbert Vanparys

Multimodal Large Language Models and Computer Vision for Open Discovery in Educational Augmented Reality

Master's thesis in Computer Science
Supervisor: Gabriel Hanssen Kiss
Co-supervisor: Pieter Simoens
June 2025

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Abstract

This thesis explores how Augmented Reality (AR) systems can support users in performing physical tasks by integrating instruction tracking with object localization using AI models. A modular prototype was developed that combines a Multimodal Large Language Model (GPT-4o) with a real-time object-detection model (YOLOv8) and Unity’s ARKit spatial mapping. The system guides users through step-based procedures by evaluating images to assess task progress and placing 3D visual cues over relevant objects in the real world. GPT-4o not only confirms or corrects user actions based on step-specific prompts and user images, but also translates natural-language object names into detector labels. Object positions are determined by raycasting YOLO-predicted 2D centroids into a LiDAR mesh to anchor arrows in 3D space.

Seventeen participants (including two domain experts) completed two procedural tasks—a VR treadmill setup and a gastroscopy trainer preparation—while thinking aloud. Data was gathered through system logs, observations, questionnaires, and interviews.

Results showed a 100% task completion rate with minimal external help. Participants rated the system highly for helpfulness ($M = 4.35 / 5$), and 70% found the object cues accurate and trustworthy. Qualitative feedback emphasized how real-time confirmations, red-arrow cues, and step-specific guidance reduced uncertainty and created a strong sense of progress. Key limitations included occasional object misdetections due to small training sets, vague instruction interpretations by GPT-4o, and LiDAR edge noise in peripheral views.

These results indicate that integrating MLLMs with vision-based localization holds promise for enabling lightweight, interactive AR task support on consumer-grade devices. Future work should expand to longer, more complex procedures and explore hands-free deployment on headsets like the Apple Vision Pro.

Summary

Augmented Reality (AR) is increasingly used to support users in performing complex physical tasks, such as equipment setup or procedural training. Traditional AR guidance systems often rely on static markers or pre-authored animations and struggle to adapt easily to different tasks, requiring more expert time. This thesis investigates whether combining Multimodal Large Language Models (MLLMs) with real-time vision-based object localization can enable a more flexible and interactive form of AR task guidance.

To explore this, a modular prototype system was developed that integrates GPT-4o for instruction tracking and YOLOv8n for object detection, built on top of Unity’s ARKit framework for spatial mapping. The system evaluates images captured by the user during task execution and determines whether the current step has been completed correctly. If so, the user receives confirmation; otherwise, the system provides corrective feedback. In parallel, it localizes task-relevant objects in the environment and overlays red 3D arrows on them using a raycasting technique that projects 2D bounding box centroids into a LiDAR-based spatial mesh. GPT-4o also translates natural-language object names from the task description into corresponding YOLO class labels, allowing the system to dynamically adapt to different tasks.

Development followed a sprint-based iterative approach, guided by weekly feedback meetings with the MirageXR team. Early sprints focused on prototyping isolated features, testing them under real-world conditions, and identifying key bottlenecks such as GPT-4o inference latency. Exploratory evaluations during development provided valuable insights into prompt clarity, UI layout, and system responsiveness.

The final prototype was tested with 17 participants (including two domain experts) across two procedural setups: assembling a VR treadmill and preparing a gastroscopy trainer. Participants completed the tasks while thinking aloud, and their behavior was recorded through system logs, structured observations, questionnaires, and interviews. All participants completed their tasks without external assistance. The system received a high helpfulness rating ($M = 4.35 / 5$), and object localization was consistently praised for its usefulness—100% of participants rated it positively, though around 30% gave neutral scores for accuracy. Users also agreed that the system was responsive enough (76.4% positive), and 94% agreed or strongly agreed that the system made the procedure easier.

Qualitative feedback highlighted strengths such as object localization reducing uncertainty, easy-to-follow step-by-step instructions, and the system’s simplicity: “just two buttons during the procedure, that is good.” However, minor issues were noted, including occasional confusion about step progression, interface text readability, and system usage. Suggestions for improvement included clearer task phrasing, improved UI layout, and enhanced user guidance. One participant remarked, “I like the idea of using AR to complete procedures — it seems more helpful than reading an instruction manual.”

In conclusion, this thesis demonstrates that combining MLLMs and vision-based models can potentially enable lightweight, modular AR systems capable of delivering real-time feedback and spatial cues on consumer-grade devices. The prototype suggests a promising direction for developing more adaptable and intelligent AR guidance solutions. Future work should investigate support for longer and more complex procedures, continuous passive object scanning, and hands-free operation on devices such as the Apple Vision Pro.

Preface

This thesis was completed as the final requirement for the Master of Science in *Information Engineering Technology* at **UGent**, during a five-month Erasmus exchange at **NTNU** in Trondheim, Norway. It reflects my ongoing interest in how emerging technologies like *AI* and *augmented reality* can improve real-world learning experiences.

The project was inspired by a simple question: what if AR manuals could respond intelligently to user mistakes and progress, rather than just showing static overlays? Exploring this idea led to the integration of multimodal large language models with real-time object localization, applied within the context of MirageXR. The result is a prototype that investigates how AI can deliver adaptive, spatially-aware guidance in physical environments.

Completing this thesis under tight time constraints and in a new research setting challenged me to work independently, make practical trade-offs, and develop a system from concept to evaluation. It also gave me the opportunity to explore the intersection of *machine learning*, *human-computer interaction*, and *spatial computing*—a combination I hope to continue working with in the future.

Xander Vanparys
Trondheim, June 2025

Acknowledgements

This thesis would not have been possible without the support, guidance, and encouragement of many people, to whom I am deeply grateful.

First, I would like to thank my supervisors, **Prof. Dr. Gabriel Hanssen Kiss (NTNU)** and **Prof. Dr. Ir. Pieter Simoens (UGent)**, for their flexibility and constructive feedback throughout the project.

A special thanks goes to **Dr. Mikhail Fominykh**, whose support throughout the exchange made a significant difference, both technically and personally.

I also want to thank the entire **MirageXR team** for their openness, collaboration, and feedback during weekly meetings. Their insights helped ground the project in real-world needs and kept the scope aligned with their real-world use case.

To my thesis partner and office mate, **Jolan Vermeersch**, thank you for the countless brainstorming sessions, debugging sessions, and shared distractions — working alongside you made this experience more enjoyable and manageable.

I'm also thankful to my **family and friends** for their constant encouragement, especially during the more stressful phases of development and writing. Your support helped keep me motivated and focused.

Finally, I'd like to acknowledge the opportunity provided by the **Erasmus+ exchange program**, which allowed me to broaden my academic experience and complete this thesis in such a stimulating international environment.

Xander Vanparys

Contents

Abstract	iii
Summary	v
Preface	vii
Acknowledgements	ix
Contents	xi
Figures	xvii
Tables	xxi
Code Listings	xxiii
1 Introduction	1
1.1 Research Questions	2
1.2 Thesis Structure	2
2 Background	5
2.1 Augmented Reality and Learning	5
2.2 Object Detection in Vision-Based AI	6
2.3 Multimodal Large Language Models	6
2.3.1 Prompt Engineering	6
2.4 Procedural Training and Instruction Tracking	7
2.5 MirageXR, Unity, and ARFoundation	8
3 Related Work	9
3.1 Augmented Reality for Procedural Training	9
3.2 MLLMs for Robotic Task Execution	10
3.3 MLLMs for Human-Centered Procedural Training	11
4 Requirement Analysis	13
4.1 Stakeholder Analysis	13
4.2 Requirements Justification	13
4.3 Requirements	14
4.3.1 Functional Requirements	14
4.3.2 Non-Functional Requirements	15
4.4 Design Goals	15
4.5 Conclusion	16
5 Methods	17
5.1 Development Approach	17
5.2 Iteration Cycles	17
5.3 Research Methods	18

5.3.1	Model Selection Process	18
5.3.2	Development and Testing Process	18
5.3.3	Intermediate Instruction Feedback User Evaluation	19
5.3.4	Latency and Responsiveness Evaluation	20
5.3.5	Object Localization Evaluation	20
5.4	Final User Evaluation	21
5.4.1	Overview and Purpose	21
5.4.2	Task Design	21
5.4.3	Participant Overview	22
5.4.4	Procedure	22
5.4.5	Data Sources	22
5.4.6	Custom Object Detection Model Training	23
5.5	Technological Stack	23
5.6	Ethical Considerations	24
5.6.1	Informed Consent	24
5.6.2	Ethical Oversight	24
5.6.3	Data Privacy	24
5.7	Use of AI Tools	24
6	Development	27
6.1	Sprint Overview	27
6.2	Sprint 1: Basic Object Detection Pipeline	28
6.2.1	Goals and Motivation	28
6.2.2	Exploration	28
6.2.3	Implementation	29
6.2.4	Challenges and Adjustments	32
6.2.5	Testing & Evaluation	32
6.2.6	Summary	33
6.3	Sprint 2: Instruction Tracking & Feedback	34
6.3.1	Goals and Motivation	34
6.3.2	Exploration	34
6.3.3	Implementation	35
6.3.4	Challenges and Adjustments	38
6.3.5	Controlled Evaluation of Instruction Tracking	39
6.3.6	User Workshop Evaluation	42
6.3.7	Summary	44
6.4	Sprint 3: Object Localization	45
6.4.1	Goals and Motivation	45
6.4.2	Exploration	45
6.4.3	Implementation	45
6.4.4	Adjustments and Improvements	49
6.4.5	Testing & Evaluation	58
6.4.6	Summary	59
6.5	Sprint 4: Open-Sourcing RecogNX: A Modular Unity Package for Object Localization	61

6.5.1	Goals and Motivation	61
6.5.2	Exploration and Package Design	61
6.5.3	Implementation	62
6.5.4	Testing and Validation	63
6.5.5	Challenges and Adjustments	64
6.5.6	Public API Overview	64
6.5.7	Conclusion and Next Steps	64
6.6	Sprint 5: Backend Refactor for Object-Aware Instruction Tracking	65
6.6.1	Goals and Motivation	65
6.6.2	Exploration	65
6.6.3	Implementation	66
6.6.4	Challenges and Adjustments	69
6.6.5	Testing and Evaluation	69
6.6.6	Summary	70
6.7	Sprint 6: Integration and Final System Overview	71
6.7.1	Goals and Motivation	71
6.7.2	Sprint Overview	71
6.7.3	Exploration	72
6.7.4	System Design	73
6.7.5	Component Interaction During AR Task Execution	76
6.7.6	Usage	80
6.7.7	Challenges and Adjustments	83
6.7.8	Testing & Evaluation	83
6.7.9	Summary	84
6.8	Future Sprints and Adjustments to Requirements	85
6.8.1	Sprint 7: Apple Vision Pro Version	85
6.8.2	Sprint 8: MirageXR Integration	85
6.9	Architectural Evolution and Design Rationale	87
6.9.1	Initial Prototype Approach	87
6.9.2	Shift to an Open-Source Package	87
6.9.3	Final Architecture and Evaluation	87
6.9.4	Other Adaptations	87
6.9.5	Design Rationale	88
7	System Overview	91
7.1	High-Level Overview	91
7.2	Component Breakdown	92
7.2.1	Backend Server	92
7.2.2	RecognX Unity Package	93
7.2.3	Testing Frontends	93
8	Results	95
8.1	Object Detection Performance	95
8.1.1	Inference speed	95
8.1.2	Custom YOLO Models Performance	95
8.2	Instruction Tracking Evaluation During Development	96

8.2.1	Controlled Evaluation of Instruction Tracking	96
8.2.2	User Evaluation: Exploratory Evaluation of Instruction Guidance	96
8.3	OpenAI API Usage	96
8.4	Final Evaluation	97
8.4.1	Participant Overview	97
8.4.2	Questionnaire Results	98
8.4.3	Observations during user execution	103
8.4.4	Post-Experiment Interviews	107
9	Discussion	111
9.0.1	Overview and Research Goals	111
9.1	RQ1: Instruction Tracking with Multimodal Language Models	112
9.1.1	System Design and Implementation	112
9.1.2	Step Tracking Accuracy	112
9.1.3	Context Awareness and Visual Understanding	112
9.1.4	Feedback Consistency and Prompt Reliability	113
9.1.5	Feedback Quality	114
9.1.6	Responsiveness and Latency Tolerance	115
9.1.7	Usage Cost	116
9.2	RQ2: Object Localization in AR using Vision-Based Models	116
9.2.1	System Design and Implementation	116
9.2.2	User Perception and Feature Integration	116
9.2.3	Challenges and Limitations	117
9.2.4	Proposed Improvements	118
9.3	RQ3: Integration and Impact on Procedural Learning	119
9.3.1	UI Feedback and Usability Observations	121
9.3.2	Implications for Future AR Training Tools	121
9.4	Limitations	122
9.4.1	Project Timeline and Scope	122
9.4.2	Final User Evaluation	123
9.5	Contribution to Sustainable Development Goals	124
9.6	Data Privacy and Security	125
10	Conclusion	127
10.1	RQ1: MLLMs for Task Guidance	127
10.2	RQ2: Vision-based AI for AR Feedback	128
10.3	RQ3: User Performance and Training Value	129
10.4	Main Question	130
10.5	Future Work	130
	Bibliography	135
A	Extended YOLO Performance Results	139
B	User evaluation: Workshop	141
B.1	User Workshop Notes: 2 April 2025	141
B.1.1	General Impressions and Task Successes	141
B.1.2	Observations and Edge Cases	141

B.1.3	Feature Suggestions	142
B.1.4	Limitations and Concerns	142
B.1.5	Potential Application Domains (As Identified by Participants)	142
B.1.6	What Users Liked Most	143
B.2	Representative Workshop Logs	143
B.2.1	Scenario: Prepare to Go to Lunch (Lab to Canteen)	143
B.2.2	Scenario: Put an Earphone in Your Ear	144
B.2.3	Scenario: Set Up My VR Work Station	145
B.2.4	Scenario: Tie Your Shoe	146
B.2.5	Scenario: Write “Hello” on Paper	148
C	Unity Debugging Extras	149
C.1	Unity Hierarchy and Canvas Debugging	149
C.1.1	Unity Hierarchy Screenshots	150
C.1.2	Unity Canvas Resolution Debugging	151
C.2	Additional Debugging Visualizations	151
D	GPT Prompt for Object Extraction	155
D.0.1	System Message	155
D.0.2	User Message Example	155
E	Final Evaluation Tasks	157
E.1	VR Treadmill Setup	157
E.2	Gastroscopy Setup	158
F	Final Evaluation: Questions	161
F.1	Participant Demographics and Questionnaire	161

Figures

6.1	Performance comparison of YOLO models 5, 8, 9, 10, and 11 on the COCO dataset showing mAP vs. speed trade-offs. YOLOv8 achieves competitive accuracy while maintaining real-time inference capabilities [30].	29
6.2	Next.js frontend capturing webcam images, sending them to the backend, and rendering bounding boxes.	31
6.3	Responsive design: The same frontend deployed on a mobile device, showing real-time detection results from a custom YOLO model. . .	31
6.4	Use of the frontend for testing a custom trained YOLO model on VR equipment.	32
6.5	Sequence diagram of Task Creation Process	36
6.6	Sequence diagram of Instruction Submission	37
6.7	Sequence diagram of Live Instruction Tracking	38
6.8	Inconsistency across localization attempts on the same object under slightly different conditions	50
6.9	Unity Inspector toggles for enabling and comparing different object localization methods during debugging.	51
6.10	Debugging bounding box alignment across RGB and depth modalities.	51
6.11	Visualization of debug rays originating from the centers of YOLO-projected bounding boxes in screen space, illustrating direction and intersection with the 3D scene. The blue dots are supposed bounding box centers.	52
6.12	Rendered YOLO bounding boxes in Unity's screen space, highlighting coordinate transformation misalignments between 2D detections and 3D projections.	52
6.13	Initial Unity hierarchy with multiple overlapping UI canvases and redundant subcomponents.	53
6.14	Decluttered Unity hierarchy with a single UI canvas and no redundant subcomponents.	54
6.15	Example of a label incorrectly projected into the scene, highlighting a misalignment between the label position and the corresponding object.	54

6.16	Sequence diagram: Object Localization Flow from Unity to backend and back	60
6.17	Directory structure of the RecognX Unity package.	61
6.18	Class diagram of the core <code>DiscoveryManager</code> and the <code>LocalizedObject</code> class.	62
6.19	Runtime execution flow of the <code>DiscoveryManager</code>	62
6.20	Screenshot of label prefab instantiated at detected object location. .	63
6.21	Class diagram: Task has many Steps, and each Step references one or more Objects, which are also stored globally on the task level. .	66
6.22	Old Approach: Users had to locate all relevant objects before beginning the task.	72
6.23	New Approach: Users can locate required objects step-by-step during execution.	72
6.24	UML Class Diagram of the <code>ARInstructionManager</code> package. Some implementation details (such as certain private fields, internal utilities, and lower-level helper classes) have been omitted for clarity and readability.	74
6.25	State Machine Diagram: Overview of the <code>TaskState</code> transitions showing the AR instruction session flow.	75
6.26	Sequence diagram illustrating interactions during the app startup. .	76
6.27	Sequence diagram illustrating interactions during the task selection phase.	77
6.28	Sequence diagram illustrating interactions during task initiation, showing how step description and relevant objects are presented to the user.	77
6.29	Sequence diagram for dynamic object scanning during step execution.	78
6.30	Sequence diagram for step completion and progression.	79
6.31	QR code linking to a video walkthrough demonstrating the final Unity frontend app using the developed package.	84
7.1	High-level system architecture illustrating the modular design and data flow between the backend service, the Unity package, and optional integration with the <code>MirageXR</code> platform.	91
8.1	Breakdown of OpenAI API usage costs for input and output tokens.	96
8.2	Participant demographics: gender (left) and age bracket (right). . .	97
8.3	Distribution of participants across the two evaluation tasks (IMTEL vs. SINTEF).	97
8.4	Normalized distribution of prior AR experience within each task group.	98
8.5	Participant responses to the statement: “The system was helpful in completing the procedure.” (1 = Strongly Disagree, 5 = Strongly Agree).	99
8.6	Participant perceptions of the object localization feature.	99

8.7	Participant ratings of the system's instruction clarity and response timeliness.	100
8.8	Participant responses regarding confusion and perceived helpfulness of the system.	101
9.1	Attempt to fool GPT by placing the VR headset incorrectly on the back of the head	113
C.1	AR Session object hierarchy with redundant subcomponents.	150
C.2	Main Camera hierarchy view showing overlapping components.	150
C.3	ObjectLocatorManager component contributing to hierarchy clutter.	150
C.4	XR Origin configuration with redundant subcomponents.	150
C.5	Root canvas structure with incorrect portrait resolution.	151
C.6	UI canvas with inconsistent resolution.	151
C.7	Corrected canvas configuration matching iPhone screen resolution and landscape orientation.	151
C.8	Bounding-box debugging visualization example.	152
C.9	Bounding-box debugging visualization example.	152
C.10	Bounding-box debugging visualization example.	152
C.11	Bounding-box debugging visualization example.	153
C.12	Bounding-box debugging visualization example.	153
C.13	Bounding-box debugging visualization example.	153

Tables

6.1	Representative YOLOv8n inference times. Full table available in Appendix A.	33
6.2	Effect of Instruction Complexity on Latency (Low Resolution)	41
6.3	Effect of Image Resolution on Latency (Same Instruction Set)	42
6.5	Public members of the <code>DiscoveryManager</code> class.	64
6.6	Public properties of <code>ARInstructionManager</code>	80
6.7	Public methods of <code>ARInstructionManager</code>	80
6.8	Public events of <code>ARInstructionManager</code>	81
6.9	Updated Functional Requirements for the Current Prototype	86
6.10	Non-Functional Requirements Deferred to Future Development . . .	86
8.1	Performance of task-specific YOLOv8n models on available training and validation data.	95
A.1	Full results of YOLOv8n inference benchmark on 29 test images. . .	140
E.1	Objects used in the VR Treadmill task.	158
E.2	Objects used in the Gastrosocopy task.	159

Code Listings

6.1	Sample API response	30
6.2	Manual raycast through snapshot mesh using triangle iteration . . .	55
6.3	Corrected viewport normalization for raycasting	56
6.4	Incorrect pose-based ray projection (before correction)	57
6.5	Reprojecting rays using stored pose	57
6.6	Example integration script using the RecognX package.	63
6.7	Example GPT Object Extraction Output	67
6.8	Example of disambiguating identical YOLO classes using distinct step_names.	69
6.9	Simplified example of integrating the ARInstructionManager in a Unity application, demonstrating event subscriptions and essential method calls.	82

Chapter 1

Introduction

Augmented Reality (AR) is increasingly being recognized as a promising medium for learning and procedural task assistance. In particular, head-mounted and mobile AR systems offer new possibilities for supporting users in real-world environments, where traditional instruction methods fall short. In recent years, the convergence of vision-based artificial intelligence (AI) and powerful language models has introduced the potential for intelligent AR systems capable of not only displaying step-by-step instructions but also dynamically interpreting user behavior and providing real-time guidance. This thesis explores how these developments can be combined to enhance the effectiveness of procedural learning experiences in AR.

The focus of this work lies in the design and evaluation of an AR system that integrates two key capabilities: instruction tracking using Multimodal Large Language Models (MLLMs) and object localization using computer vision. Rather than approaching AR as a passive display layer, this thesis investigates how AI-driven reasoning and perception can be used to create an interactive assistant that actively supports users in completing physical tasks. The system is implemented as a mobile AR prototype and evaluated in the context of educational and technical training procedures.

This research is situated at the intersection of human-computer interaction, computer vision, and AI-powered education. While prior work has explored AR for step-based guidance or computer vision for object detection, few systems have combined these capabilities in a cohesive framework driven by natural language understanding and visual input. The thesis builds on recent advances in multimodal reasoning (e.g., GPT-4o), object detection (e.g., YOLO), and spatial computing with LiDAR-based AR hardware, contributing a practical demonstration of how these technologies can work together to support real-world procedural learning.

1.1 Research Questions

The research is guided by the following main research question:

How can an AR system be designed to effectively combine instruction tracking using Multimodal Large Language Models and object localization, leveraging vision-based AI models, to support users in performing physical tasks?

This question is explored through three subquestions:

1. *How can MLLMs be leveraged to track and guide users through multi-step tasks using visual input and structured prompts?*
2. *How can a robust object localization pipeline be designed and implemented in AR systems using vision-based AI models and AR devices to provide visual feedback during task execution?*
3. *To what extent does the integration of instruction tracking and object localization improve user performance and training outcomes in AR-based procedural learning?*

To answer these questions, a prototype AR system was developed that uses GPT-4o to track task progress from images and text prompts, a YOLOv8-based backend for object detection, and Unity ARKit for spatial visualization and user interaction. The system was tested with 17 participants across two procedural tasks in an educational setting.

1.2 Thesis Structure

To address the research questions, the remainder of this thesis is structured as follows:

- **Chapter 2 – Background** provides foundational context on AR, object detection, and language models, establishing the technological basis for this research.
- **Chapter 3 – Related Work** reviews prior research on AR training systems and multimodal instruction guidance.
- **Chapter 4 – Requirement Analysis** defines the functional and non-functional requirements of the system, derived from literature, stakeholder input, and research goals.
- **Chapter 5 – Methods** outlines the technologies and evaluation methods used throughout the project, including the development approach and evaluation setup.
- **Chapter 6 – Development** presents the implementation process through iterative sprints, detailing how each component was developed, tested, and refined.
- **Chapter 7 – System Overview** summarizes the final integrated prototype, describing the system architecture.

- **Chapter 8 – Results** presents the findings across development of the system and from the final user evaluation, including both quantitative results (e.g., user questionnaire data) and qualitative insights from user behavior and feedback.
- **Chapter 9 – Discussion** reflects on the research questions, interpreting the results in context, discussing limitations, and situating the findings within the broader literature.
- **Chapter 10 – Conclusion** synthesizes the thesis findings, answers the research questions, and proposes directions for future work.

Chapter 2

Background

This chapter provides an overview of the key technologies and concepts underpinning this research project.

2.1 Augmented Reality and Learning

AR refers to technologies that overlay digital information onto the physical world, creating a composite view that enhances the user’s perception and interaction with their environment [1]. AR has seen increasing adoption across various domains, including healthcare, industrial maintenance, and education, due to its ability to provide intuitive, situated, and often hands-free access to information [2].

In educational contexts, AR is often praised for its potential to support situated learning, where learners engage with content directly within the context in which it is applied [3]. It can make abstract concepts more tangible, offer real-time feedback, and foster learner engagement by making tasks more interactive and visually guided [4].

However, realizing this potential in practice presents several challenges. AR learning systems often depend on manual content creation, requiring designers or experts to anticipate user needs and encode all feedback explicitly [5]. Moreover, the limited flexibility of traditional AR guidance—typically based on static annotations and QR codes—can make such systems brittle or overly prescriptive in real-world scenarios [3].

These limitations have led researchers to explore ways to make AR systems more adaptive and intelligent. One promising direction is the integration of AI techniques, especially those involving multimodal input and reasoning, which can allow AR systems to better understand user actions, the task context, and provide context-aware guidance dynamically [6]. This thesis builds on this line of research, aiming to combine the strengths of AR and recent advances in AI to support procedural learning in a more flexible and scalable way.

2.2 Object Detection in Vision-Based AI

Object detection is a foundational task in computer vision that involves identifying instances of predefined object classes and locating them within an image, typically by predicting bounding boxes and class labels. It differs from classification by requiring spatial localization in addition to recognition.

Modern object detectors are typically categorized into two types: two-stage detectors, like Faster R-CNN [7], which first generate region proposals and then classify them, and single-stage detectors, like YOLO (You Only Look Once) [8], which perform detection in one pass for higher speed.

YOLO and its successors (e.g., YOLOv5, YOLOv8...) have become widely used due to their ability to balance real-time performance and accuracy. These models use fully convolutional networks to simultaneously predict bounding boxes and class probabilities across a grid of the input image.

2.3 Multimodal Large Language Models

MLLMs are a recent development in artificial intelligence that enable systems to process and reason over multiple modalities, most commonly visual and textual data. These models combine a vision encoder—often a convolutional neural network or vision transformer—with a large language model (LLM), creating a unified architecture capable of interpreting images in conjunction with natural language prompts [9].

MLLMs are trained on large datasets containing paired image-text data, allowing them to develop a shared representation space across modalities. As a result, they can perform a wide variety of tasks such as visual question answering, caption generation, document understanding, and instruction following. A prominent example is GPT-4o x[10], which accepts both images and text as input and produces text output, enabling reasoning grounded in visual information.

One of the key advantages of MLLMs is their flexibility across tasks without requiring task-specific retraining. Their generalization capabilities allow them to operate in diverse domains, making them well-suited for applications in accessibility, education, robotics, and user assistance. Their ability to interpret complex visual scenes, understand structured prompts, and generate context-aware responses positions them as a core component in next-generation AI systems.

In the context of this thesis, MLLMs are particularly relevant for their potential to enable intelligent feedback and dynamic instruction tracking in AR learning environments. Their ability to interpret images and structured prompts opens new possibilities for procedural guidance that adapts to real-time user actions.

2.3.1 Prompt Engineering

Prompt engineering is a key technique for guiding the behavior of LLMs and MLLMs without modifying their underlying parameters.

Concept and Definition

Prompt engineering involves designing prompts that guide a model's output toward desired behaviors in tasks such as reasoning, question answering, or classification. According to Sahoo et al. [11], prompt engineering leverages the pre-trained capabilities of LLMs by supplying carefully designed textual or visual inputs that elicit specific model outputs. This approach is particularly valuable when retraining or fine-tuning the model is impractical or computationally expensive.

Recent Advancements

Recent years have seen significant progress in prompt engineering, resulting in techniques that broaden its applicability and improve model performance:

- **Chain-of-Thought (CoT) Prompting:** Introduced by Wei et al. [12], this technique improves a model's reasoning abilities by guiding it through intermediate reasoning steps rather than direct answers, leading to greater transparency and accuracy in multi-step tasks.
- **Zero-shot prompting** — Asking the model to complete a task without any examples [13].

Translate the following English sentence to French: "Where is the train station?"

- **Few-shot prompting** — Providing a few examples of input–output pairs before the actual query to demonstrate the desired format or behavior [13].

Translate:

English: "Good morning" → French: "Bonjour"

English: "Thank you" → French: "Merci"

English: "Where is the train station?" →

- **Self-Consistency and Auto-CoT:** Explored by Wang et al. [14] and related works, these techniques involve aggregating multiple reasoning paths or automatically generating prompts, which can improve the robustness and reliability of model outputs by reducing hallucinations and increasing output stability.

Prompt engineering has become a vital methodology for leveraging the capabilities of LLMs and MLLMs. The conceptual foundation provided here supports the design and implementation of instruction tracking and feedback mechanisms explored in this thesis.

2.4 Procedural Training and Instruction Tracking

Procedural tasks involve a sequence of actions performed in a specific order to complete a goal—ranging from assembling machinery to sterilizing lab equipment.

Step-by-step guidance or structured cueing during procedural tasks can reduce errors and improve learning outcomes by directing attention to relevant elements. This is supported by studies showing that learners reflect more effectively on their actions when given visual cues — a method known as cued retrospective reporting [15]. Traditional methods like manuals or videos provide static support but lack the adaptability to respond to user mistakes, incomplete actions, or the actual context in which the task is performed.

AI-driven instruction systems, particularly those using MLLMs, could offer dynamic, context-aware guidance. These systems could interpret natural-language instructions, infer task structure, and adjust feedback based on visual input—making them well-suited for real-world procedural support.

2.5 MirageXR, Unity, and ARFoundation

MirageXR is an open-source AR platform designed for workplace and educational training. It's built on Unity and supports the creation, delivery, and tracking of guided tasks in AR. It also provides infrastructure for collaboration, analytics, and the integration of third-party services [16].

Unity is a widely adopted game engine and development platform for real-time 3D content, including AR applications [17]. ARFoundation, Unity's cross-platform AR framework, abstracts away platform-specific differences by providing a unified API for ARKit (iOS) and ARCore (Android) [18]. It supports essential features such as surface detection, camera tracking, light estimation, and depth sensing.

The system developed in this thesis is designed for integration into the MirageXR platform, leveraging ARFoundation to access sensor data (e.g., depth, LiDAR), manage spatial anchors, and overlay digital content in the user's physical environment.

Chapter 3

Related Work

3.1 Augmented Reality for Procedural Training

AR has shown increasing promise for procedural training in industrial and educational contexts. Across studies, AR systems consistently reduce cognitive load and improve performance by delivering in-situ, spatially anchored guidance.

A major trend is the use of head-mounted displays (HMDs) to provide hands-free, context-aware instructions. Evans et al.[19] built an AR assembly assistant for the HoloLens that projects dynamic instructions into the user’s environment. While lacking formal evaluation, internal feedback suggested improved usability over handheld devices. Similarly, García et al.[20] presented *Holorailway*, a HoloLens-based system for railway assembly that uses QR code detection for spatial alignment. Their study reported a 47% reduction in assembly time and an 88% drop in errors, underscoring the impact of immersive AR tools in real-world settings.

Other work highlights the benefits of combining AR with sensor data. Richardson et al. [21] fused self-reports with motion and interaction metrics to evaluate AR, tablet, and desktop training. AR users made fewer errors, worked faster, and reported higher satisfaction—an effect linked to AR’s contextual feedback. The study supports the value of multimodal data in evaluating training systems.

Limbu et al. [22] approached procedural training from the expert side. Their WEKIT system enables domain experts to annotate their environment with 3D models, videos, and spatial markers, which are later replayed for learners through a HoloLens interface. In a comparative study, they found no statistically significant difference in knowledge retention between learners trained using AR and those using traditional paper-based materials—indicating that the AR-based approach was at least as effective. Additionally, participants responded positively to the AR experience, and expert reviewers highlighted the clarity and relevance of the captured content, suggesting that well-designed AR systems can serve as effective tools for preserving and transferring expert knowledge.

More recently, XaiR [6] introduced a cognitive assistant that integrates MLLMs (e.g., GPT-4V, Ferret) with AR/XR headsets to guide users step-by-step through

physical tasks. By combining live camera input with 3D spatial mapping, the system detects objects, interprets instructions, and renders overlays in near real-time. In tasks like coffee making and multimeter use, XaiR achieved over 90% accuracy in step tracking and 85% in AR content anchoring, demonstrating the feasibility of dynamic, model-driven guidance without manual annotations or static setups.

Together, these studies establish AR as a viable medium for procedural learning. While earlier systems rely on predefined environments or markers, newer approaches like XaiR demonstrate the potential of combining MLLMs with real-time perception to support more adaptive instruction. Building on this trajectory, the system in this thesis integrates instruction tracking via MLLMs with real-time object localization, aiming to deliver step-by-step guidance grounded in the user's actual environment—without relying on pre-marked objects or fixed setups.

3.2 MLLMs for Robotic Task Execution

Recent advances in MLLMs such as GPT-4V have introduced a new paradigm for robotic task execution, allowing systems to interpret visual input, reason about goals, and generate actionable commands. Across the literature, a key trend is the use of MLLMs to enable zero-shot or few-shot generalization—reducing reliance on manual programming or domain-specific retraining. Similar architectures and capabilities underpin AR-based task guidance for humans.

Several works have demonstrated how vision-language models can translate perceptual inputs into manipulation policies. Shridhar et al.[23] proposed CLIPort, which combines CLIP-based visual features with a Transformer policy to generate object manipulation commands from natural language. While focused on robotic arms, CLIPort exemplifies a broader pattern: using pretrained vision-language representations to support generalized procedural behavior. Expanding this idea, Gao et al.[24] introduced PaLM-E, an embodied agent that integrates language models with sensor data and real-time visual input. It demonstrated effective performance in both simulated and physical environments, signaling that LLMs can anchor high-level reasoning in low-level robotic control.

More recent systems have pushed toward responsiveness in dynamic scenes. Mon-Williams et al.[25] developed ELLMER, which couples GPT-4 with real-time vision and force feedback to guide robots through long-horizon physical tasks. Unlike earlier systems that rely on static inputs, ELLMER reacts adaptively to environmental changes, such as altered object positions or force constraints—pointing toward greater flexibility in unstructured settings. Similarly, Liang et al.[26] introduced TaskMatrix.AI, a generalized framework that connects MLLMs to external APIs for perception, planning, and execution. Though primarily evaluated in virtual settings, it highlights the growing ambition to position LLMs as central orchestrators of real-time task execution.

3.3 MLLMs for Human-Centered Procedural Training

Parallel to robotics, MLLMs are increasingly used to support human learners in physical task settings. In these systems, the models act as instructional agents—providing context-aware guidance, interpreting user behavior, and generating adaptive feedback.

A notable example is TAGGAR by Stover and Bowman [27], which uses vision-language models to interpret natural language tasks and overlay real-time guidance in augmented reality. The system dynamically adjusts to user progress, enabling general-purpose support in physical environments. Similarly, Jiang et al. [28] built a remote instruction platform for massage therapy that fuses video, motion sensors, and pressure data into GPT-4-generated feedback. Their case study showed that users found the guidance practical and flexible, even for a tactile, embodied skill.

Another promising direction is procedural information extraction from visual input. Hussain [29] trained a domain-specific MLLM to summarize cooking and medical tutorials by identifying step boundaries and generating concise instructional text. Although applied to passive video rather than live environments, this work highlights the ability of MLLMs to understand procedural structure and translate it into usable instructions.

Action verification and progress tracking are also emerging themes. XaiR [6] exemplifies this trend, using GPT-4V and Ferret to interpret user input and track procedural progress through real-time task guidance. Beyond spatial anchoring, the system leverages MLLMs for instruction understanding and dynamic step selection in real-world tasks.

Apart from the last example, many current systems still treat instruction generation, object recognition, and spatial feedback as separate modules. Their reliance on structured inputs or static demonstrations limits their ability to adapt to real-world, unpredictable scenarios. Large-scale user evaluations also remain rare, and hardware requirements often restrict broader adoption.

Summary

This chapter reviewed how AR and MLLMs support procedural training. AR systems, especially those using headsets like the HoloLens, improve performance by providing spatial, hands-free guidance. Recent tools like XaiR integrate MLLMs to enable dynamic, real-time instruction without static setups.

In robotics, MLLMs help interpret visual input and generate actions, enabling flexible, zero- or few-shot task execution. Similar approaches now assist human users by tracking progress and providing adaptive feedback through AR.

This thesis aims to extend these trends by combining MLLM-based instruction tracking with real-time object localization and spatial anchoring, offering flexible, markerless support for real-world tasks.

Chapter 4

Requirement Analysis

This chapter defines the requirements for developing an AI-powered interactive assistant that provides real-time task guidance. The system combines computer vision, MLLMs, and AR overlays to support users in performing physical tasks.

4.1 Stakeholder Analysis

This section identifies the key stakeholders involved in the development, deployment, and eventual use of the AR instruction tracking and object localization system. These stakeholders shaped key requirements and design decisions throughout the development of this master's project.

- **System Developer:** Responsible for designing, implementing, and maintaining the system developed in this master's project.
- **MirageXR Team:** Engaged through regular meetings to review progress, provide feedback, and align development with platform requirements. Their input shaped key design decisions to ensure seamless integration with their product.
- **End Users (Trainees):** Individuals engaged in hands-on procedural training who will use the AR system for real-time guidance and feedback, relying on accurate object localization and intuitive instruction tracking.
- **Training Providers and Platforms:** Organizations and platform developers aiming to enhance procedural training through integrated AR systems.

4.2 Requirements Justification

The requirements for this project were informed by a literature analysis and recent technological developments in the fields of computer vision, MLLMs, and AR-enhanced procedural training:

- **AI-driven instruction tracking:** Recent systems demonstrate the potential of MLLMs, such as GPT-4V, to interpret natural language instructions and

visual input for procedural task support [6, 27].

- **Computer vision for object localization:** Recent object detection models have significantly improved in accuracy and inference speed, potentially supporting real-time AR overlays that help users finding relevant objects during training tasks [30].
- **Procedural training challenges:** Traditional training often requires significant expert time and effort. Recent systems like WEKIT [22] and XaiR [6] demonstrate that AI-enhanced AR can alleviate these burdens by capturing expert knowledge and delivering just-in-time, adaptive guidance to trainees.
- **XR and AR integration:** Studies show that AR systems which overlay instructions and cues directly into the real environment reduce user error and improve task performance [20, 21]. In some cases, they also report increased user satisfaction.

4.3 Requirements

4.3.1 Functional Requirements

Computer Vision for Object Localization & 3D Mapping

- **FR-1:** The system must detect objects relevant to the current task using a computer vision model.
- **FR-2:** The system must estimate the relative position of detected objects within the user's workspace.
- **FR-3:** The system must provide positional data for AR visualization.
- **FR-4:** The system must integrate with MirageXR's input and data flow, specifically using or expanding its APIs for real-time object detection data and object storage.

Multimodal Large Language Model for Instruction Generation

- **FR-5:** The system must analyze real-time user input (image and text) to determine task progress.
- **FR-6:** The system must generate context-aware, step-by-step feedback based on the user's environment and the task definition.
- **FR-7:** The system must validate if the user executed an instruction correctly by comparing the updated scene to the expected outcome.
- **FR-8:** The system must adapt to different task complexities, allowing for flexible levels of instruction detail.
- **FR-9:** The system must integrate with MirageXR's task management and storage system to track task progress and state consistently.
- **FR-10:** The system must support MLLM communication via MirageXR's APIs.

Augmented Reality for Guidance & Feedback

- **FR-11:** The AR interface must display step-by-step instructions visually, synchronized with MirageXR's task state.
- **FR-12:** The system must overlay AR markers to highlight relevant objects for each task step.
- **FR-13:** The system must display corrective feedback if an action is incorrect.
- **FR-14:** The AR interface must update dynamically based on the user's actions and progress.
- **FR-15:** The system must align with MirageXR's data structure to ensure that AR guidance and overlays synchronize correctly with instructional steps.

4.3.2 Non-Functional Requirements

Performance Requirements

- **NFR-1:** The system should process computer vision detections with minimal latency to ensure a smooth user experience.
- **NFR-2:** Instruction generation should be fast enough to feel responsive while maintaining accuracy.

Integration with MirageXR

- **NFR-3:** The system must be integrated into **MirageXR's architecture**, using its APIs for input handling, data storage, and AR visualization.
- **NFR-4:** Task progress and state tracking must be stored and managed within **MirageXR's database and backend services**.

Usability and User Experience

- **NFR-5:** The interface must be intuitive and require minimal training for first-time users.

Data Privacy and Security

- **NFR-6:** The system must comply with **privacy and security standards** relevant to MirageXR's deployment.

4.4 Design Goals

The system was designed with the following goals to facilitate future integration and system effectiveness:

- **Modular architecture:** Allows core functionalities (object detection, instruction tracking, AR overlays) to be developed and tested independently.

- **Compatibility with MirageXR:** Designed to integrate within MirageXR's architecture and APIs. The system's prototype-first approach facilitates both standalone testing and easy integration once core functionalities are fully implemented.
- **Real-time performance:** Supports low-latency object detection, instruction feedback, and AR updates for a smooth user experience.
- **Generalization:** Designed to support a variety of tasks and different levels of instruction complexity.

4.5 Conclusion

This requirement analysis outlines the technical and design specifications for the system, ensuring alignment with the project's goals and integration into MirageXR. The structured requirements provide a foundation for both the prototype and the full deployment within MirageXR.

Chapter 5

Methods

This chapter describes the methodology used to develop and evaluate the software system through an iterative process. It outlines the development approach, the structure of the iterative cycles, the tools and technologies used, the evaluation methods and ethical considerations.

5.1 Development Approach

The development of the software system was structured around a **sprint-based iterative approach** tailored to accommodate evolving requirements and technical challenges. This approach, inspired by Agile methodologies, was chosen for its ability to support continuous adaptation and responsiveness to feedback, both of which were essential given the exploratory nature of the project and the collaborative integration with the MirageXR team.

The development approach began with defining a high-level system overview, identifying key components such as instruction tracking and object localization. The system was then divided into smaller, manageable parts, with each component assigned to different sprints for focused development and testing.

Given the limited timeframe of five months afforded by the Erasmus exchange program, the approach prioritized rapid prototyping and experimentation. Early sprints aimed to develop core functionalities quickly, often using isolated prototypes and separate frontends to explore and validate core features before addressing the complexities of system integration. This pragmatic strategy aimed to balance the need for speed with the flexibility to adapt as new insights emerged.

5.2 Iteration Cycles

Each sprint cycle was structured to include **planning, implementation, testing, evaluation, and adaptation**. The expected duration of sprints often differed from the actual duration due to debugging challenges, evolving requirements, and the

open-ended nature of the project scope. The exact desired functionality sometimes became clearer only as more parts of the system were created, necessitating adjustments to the design and occasional revisiting of work from previous sprints.

Weekly meetings with the MirageXR team provided essential feedback on progress, challenges, and evolving requirements, ensuring that the project remained aligned with stakeholder expectations.

During the evaluation phase of each sprint, informal observations of system behavior—including usability issues, performance bottlenecks, and feature integration challenges—were made. While these observations were not systematically documented in every case, they provided valuable insights that informed design adjustments and bug fixes in subsequent sprints.

5.3 Research Methods

This section describes how the system was developed and evaluated, including model selection, testing, and user-based assessment methods.

5.3.1 Model Selection Process

Selection of Object Detection Model

To determine which object detection model to use, I combined online research with knowledge acquired during previous coursework in machine learning and computer vision. This background enabled me to assess models based on their accuracy, inference speed, and suitability for real-time performance in an AR-guided system.

The final selection process and reasoning are also documented in Sprint 1 (subsection 6.2.2).

Selection of Multimodal Language Model

To decide which MLLM to use, I combined online research with a focused literature review. I explored academic papers and existing projects with similar goals, particularly those involving instruction tracking and augmented reality. This helped me compare available models based on key criteria such as input/output capabilities, performance, and ease of integration.

Details of this comparison and the rationale behind the final decision are provided in Sprint 2 (subsection 6.3.2).

5.3.2 Development and Testing Process

Iterative Development and Debugging

During development, I used logging and my own observations to create, test, and debug features. I first implemented and evaluated core backend function-

ality through a custom web-based frontend. After, I extended testing into the AR environment by developing additional AR applications.

Throughout this process, I relied on console output to trace system behavior, inspect errors, and refine responses. Testing included different conditions such as varying scanning angles and lighting environments, in order to uncover practical limitations and edge cases. These observations directly informed improvements to prompt design, interface layout, and feedback mechanisms.

Exploratory Evaluation During Development

In parallel with development, I conducted exploratory, task-based evaluations to assess how well the system could track a user's progress through procedural tasks. I created tasks of varying lengths and complexities to simulate realistic use cases, and evaluated system behavior based on my own observations and subjective judgment.

This helped uncover both strengths and weaknesses in the system's ability to recognize task steps, provide appropriate feedback, and maintain coherence across multi-step procedures. Findings from this testing cycle directly guided subsequent iterations and refinements.

5.3.3 Intermediate Instruction Feedback User Evaluation

Participant Recruitment and Setup

To evaluate the instruction tracking functionality from a user perspective, A workshop with a convenience sample of 15 participants was organized. The session began with a short explanation and a live demonstration of the system, ensuring all participants understood how to interact with it.

Workshop Procedure

After the demonstration, participants were encouraged to explore the instruction tracking functionality on their own. They were asked to come up with a variety of procedural tasks using objects found in the lab and to submit step-by-step instructions to the system. The goal was to test the system's flexibility, accuracy, and limitations—including attempts to intentionally mislead it.

Throughout the session, participants were encouraged to verbalize their thoughts using a think-aloud protocol. This provided insight into their decision-making, expectations, and reactions during interaction with the system.

Data Collection and Analysis

The system automatically logged user interactions during the workshop, including submitted instructions, captured images, and the feedback provided for each step. Notes were taken from observed user behaviour and verbalized thoughts. At

the end of the session, I conducted a semi-structured group interview to collect additional feedback. While I used a set of guiding questions (“In what fields do you think this system could be useful?” and “Can you suggest additional features that would improve it?”) to initiate discussion, participants were also encouraged to freely share observations and suggestions. Every participant was also asked explicitly what they think about the system.

The collected data was analyzed qualitatively. Thematic analysis was performed on the user feedback, focusing on perceived strengths and weaknesses of the system. These insights were then used to guide further development, particularly for prompt refinement.

5.3.4 Latency and Responsiveness Evaluation

Bottleneck Identification

Responsiveness was a key concern during development, as user experience in AR systems heavily depends on low-latency feedback. Through observation and profiling, the primary bottleneck was identified to be the calls made to the MLLM API. These calls introduced noticeable delays between a user submitting an image and receiving feedback on the corresponding instruction step. Identifying this bottleneck early on shaped the development focus, prompting targeted testing to evaluate and mitigate latency under realistic usage conditions.

Latency Testing

To better understand and optimize response times, I conducted informal latency testing during development. I tested the system’s performance using variations in image resolution and instruction complexity, based on typical task scenarios previously used in exploratory evaluations. For each configuration, I measured response times and averaged the results to assess how different inputs affected processing speed.

This testing helped guide implementation decisions, including setting appropriate resolution thresholds. While no automated benchmarking pipeline was implemented, this pragmatic testing approach ensured that performance was sufficient for real-time feedback in common use cases.

5.3.5 Object Localization Evaluation

Informal Development Evaluation

Throughout the development process, I conducted informal evaluations of object localization accuracy using my own judgment and experience. This involved repeatedly testing the system with different objects, angles, and lighting conditions to assess whether the 3D positions produced were sufficiently accurate. These tests were integrated into the iterative development cycle and guided adjustments to the backend pipeline and projection logic.

A formal, quantitative evaluation was deemed impractical due to the complexity of aligning Unity's AR coordinate space with real-world physical measurements. Performing such measurements consistently across a wide variety of object types and environments would have required a significant amount of setup effort. Furthermore, given the system's general-purpose design and its dependency on the object detection model in use, a controlled benchmark would have involved too many variables to yield meaningful results within the time constraints of the project.

User Feedback-Based Validation

Instead of a formal benchmark, localization performance was indirectly validated during the final user evaluation phase. Participants were observed interacting with localized objects as part of their procedural tasks, and many provided qualitative feedback about the accuracy and helpfulness of the AR overlays. This feedback—combined with behavioral observations during scanning—helped confirm that the localization was generally sufficient for supporting real-world AR guidance, while also identifying cases where misalignment or occlusion reduced clarity.

5.4 Final User Evaluation

5.4.1 Overview and Purpose

The final user evaluation aimed to assess how effectively the AR-guided instruction tracking and object localization system supported users in realistic, industry-relevant scenarios. It focused on understanding user interaction with the system across the full workflow, including task comprehension, object scanning behavior and accuracy, real-time feedback, and overall task completion.

To this end, 17 participants each performed one of two predefined procedural tasks using the AR interface. No participant completed both tasks, ensuring that each individual's interaction represented their first use of the system. The scenarios were chosen to reflect real-world applications and involved multiple sequential steps requiring interaction with unfamiliar physical objects, guided by multimodal system feedback. The evaluation combined behavioral observation, system logging, a post-task questionnaire, and semi-structured interviews to gather both quantitative and qualitative insights.

5.4.2 Task Design

Two task scenarios were created:

- **VR Treadmill Setup (IMTEL Lab):** Participants were guided through the process of setting up a virtual reality treadmill station, including locating

and interacting with physical hardware such as power switches, base stations, shoes, and controllers.

- **Gastroscopy Training Setup (SINTEF Lab):** Participants followed a set of procedural steps resembling a medical preparation task, involving objects like a monitoring screen, camera, controller and a synthetic stomach.

Each task was broken down into step-by-step instructions and required users to confirm their actions through capturing images. The tasks were selected to vary in terms of familiarity, object types, and interaction complexity, allowing for comparative analysis. A complete overview of each task, including the specific steps and the associated objects, is provided in Appendix E.

5.4.3 Participant Overview

The 17 participants were recruited as a convenience sample from university and SINTEF lab environments and assigned to one of two tasks: VR Treadmill Preparation (IMTEL case) or Gastroscopy Setup (SINTEF case). Each group included one domain expert with prior experience in the respective procedure to provide expert-level feedback, while the remaining participants were novices.

5.4.4 Procedure

Participants received a brief demonstration of how to use the system, followed by an opportunity to read through the task summary. They then proceeded to perform the task using the AR system. Throughout the process, the system logged each step, including captured images, localized objects, and feedback events.

A **think-aloud protocol** was encouraged, with participants verbally expressing their thoughts, confusion, or expectations as they interacted with the system. Upon task completion, participants filled out a **questionnaire** and participated in a **semi-structured interview**. The full list of questions is included in section F.1. Notes were taken during the interview, and the data was later **thematically analyzed**.

5.4.5 Data Sources

The following data sources were collected and analyzed:

- **System Logs:** Captured interaction events, image submissions, and system-generated feedback per participant and per task step.
- **Behavioral Observations:** Recorded by an assistant during task execution, noting participant actions, confusion, and requests for clarification.
- **Post-Task Questionnaire:** Included Likert-scale and open-ended questions on instruction clarity, system feedback, ease of use, and overall satisfaction.
- **Post-Experiment Interview:** A semi-structured format with guiding questions, used to extract qualitative themes such as perceived system understanding and value for learning.

The results of this evaluation are discussed in detail in section 8.4.

5.4.6 Custom Object Detection Model Training

To enable real-time object localization during the final evaluation tasks, two custom object detection models were trained using YOLOv8n — one for the *VR Treadmill* setup and one for the *Gastroscopy* setup. For each task, annotated images of relevant objects were collected directly in the intended environment, aiming to replicate the typical user viewpoint during execution.

The VR Treadmill dataset consisted of 63 training images, which were augmented to 189 using standard techniques (e.g., rotation, scaling, brightness adjustment), and 16 validation images. Due to limited time for manual annotation, no separate test set was created; instead, the focus was placed on maximizing model performance using the available data. Qualitative evaluation was performed in the lab to confirm whether detection quality was sufficient for task support.

The Gastroscopy dataset was smaller, with 20 training images (augmented to 60) and 5 validation images. Despite the reduced size, object density was generally higher across frames, as most relevant items were consistently present within the scene, partially offsetting the limited dataset size.

Annotations were performed manually using *Roboflow Annotate*¹. The YOLOv8n architecture served as the baseline, and training involved data augmentation and hyperparameter tuning to improve generalization. The final models could easily be interchanged within the AR application to perform object localization during the task execution.

Due to the time-intensive nature of manual annotation, especially with many distinct object classes, dataset size was constrained to what could be completed within project limits.

5.5 Technological Stack

The system was implemented using a combination of Unity, FastAPI, GPT-4o, and YOLOv8n. Unity served as the platform for building the AR application, offering native support for spatial mapping and input handling. The backend was developed using FastAPI (Python) due to its high performance and my prior familiarity with it. GPT-4o was used via OpenAI's API to perform multimodal instruction tracking, while object detection relied on YOLOv8n, selected for its fast and accurate inference capabilities.

To support early validation of features, a lightweight frontend was developed using Next.js and deployed via Vercel. This interface allowed convenient testing of GPT-based instruction feedback and object detection before full AR integration. Backend endpoints were exposed using Cloudflared during development, allowing

¹<https://roboflow.com/annotate>

real-time communication between system components without full deployment. Version control was managed through GitHub.

5.6 Ethical Considerations

Ethical considerations were carefully addressed throughout the project, with particular attention to data privacy and informed participant consent during user testing. In accordance with the guidelines provided by Sikt for conducting research without processing personal data, the project ensured that no collected data could, at any point, be traced back to individual participants.²

5.6.1 Informed Consent

Participants were fully informed about the purpose of the study, the nature of the data being collected, and their right to withdraw at any time without consequence. All data collected during evaluations was stored securely and used solely for analytical purposes.

5.6.2 Ethical Oversight

While the project did not require formal approval from an ethics committee, it adhered to best practices regarding informed consent, voluntary participation, and responsible handling of participant data.

5.6.3 Data Privacy

All data was stored locally and managed in accordance with the principles of data minimization and purpose limitation. Due to the non-commercial, lab-based nature of the project and the absence of personally identifiable data, a formal GDPR assessment was not deemed necessary.

In summary, the project maintained high ethical standards by prioritizing transparency, obtaining informed consent, and ensuring responsible data management.

5.7 Use of AI Tools

This thesis made use of AI tools (specifically OpenAI's ChatGPT) for the following tasks:

- Rewording and improving clarity in technical and reflective sections
- Suggestions for phrasing and structural organization
- Assistance with LaTeX formatting and citation style corrections
- Support during thematic analysis of user evaluation data

²Sikt – Guide to Data Protection in Research

- Help creating tables, figure captions, and structured summaries

All AI-assisted outputs were reviewed and edited by the author. Final decisions regarding content, structure, and interpretation were solely the responsibility of the author.

Chapter 6

Development

This chapter outlines the iterative development process, structured around sprints focused on implementing, testing, and refining key system features. The sprint-based approach enabled rapid prototyping and adaptation to feedback and evolving requirements.

6.1 Sprint Overview

This section provides a high-level summary of each sprint. Detailed descriptions can be found in the corresponding sections.

- **Sprint 1: Basic Object Detection Pipeline** Initial setup of the object detection backend, including model selection, inference integration, and basic frontend visualization. (See section 6.2)
- **Sprint 2: Instruction Tracking & Feedback** Integration of multimodal instruction tracking and feedback logic. Focused on system prompt design and basic frontend testing. (See section 6.3)
- **Sprint 3: Object Localization** Development of the 3D localization pipeline, including ARKit depth probing, mesh raycasting, and coordinate projection for object positioning. (See section 6.4)
- **Sprint 4: Open-Sourcing RecognX** Packaging the object localization code as a standalone, open-source Unity package (RecognX), with modular components and public documentation. (See section 6.5)
- **Sprint 5: Backend Refactor for Object-Aware Instruction Tracking** Refactoring the backend to support tight coupling between detected objects and task steps, enabling object-aware guidance. (See section 6.6)
- **Sprint 6: Integration and Final System Overview** Bringing together all system components into a unified AR package. (See section 6.7)

Reflections on future sprints and architectural changes are discussed in:

- **Future Sprints and Adjustments to Requirements** (See section 6.8)
- **Summary & Architectural Evolution** (See section 6.9)

6.2 Sprint 1: Basic Object Detection Pipeline

6.2.1 Goals and Motivation

The goal of this sprint was to develop the core system pipeline for capturing real-world images and sending them to a backend service for object detection. The objective was to achieve initial object detection with bounding box annotations and visualize these results in a web frontend for testing.

This sprint directly addresses:

- **FR-1:** The system must detect and track objects relevant to the current task using a computer vision model.

It also lays the groundwork for:

- **FR-2:** Estimating the relative position of detected objects within the user's workspace.
- **FR-3:** Providing positional data for AR visualization.
- **FR-4:** Developing a modular architecture for future integration with MirageXR's APIs and data handling.

6.2.2 Exploration

The initial phase of this sprint focused on determining the most suitable object detection model to integrate into the system pipeline. The primary requirement was to recognize a variety of everyday objects for testing, while ensuring real-time performance suitable for AR environments. This requirement aligns with **FR-1** (object detection) and indirectly supports integration with future AR components.

Model Candidates and Rationale

Several object detection model families were considered:

- **Apple Ferret:** Highlighted in the Xair paper [6] for its multimodal capabilities, Ferret [31] is a generalist model that supports real-time object detection and textual prompts to refine detection targets. However, challenges in practical deployment (see Section 6.2.4) led to deferring its integration.
- **YOLO (You Only Look Once):** A well-established standard for real-time object detection, YOLO models—especially the latest versions—offer reliable accuracy and high-speed inference [8, 32].
- **Other Models:** Faster R-CNN [7] and SSD [33] were also considered but deprioritized due to slower inference or lower accuracy.

Selection of YOLOv8n

Given time constraints and the need for a reliable, real-time solution, the YOLO family was chosen for the initial object detection module. YOLOv8n (nano) was

selected for its faster training and inference speed, as well as its relatively low parameter count.

This decision was influenced by:

- **Project needs:** Maintain low latency and a responsive AR user experience (**NFR-1**).
- **Industry Adoption:** YOLOv8 has been extensively field-tested and is widely used in real-world applications, ensuring robust performance and community support even compared to newer YOLO versions.
- **Prior Experience:** Previous work with YOLOv8 in earlier projects provided familiarity with its deployment, enabling faster development and debugging.
- **Model Performance Trade-offs:** As shown in Figure 6.1, YOLOv8 achieves competitive accuracy while maintaining high real-time inference speed, making it well-suited for AR applications that demand both precision and responsiveness.
- **MirageXR team recommendations:** Begin with the standard COCO dataset and later consider fine-tuning on custom classes for specific tasks.

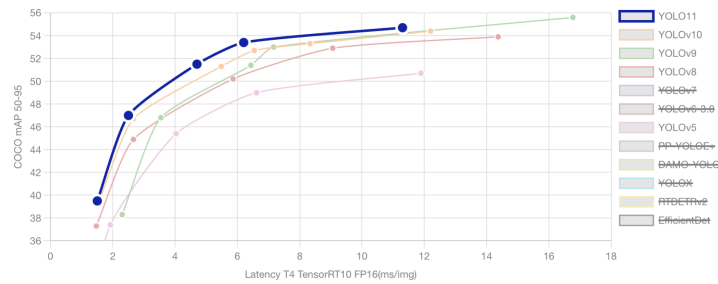


Figure 6.1: Performance comparison of YOLO models 5, 8, 9, 10, and 11 on the COCO dataset showing mAP vs. speed trade-offs. YOLOv8 achieves competitive accuracy while maintaining real-time inference capabilities [30].

Modularity and Future Integration

The system architecture was designed with modularity in mind (**FR-4**), enabling straightforward replacement or extension of the object detection module in future iterations.

6.2.3 Implementation

This section describes the development of the testing frontend, backend services, and communication pipeline for object detection. The architecture supports **FR-1** (object detection) and modularity (**FR-4**) for future AR integration.

Backend Implementation

The backend was implemented using FastAPI [34] to provide a lightweight, high-performance REST API.

Core Server Setup Defined in `main.py`, the server configures CORS support and registers:

- `/yolo/detect/` — for object detection with YOLOv8.
- `/` — a health check endpoint.

Cross-origin requests were enabled to support local frontend testing and future integrations.

YOLOv8 Integration Implemented in `models/yolo_model.py` using `ultralytics` with YOLOv8 [32]. Processing steps:

1. Decode images using `numpy` and `OpenCV`.
2. Run YOLOv8 for bounding box, confidence, and class label generation.
3. Format results as JSON compatible with the frontend.

Sample API Response See Listing 6.1 for a typical response with bounding box coordinates, classes, and confidence scores.

```
1 {  
2   "bounding_boxes": [  
3     [102.5, 50.2, 220.3, 180.4],  
4     [300.0, 75.0, 420.0, 200.0]  
5   ],  
6   "confidences": [0.85, 0.78],  
7   "classes": ["bottle", "laptop"]  
8 }
```

Code listing 6.1: Sample API response

Performance Considerations YOLOv8n was selected to meet real-time interaction needs (NFR-1).

Modularity and Scalability The backend's design (FR-4) enables easy integration of alternative detection models or additional processing steps in future iterations.

Frontend Implementation

Developed using Next.js¹ and deployed on Vercel², the frontend captures webcam frames, sends them to the backend, and displays annotated images with object la-

¹<https://nextjs.org/>

²<https://vercel.com/>

bels. It was designed for full responsiveness on desktop and mobile (Figures 6.2 and 6.3), enabling quick validation of different YOLO models and custom-trained variants. Options for capture interval and camera selection support flexible testing.

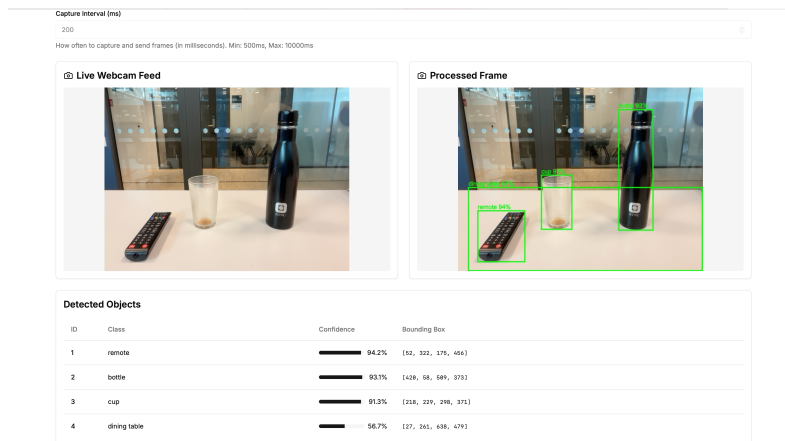


Figure 6.2: Next.js frontend capturing webcam images, sending them to the backend, and rendering bounding boxes.

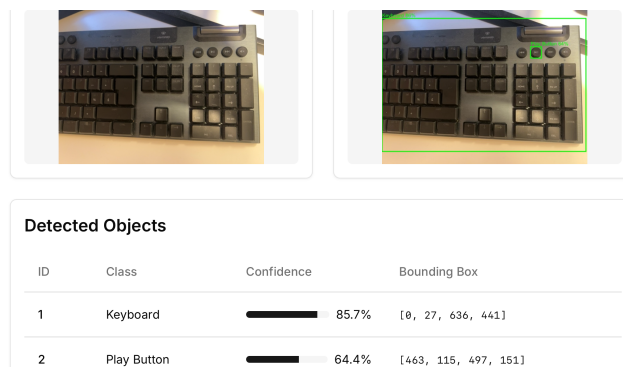


Figure 6.3: Responsive design: The same frontend deployed on a mobile device, showing real-time detection results from a custom YOLO model.



Figure 6.4: Use of the frontend for testing a custom trained YOLO model on VR equipment.

6.2.4 Challenges and Adjustments

Apple Ferret Integration

Apple Ferret initially appeared promising for multimodal detection but proved challenging to set up. Its complexity led to deferring integration, keeping YOLOv8n as the main object detection module.

Frontend Responsiveness

Originally designed for desktop, the frontend was adapted for mobile, facilitating easier testing in real-world scenarios.

6.2.5 Testing & Evaluation

Evaluation focused on qualitative observation of detection outputs and responsiveness, ensuring alignment with **FR-1** and **NFR-1**. An example is shown in Figure 6.4

Functional Testing

Using the frontend, detection results were verified for:

- Accurate bounding boxes and class mappings.
- JSON formatting suitable for AR overlays (**FR-3**).

Performance Assessment

YOLOv8n demonstrated near real-time detection performance across a variety of test images. To quantify this, we conducted a benchmark using 29 representative frames. For each frame, inference was run 10 times to reduce noise, and the

average latency and effective FPS were recorded. As shown in Table 6.1, the average inference time ranged from 48–60 ms, corresponding to approximately 17–21 FPS.

Image	Avg. Inference Time (s)	FPS
1	0.053	18.78
2	0.056	18.01
3	0.048	20.78
...	...	...

Table 6.1: Representative YOLOv8n inference times. Full table available in Appendix A.

Qualitative testing using the frontend showed that even a 200 ms interval between frames yields a responsive and smooth experience. This is more than fast enough for applications like showing the position of objects in space, validating YOLOv8n’s suitability for spatial AR guidance.

These results confirm that the system satisfies **NFR-1** (low-latency interaction) and is technically capable of supporting continuous feedback scenarios (5FPS+) in future iterations.

6.2.6 Summary

Sprint 1 delivered a robust object detection pipeline using YOLOv8n, meeting **FR-1** and **NFR-1**. The modular backend (**FR-4**) supports future AR integration (**FR-3**) and model flexibility. The responsive frontend enabled rapid testing and validation. Overall, the system is ready for AR and instruction tracking integration in the next sprints.

6.3 Sprint 2: Instruction Tracking & Feedback

6.3.1 Goals and Motivation

The goal of this sprint was to implement instruction tracking and feedback using a MLLM. The objective was twofold: first, to determine the user's current progress within a multi-step task (instruction tracking), and second, to generate context-aware, step-by-step feedback to guide the user towards successful task completion.

This sprint directly addresses:

- **FR-5:** The system must analyze real-time user input (image and text) to determine task progress.
- **FR-6:** The system must generate context-aware, step-by-step instructions based on the user's environment and the task definition.
- **FR-7:** The system must validate if the user executed an instruction correctly by comparing the updated scene to the expected outcome.

It also lays the groundwork for:

- **FR-8:** Adapting to different task complexities.
- **FR-9:** Integration with MirageXR's task management and storage system.
- **FR-10:** Supporting multimodal communication APIs for LLM integration.

6.3.2 Exploration

The initial phase of this sprint focused on identifying the most suitable approach to implement instruction tracking and feedback using a MLLM. The primary requirement was to design a system that could determine a user's progress within a multi-step task based on visual input and then provide context-aware, step-by-step guidance.

Model Selection and Rationale

OpenAI was selected as the provider for the MLLM due to its industry leadership and extensive support for multimodal (text and image) processing. The goal was to integrate a system capable of analyzing user instructions alongside real-time visual input (e.g., webcam images), essential for dynamic, step-by-step guidance in AR scenarios.

Several OpenAI models were evaluated:

- **GPT-4:** Released in 2023, GPT-4 is a highly capable language model that supports both text and image inputs through the vision-enhanced variant (GPT-4V). However, GPT-4's relatively higher latency and computational demands could impact real-time responsiveness in AR settings.
- **GPT-4 Turbo:** A more cost-effective and faster variant of GPT-4, GPT-4 Turbo also supports multimodal inputs, including images. Its improved performance on reasoning tasks and lower latency made it a strong candidate. However, according to OpenAI's documentation, GPT-4o is generally considered

to be more “intelligent” than GPT-4 and GPT-4 Turbo, especially in real-time interactions [35].

- **GPT-4o:** Released in 2024, GPT-4o represents the latest generation in OpenAI’s lineup. It features:
 - **Low Latency:** Faster response times than previous GPT-4 variants, making it highly suitable for real-time feedback systems.
 - **Integrated Vision-Text Reasoning:** Seamless multimodal capabilities essential for tasks involving both textual instructions and visual analysis.
 - **Enhanced Intelligence:** According to OpenAI’s documentation, GPT-4o is considered more intelligent than earlier GPT-4 models [35], further supporting its use in complex, real-time instruction tracking and feedback scenarios.

Given these factors, GPT-4o was selected for the MLLM in this sprint, balancing performance, intelligence, and real-time responsiveness for AR instruction tracking and feedback.

Prompt Engineering Techniques

As discussed in subsection 2.3.1, prompt engineering offers a powerful way to steer MLLMs without retraining. In this sprint, we explored task-specific prompt templates for GPT-4o. These prompts were carefully designed to provide clear structure and support consistent instruction tracking.

While advanced techniques like chain-of-thought prompting were lightly adopted—asking the model to both evaluate a step and reason about the next—others, such as self-consistency through multiple sample aggregation, were not applied due to latency constraints.

Still, this phase demonstrated that with carefully structured prompts, GPT-4o can deliver reliable, stepwise feedback based on visual context. These findings informed the full system implementation and laid the foundation for future refinements.

6.3.3 Implementation

To mimic the MirageXR architecture, the dedicated backend server from last sprint was extended. This backend will also manage task creation, storage, user sessions and all interaction with OpenAI’s GPT-4o API.

Task Creation Workflow

Task creation begins with the user saving a new task (title, step list, and optional reference images) via the frontend interface. The backend confirms receipt, stores the task. The process is illustrated in Figure 6.5, which shows the data flow

between the user, frontend, backend, and database during task creation. Each saved task is saved and made available for retrieval and review.

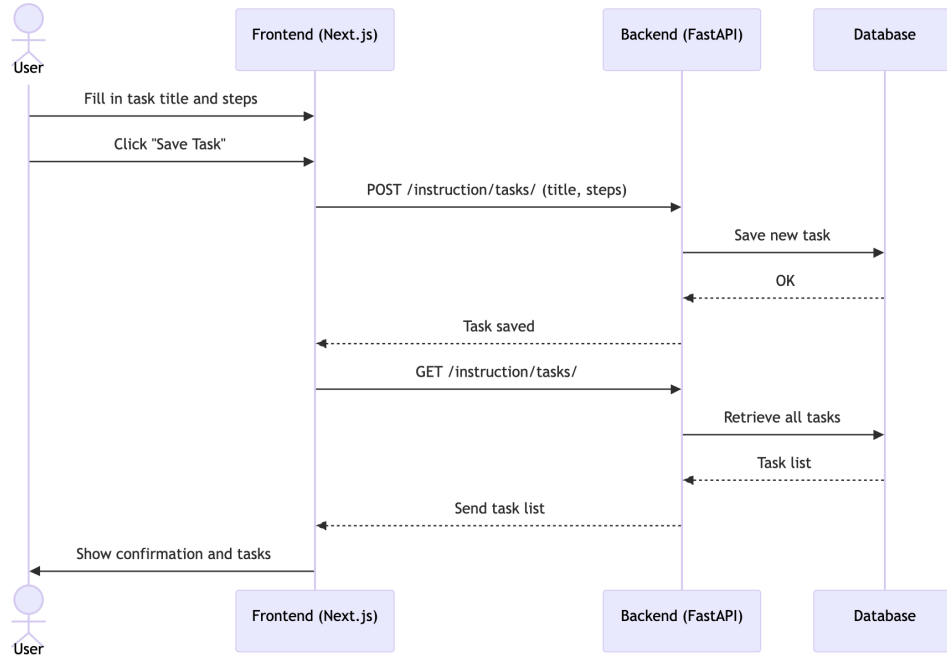


Figure 6.5: Sequence diagram of Task Creation Process

Instruction Submission

The interaction between backend and GPT-4o revolves around two flows: **instruction submission** and **live progress tracking**.

Figure 6.6 shows the sequence of operations when a user submits a task for tracking. The backend sends the full task to GPT-4o, starts a session via the SessionManager, and returns confirmation and timing data to the frontend (for performance evaluation).

System prompt:

"Be concise. Acknowledge task submission and store it for later step tracking."

Example user message:

Task: Drink water
 Steps:
 1. Take the water bottle
 2. Drink from it
 3. Put it away

Reference images:

[data:image/jpeg;base64,...]

[data:image/jpeg;base64,...]

Expected GPT-4o response:

"Understood. I've stored the steps for later tracking. Let me know when you're ready to begin."

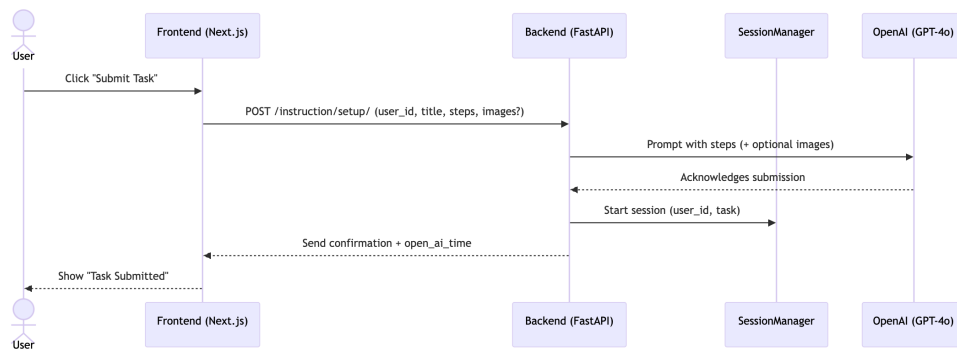


Figure 6.6: Sequence diagram of Instruction Submission

Live Instruction Tracking

For each new webcam frame, the backend sends a dynamic prompt containing the full task context, current step, and user image: As shown in Figure 6.7, each frame captured by the webcam initiates a tracking cycle. The frontend sends the image to the backend, which constructs a detailed prompt and sends it to GPT-4o.

System prompt:

"Always start response with 'Step (short description of the step) Complete: Yes' or 'Step (short description of the step) Complete: No'.

If 'No', repeat what the user must do. If 'Yes', explain the next step.

If all steps are completed, say so."

Example user message:

Task: Drink water

Steps:

1. Take the water bottle
2. Drink from it
3. Put it away

Previously completed steps:

1. Take the water bottle

Next step:

Drink from it

Current frame:

[data:image/jpeg;base64,...]

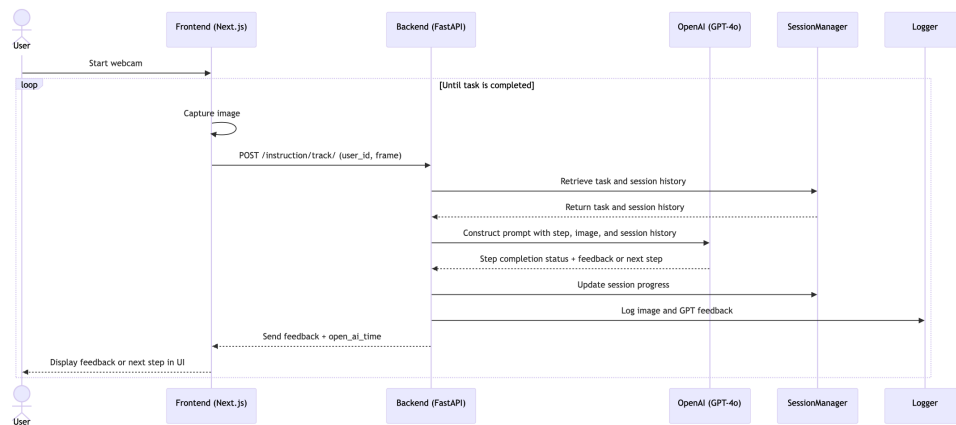


Figure 6.7: Sequence diagram of Live Instruction Tracking

Response Handling Each model response is timestamped and logged. On “Step Complete: Yes,” the backend advances to the next step; all responses and timings are stored for evaluation. This clear, structured interaction supports both stepwise tracking and easy analysis.

6.3.4 Challenges and Adjustments

During implementation and initial testing, several key challenges emerged:

- **Prompt Ambiguity:** Instructions such as “put it away” often led to inconsistent or incorrect completions, highlighting the necessity of concrete, observable criteria in prompts.
- **Model Hallucinations:** GPT-4o sometimes inferred steps or requirements not present in the original task description, particularly for vague or multi-step instructions.
- **Visual Clarity vs. Token Cost:** Initial attempts to lower JPEG quality by using more compression (to reduce API costs) had no effect, since token usage depends on image resolution. Higher resolution increased costs and latency without significant feedback accuracy gains.

- **Small Object Recognition:** Steps requiring fine-grained visual discrimination (e.g., distinguishing batteries from USB dongles) were particularly error-prone, especially with suboptimal lighting or occlusion.

To mitigate these issues:

- Prompts were refined to use explicit, stepwise instructions that helps GPT know when a task is completed (e.g., “Put the bottle away, anywhere out of the users hands is fine”).
- All images were sent at high JPEG quality (quality=100) but moderate resolution (640x480) for clarity and efficiency.

6.3.5 Controlled Evaluation of Instruction Tracking

This section documents an early-stage technical evaluation conducted during system development to assess instruction tracking performance using GPT-4o. These tests provided valuable insights and informed prompt design, backend logic, and system configuration.

The evaluation focused on several aspects:

- Accurate determination of step completion based on visual cues.
- Generation of clear and actionable guidance for the user’s next step.
- Consistent and timely system responses.
- Robustness in handling complex, ambiguous and multi-step instructions.

These criteria were selected to expose both the strengths and weaknesses of the instruction tracking pipeline in realistic scenarios.

Evaluation Setup

A diverse set of task scenarios was created to probe the system’s performance across a spectrum of instruction-following challenges. Each scenario consisted of a well-defined sequence of steps and was submitted through the web interface. The system’s responses were evaluated both for correctness and usability, primarily through my own observations as well as measured timings.

Several variables and metrics were systematically varied or recorded:

- **Step complexity:** Tasks ranged from simple, single-action steps to long, multi-layered procedures involving detailed object manipulations.
- **Image quality and resolution:** JPEG compression levels and resolutions were adjusted to test model sensitivity to visual fidelity. Unless otherwise specified, a standard resolution of 640×480 pixels was used.
- **Prompt clarity:** The same steps were phrased in both vague and concrete terms to evaluate model robustness to linguistic ambiguity.
- **Network latency:** Both total end-to-end roundtrip time and API-specific latency (backend to OpenAI and back) were recorded. Internet speed was measured before testing. The backend was locally hosted and exposed via a cloudflared tunnel, introducing a realistic backend-frontend setup.

Instruction Tracking Performance

Here are a few examples of the scenario's used for testing:

Scenario 1: Drinking Water A simple three-step task was chosen as a baseline:

1. Take the water bottle.
2. Drink from it.
3. Put it away.

GPT-4o identified the first two steps reliably. However, for the final “put it away” step, the model struggled—9 attempts were required before it would confirm completion. This was traced to the instruction's vagueness: “put it away” lacked clear, observable criteria. The system sometimes interpreted leaving the bottle on the table as completion, other times not, indicating a reliance on concrete instructions and visual evidence.

Scenario 2: Putting on a Green Sweater This scenario tested the model's ability to reason about object attributes (color) and state changes (from not wearing to wearing). Under good lighting and clear images, GPT-4o could eventually confirm sweater color, but under low lighting or with compressed images, it frequently failed to identify the correct color or missed the transition entirely. Multiple submission attempts were needed, highlighting the importance of visual clarity for model performance.

Scenario 3: Replacing Mouse Batteries A complex 13-step sequence challenged the system's ability to track longer tasks and recognize object-specific actions (e.g., removing a battery, distinguishing it from a USB dongle). GPT-4o managed most steps but occasionally confused objects, such as mistaking a USB dongle for a battery. On rare occasions, the model hallucinated steps not present in the instruction set, suggesting some limitations in fine-grained object recognition and sequence memory.

Scenario 4: Drinking Water (with Specific Definitions) Repeating Scenario 1 but with highly concrete prompts—such as requiring the user to visibly hold a blue bottle and bring it to their mouth—led to a marked improvement. The model confirmed each step in fewer attempts and gave more consistent feedback. This confirmed that explicit, concrete instructions significantly enhance tracking performance and reduce ambiguity.

General Observations

- **Clarity matters:** The more specifically a step was defined and visually demonstrable, the better the model performed.
- **Ambiguity and hallucination:** Vague instructions caused the model to hallucinate requirements or misinterpret actions.

- **Small object recognition:** Tasks requiring manipulation of small or similar-looking objects were error-prone, especially with low image quality or occlusion.
- **Response format consistency:** GPT-4o consistently formatted responses with “Step Complete: Yes/No” prefixes, enabling robust backend tracking.

JPEG Quality Considerations

Initially, images were compressed using JPEG quality=30, aiming to reduce token usage (and thus API cost). However, it became clear that OpenAI’s billing is based solely on image resolution, not file size. Comparative testing with JPEG quality=30 versus quality=100 revealed that higher quality substantially improved visual clarity and step recognition accuracy, with negligible effect on latency or token usage. For all subsequent experiments, images were submitted at 640×480, quality=100, balancing clarity and responsiveness.

Latency and Performance Metrics

To understand how instruction complexity and visual fidelity affected system responsiveness, two sets of experiments were conducted, with all images at JPEG quality=100.

A. Impact of Instruction Complexity All prompts used 640×480 resolution images. Results showed:

- Backend + network latency (Sending image from the frontend to the backend, processing it there and then send it + the full prompt to OpenAPI) averaged 186–280 ms.
- GPT-4o response times ranged from ~2500 ms for specific instructions to ~2700 ms for simpler ones.
- More complex, explicit instructions did not increase latency, and often improved tracking accuracy.

Instruction Type	GPT Time(ms)	Backend + Network(ms)	Total(ms)
Simple	2699	282	2981
Explicit	2532	171	2704

Table 6.2: Effect of Instruction Complexity on Latency (Low Resolution)

B. Impact of Image Resolution Challenging steps containing the use of smaller objects (e.g., distinguishing batteries from dongles) were repeated at 640×480 and 1280×960 resolutions.

- Backend + network latency rose by ~37%
- GPT-4o response time increased by ~32% at higher resolution.

- Token usage increased from ~550 to ~890 per prompt.
- Accuracy gains were marginal; most improvements came in edge cases, while hallucinations and tracking issues remained.

Resolution Type	GPT Time(ms)	Backend + Network(ms)	Total(ms)
Low	2658	186	2845
High	3518	261	3779

Table 6.3: Effect of Image Resolution on Latency (Same Instruction Set)

Key Takeaways Increasing image resolution increased latency and token usage substantially, with limited impact on step accuracy or object recognition. More detailed instructions consistently improved performance. As a result, the default configuration moving forward used high-quality, low-resolution images with more specific instructions.

Conclusion

This first evaluation demonstrates that **GPT-4o can reliably track procedural tasks when input is visually clear and instructions are specific**. High image resolution brought only minor accuracy improvements but at significant latency and cost, while vague prompts remained the main source of tracking errors and model hallucinations.

The system consistently tracked user progress in the intended step order, with mean response times of 2.63 seconds—acceptable for real-time interaction. Real-world variations in lighting and angle did introduce some robustness challenges, but these did not fundamentally undermine system functionality.

6.3.6 User Workshop Evaluation

To complement the controlled evaluation, a hands-on workshop was conducted where participants freely interacted with the instruction tracking prototype in a realistic lab environment. This allowed natural task creation, diverse object use, and real user phrasing, surfacing both strengths and weaknesses not visible in scripted scenarios.

Goals and Setup

Workshop objectives included identifying valuable use cases, observing user behavior with the prototype, gathering feedback on instruction clarity and system performance, and exposing limitations in backend logic or user interaction.

Participants submitted and executed tasks involving everyday and lab objects (electronics, wearables, ...), with complexity ranging from simple (e.g., “write ‘Hello’ on paper”) to multi-stage procedures (e.g., “set up a VR workstation”).

Instructions were written in natural language—sometimes in participants’ native languages—testing both robustness and multilingual handling. Step-by-step, users submitted images and received real-time feedback from the GPT-4o backend; all interactions were logged.

Frontend and Interaction Flow Feedback

The test interface was intentionally minimal, as the main focus was backend robustness. Still, users frequently requested:

- Progress indicators and clearer step navigation
- “Skip” or “override” buttons for error correction/testing
- Multilingual support and clearer task/step separation
- More guidance for writing precise instructions

UI suggestions will inform future AR development.

Selected Scenarios and Observations

Three representative tasks are shown to illustrate some findings:

- **Locking a locker:** Required clear visual evidence (e.g., code dial visibly set) to complete steps, highlighting the importance of specific, observable instructions.
- **Setting up a VR workstation:** Recognized device state (LEDs on/off, open apps on a monitor) and tracked hardware/software steps, but sometimes missed ambiguous physical actions unless gestures were explicit (e.g., “finger on button” to show that you turned a device on).
- **Writing “Hello” on paper:** Accurately followed stepwise micro-actions, distinguishing between writing letters and other pen/paper positions, and tolerated grouped steps like writing “LL” in one go.

Additional workshop insights:

- Instruction precision was critical; vague or ambiguous steps led to repeated failures or misrecognition.
- Visual clarity (e.g., holding objects in frame) directly improved completion recognition.
- Participants tested “fooling” the system or used various languages—most things were recognized, but output was always in English. Probably because the system prompts are in English.
- Suggestions included clearer user guidance for prompt-writing and more task feedback.
- Concerns were raised about privacy and liability (for safety tasks).

Detailed examples and notes are provided in Appendix B. This includes the user workshop notes (Section B.1) and representative anonymized logs (Section B.2).

Conclusion

The workshop validated that the system reliably tracked a variety of practical actions under real-world conditions. User feedback highlighted the need for observable steps, precise prompts, and improved interface affordances—insights that directly informed the next development iterations.

6.3.7 Summary

Sprint 2 successfully established the technical foundation for real-time, stepwise instruction tracking using GPT-4o as a multimodal large language model. Through systematic exploration, careful model and prompt selection, and a series of controlled and real-world evaluations, the system demonstrated robust progress tracking, actionable feedback, and reliable validation of user actions across diverse, multi-step tasks.

Key findings highlight that tracking accuracy is driven primarily by prompt clarity and visual observability, rather than image resolution alone. Ambiguous instructions or poor visual evidence resulted in repeated failures or occasional model hallucinations, while explicit, concrete prompts enabled consistent performance and clear user guidance. Response times remained within acceptable bounds for interactive AR applications.

The user workshop further validated the system’s effectiveness and surfaced valuable insights around user behavior, interface needs, and edge-case challenges, many of which directly informed subsequent iterations and prompt refinements.

Overall, Sprint 2 delivered a scalable, generalizable pipeline for instruction tracking and feedback, directly addressing requirements FR-5 through FR-7, laying critical groundwork for integration with AR frontends in future development sprints.

6.4 Sprint 3: Object Localization

6.4.1 Goals and Motivation

The goal of this development sprint was to build a robust, standalone Unity AR application that can translate 2D detections from an object detection model into accurate 3D positions within the AR environment. This functionality is critical for anchoring AR overlays and guiding users to interact with the correct objects or elements in their environment. This sprint bridges the gap between computer vision outputs and real-world spatial mapping.

This sprint directly addresses **FR-2** (estimate relative position; see Section 4.3), **FR-3** (provide positional data for AR overlays), and **FR-12** (overlay AR markers to highlight relevant objects). It also lays the groundwork for **FR-14** (update AR interface based on user progress) and **FR-15** (align AR overlays with MirageXR's data structure).

6.4.2 Exploration

At first glance, 2D-to-3D object localization appeared straightforward: take the 2D bounding box from a detector like YOLO, rescale it to align with the environment depth map produced by the AR device's LiDAR sensor, and use the depth at the bounding box center to reconstruct the 3D position. However, practical experimentation quickly revealed numerous edge cases and technical challenges, such as depth map misalignment, low resolution, and unstable placements.

This prompted exploration of two alternative or complementary approaches:

- **Mesh-based raycasting:** Raycasting from the camera through the bounding box center into Unity's reconstructed scene mesh to determine a 3D intersection point.
- **LiDAR-based point cloud proximity:** This approach was conceived as a refinement rather than a replacement. The reasoning was that by taking an initial 3D estimate from the depth map or mesh method and then mapping it to the closest real point captured by the LiDAR sensor, the final position would be anchored to an actual measured point in the environment. This was expected to improve localization accuracy, especially in cases where sensor noise or alignment issues affected the initial estimate.

Initial results from all methods revealed inconsistent and unstable placements, motivating a systematic investigation into the sources of error and the relative performance of each approach.

6.4.3 Implementation

This section details the technical implementation of the object localization pipeline, structured around the three primary methods developed: depth-based projection, mesh-based raycasting, and LiDAR point cloud proximity matching.

YOLOv8n Object Detection

A classic YOLOv8n model pre-trained on the COCO dataset was used to detect a limited subset of classes (e.g., bottle, cell phone, laptop). To simplify testing of the object localization pipeline and avoid cluttered results, only predictions for these classes were accepted. The AR application captures an RGB frame, sends it to the backend for inference, and receives JSON-encoded labels and bounding box coordinates as outputs.

Depth-Based 3D Localization

Goal: Estimate the 3D position of a detected object using the ARKit-provided environment depth map and the center of its 2D bounding box.

The system follows these steps:

1. **Bounding Box Centering:** Given bounding box coordinates (x_1, y_1, x_2, y_2) , compute the center:

$$(x_c, y_c) = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

2. **Depth Coordinate Rescaling:** Rescale the center to the resolution of the depth map, yielding (p_x, p_y) .
3. **Depth Smoothing:** Sample a 3×3 window around (p_x, p_y) , averaging valid values to denoise.
4. **Screen-to-World Ray Construction:**

At this point, the bounding box center is located in image (pixel) space and mapped to the corresponding location in the ARKit depth map. However, to reconstruct a 3D position in the world, it is necessary to convert this 2D location into a direction vector in camera (viewport) space, as used by Unity's rendering pipeline.

Unity represents points on the screen using normalized *viewport coordinates*, where $(0, 0)$ is the bottom-left and $(1, 1)$ is the top-right. In the code, these coordinates are typically stored as `viewportPoint` or `normalizedViewport`. This normalization is required because image, depth, and screen resolutions may differ, and viewport coordinates provide a device-independent way to describe points on the camera's view.

We map the pixel coordinates (p_x, p_y) to viewport coordinates (u, v) as:

$$u = \frac{p_x}{W_{\text{depth}}}, \quad v = 1 - \frac{p_y}{H_{\text{depth}}}$$

where W_{depth} and H_{depth} are the width and height of the depth map, and the $1 - v$ flip is necessary because image coordinates typically have $y = 0$ at the top, whereas Unity's viewport has $y = 0$ at the bottom. To obtain a direction in camera-local (clip) space, we could manually compute:

$$\text{localPoint} = ((u - 0.5) \cdot 2, (v - 0.5) \cdot 2, 1)$$

This transformation shifts the origin to the center of the viewport (where (0.5, 0.5) is center screen) and maps it to the normalized device coordinate space $[-1, 1]$, with $z = 1$ pointing forward.

However, in practice, this transformation is handled internally by Unity's built-in `Camera.ViewportPointToRay()` method, which returns a world-space ray corresponding to the given viewport coordinates. Therefore, we directly use this API to obtain a ray and avoid manual computation.

The vector is then normalized:

$$\text{localDirection} = \frac{\text{localPoint}}{\|\text{localPoint}\|}$$

This gives a unit direction vector in the camera's local coordinate system pointing through the detected object.

5. **World Transformation:**

In Unity, AR objects exist in world space, not camera-local space. Thus, the direction vector from the previous step must be rotated and translated into the world coordinate frame.

Let $\mathbf{c}$ denote the AR camera's position and R its rotation (as a rotation matrix or quaternion). We rotate the local direction vector by the camera's rotation:

$$\text{worldDirection} = R \cdot \text{localDirection}$$

This step accounts for the actual physical orientation of the device at the moment of image capture.

The final estimated 3D position of the object is then given by:

$$\text{worldPosition} = \mathbf{c} + \text{worldDirection} \times \text{depth}$$

where depth is the mean depth value extracted from the depth map.

Why is this necessary? This transformation chain ensures that, no matter the device orientation, screen resolution, or AR session state, the estimated 3D position of the detected object is mapped correctly into the real-world environment.

Testing and Early Findings: At first, I expected this approach to provide robust object localization. However, real-world testing quickly revealed that it was unreliable outside ideal conditions. Labels frequently appeared misaligned, floating, or missing—particularly for small or off-center objects. I attributed these failures primarily to the low resolution of the ARKit depth map on my device, which yielded only a coarse representation of the scene.

Mesh-Based 3D Localization

Goal: Estimate the 3D position of a detected object by raycasting from the 2D bounding box center into Unity's reconstructed environment mesh.

1. **Bounding Box Centering and Viewport Normalization:** The bounding box center (x_c, y_c) is computed and normalized to viewport coordinates as in the previous method (section 6.4.3).
2. **Ray Generation:** A ray is generated from the AR camera through the normalized viewport point using Unity's `Camera.ViewportPointToRay()` method. This produces a 3D ray pointing from the camera into the scene, passing through the detected object's position in screen space.
3. **Mesh Intersection:** The ray is cast into the environment mesh using Unity's `Physics.Raycast()`. If the ray intersects with any collider associated with the AR mesh (e.g., via `ARMeshManager`), the point of intersection is recorded as the estimated 3D position of the detected object.
4. **Result:** If an intersection is found, a label or marker is placed at this point in the AR scene.

Testing and Early Findings: To address the limitations of the depth-based approach, I explored mesh-based raycasting, reasoning that Unity's reconstructed AR mesh might offer higher spatial resolution. However, my early visualizations suggested that the mesh was either too sparse or too slow to generate, resulting in further missed or inconsistent label placements. In retrospect, some of these issues stemmed from errors in my mesh visualizations, which led me to incorrectly blame mesh quality.

LiDAR Point Cloud Refinement

Goal: Refine the estimated 3D position of a detected object by matching it to the nearest real-world point in the LiDAR point cloud, as provided by Unity's `ARPointCloudManager`.

This method builds upon the previous approaches by anchoring detections to actual measured points in the environment.

1. **Initial Position Estimation:** As in the previous methods, an initial world-space position for the detected object is estimated—either via depth map projection or mesh raycasting.
2. **Candidate Filtering in Screen Space:** All active point clouds are iterated. Each LiDAR point is projected from 3D world space into 2D screen space using the AR camera's perspective. A point is considered a candidate if its 2D projection falls within the bounding box of the detected object.
3. **Proximity Matching:** For each candidate point, compute the Euclidean distance to the initially estimated world position. The point with the smallest distance is chosen as the refined object location.
4. **Label Placement:** If a suitable match is found, a label or marker is placed at the corresponding 3D point in the AR scene.

Testing and Early Findings: Localization remained unreliable, which I again attributed to scenes with sparse or outdated point clouds, much like the challenges encountered with the mesh method.

6.4.4 Adjustments and Improvements

A key challenge in this sprint was the unpredictability of localization results. When objects were centered, unobstructed, and close to the camera, all three methods (depth-based projection, mesh raycasting, and point cloud matching) performed well. However, minor scene variations frequently led to significant inconsistencies—objects accurately localized in one frame could be misplaced entirely in the next, which complicated debugging. The following examples in Figure 6.8 visually illustrate these inconsistencies and the sensitivity of the localization pipeline.

Below, a few key parts of the debugging process will be described.

Investigating Root Causes

Interestingly, the similarity of localization failures across all methods suggested that the root causes were not specific to any individual approach. Instead, these recurring mislabelings pointed to more fundamental, method-agnostic problems in the overall logic and data flow. Based on these observations and code inspections, I identified several likely sources of misalignment and inconsistency:

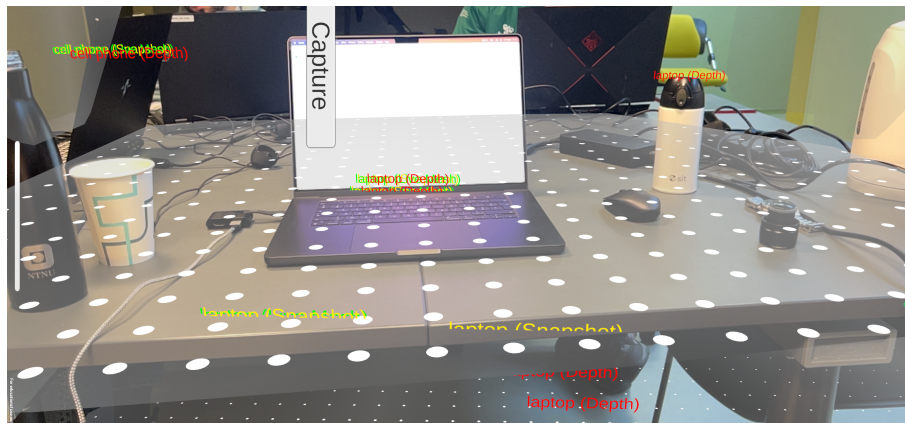
- **Sensor data quality:** The coarse resolution of the ARKit depth map often limited the accuracy of 3D projections, especially for small or off-center objects.
- **Pose desynchronization:** Latency introduced by asynchronous backend processing—sending images for detection and receiving bounding boxes—could result in a mismatch between the camera pose at the time of capture and the pose used for projection, causing drift.
- **Bounding box misalignment:** The axis-aligned nature of YOLO bounding boxes meant they sometimes included background or were not tightly fitted, especially for objects not aligned with the image axes. As a result, projecting the bounding box center could miss the true object location.

Debugging Different Localization Methods

To efficiently evaluate and compare the different localization methods, a convenient toggle was implemented in the Unity Inspector for the `ObjectLocatorManager` GameObject. This allowed quick switching between estimation techniques and facilitated debugging methods individually or simultaneously. A visual example of this debugging setup is illustrated in Figure 6.9.

Fixing Depth Map Mirroring

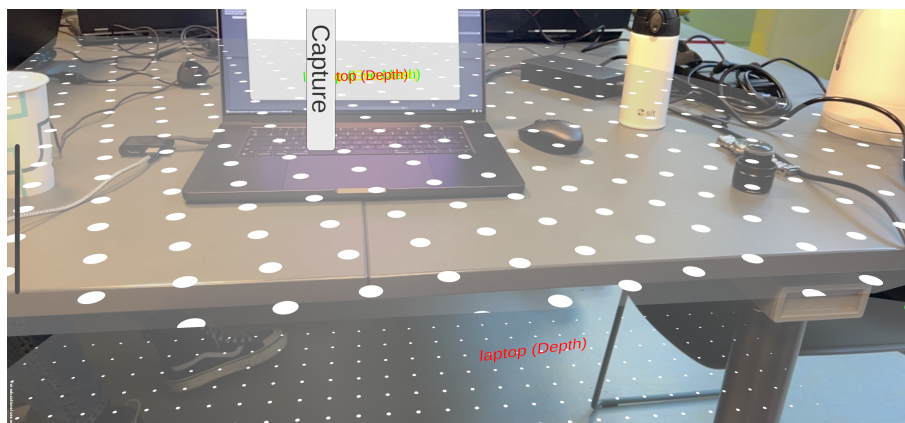
During development, I noticed that the bounding boxes on the RGB image and the corresponding ARKit depth map did not align properly, leading to inaccurate projections. After investigating, I discovered that I had accidentally applied a horizontal flip transformation to the depth images during conversion, which caused the misalignment. Removing this flip transformation ensured that the bounding



(a) On the first few attempts, all methods were very accurate. After a while you see random scattering.



(b) Wrongly recognized cell phone and floating laptops



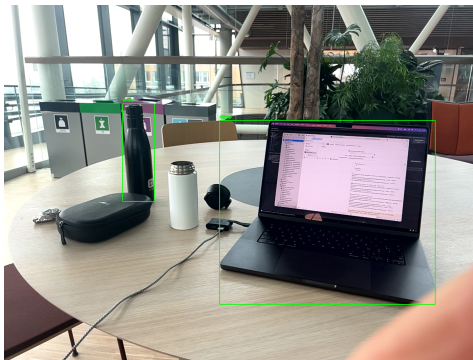
(c) Perfect first attempt, when laptop was centered. Then it put the label on the floor when it wasn't.

Figure 6.8: Inconsistency across localization attempts on the same object under slightly different conditions



Figure 6.9: Unity Inspector toggles for enabling and comparing different object localization methods during debugging.

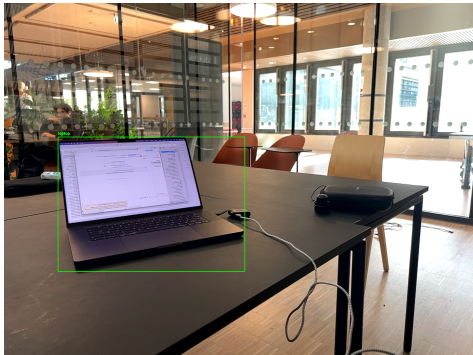
boxes aligned correctly with the RGB images. Figures 6.10a to 6.10d illustrate this debugging process and the corrected alignment.



(a) RGB image with bounding boxes overlaid (pre-fix).



(b) Horizontally mirrored depth map (pre-fix).



(c) RGB image with bounding boxes overlaid (post-fix).



(d) Corrected depth map with properly aligned bounding boxes (post-fix).

Figure 6.10: Debugging bounding box alignment across RGB and depth modalities.

This correction significantly improved the reliability of object detection, ensuring consistent inputs for subsequent depth-based localization steps.

Visual Debugging Tools

To help identify other potential problem sources, two visual debugging tools were implemented:

- **Ray Visualization:** Rays were drawn from the camera through the projected bounding box centers to inspect direction and scene intersection (Figure 6.11).
- **Bounding Box Overlays:** Bounding boxes and their supposed centers were rendered on screen within Unity using YOLO coordinates transformed to screen space (Figure 6.12).

The ray visualization did not reveal any issues, as the rays passed straight through the misplaced labels without indicating error sources. However, the 2D bounding boxes revealed significant misalignments, proving that the transformation from YOLO-normalized coordinates to Unity's screen or viewport coordinates was flawed. This realization led to deeper investigations into Unity's camera projection pipeline.

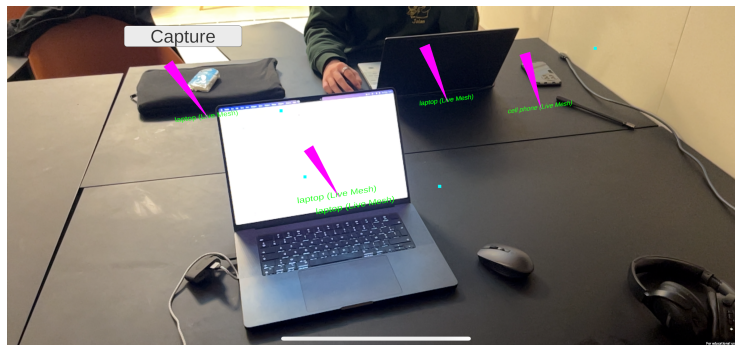


Figure 6.11: Visualization of debug rays originating from the centers of YOLO-projected bounding boxes in screen space, illustrating direction and intersection with the 3D scene. The blue dots are supposed bounding box centers.

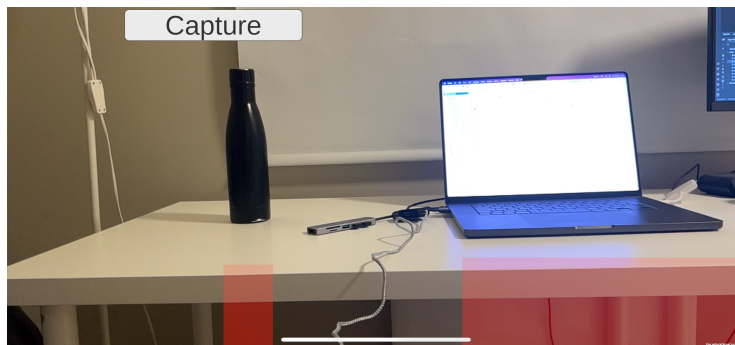


Figure 6.12: Rendered YOLO bounding boxes in Unity's screen space, highlighting coordinate transformation misalignments between 2D detections and 3D projections.

Unity Hierarchy and Canvas Cleanup

During the initial development of the object localization pipeline, my limited experience with Unity XR development resulted in a cluttered and inconsistent scene hierarchy. As shown in Figure 6.13, the initial project structure contained multiple overlapping UI canvas objects and redundant subcomponents, complicating both debugging and development.

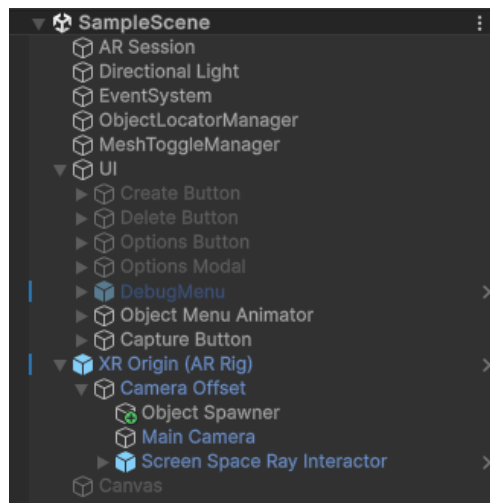


Figure 6.13: Initial Unity hierarchy with multiple overlapping UI canvases and redundant subcomponents.

Specifically, the root canvas was configured with a resolution of 1170 px by 2532 px in portrait mode, while the app was designed to run in landscape on the iPhone screen. The UI canvas containing interactive elements had a different resolution (800 px by 600 px), leading to UI inconsistencies and bounding box misalignments.

The cluttered hierarchy, combined with overlapping components in the AR Session and XR Origin objects, made it difficult to determine which object controlled specific AR functionality, further complicating debugging. Additional screenshots of the initial and corrected hierarchy configurations are included in Appendix C.

To resolve these issues, the hierarchy was reorganized, redundant components were removed, and a single UI canvas was used. Figure 6.14 shows the improved hierarchy with a streamlined structure. This decluttering step improved maintainability and the resolution match resolved earlier bounding box alignment problems.

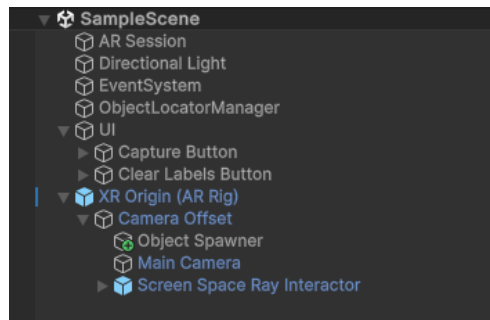


Figure 6.14: Decluttered Unity hierarchy with a single UI canvas and no redundant subcomponents.

Viewport Projection Testing

After resolving the hierarchy and canvas configuration issues, the bounding boxes were correctly overlaid, but the labels were still misplaced. It was time to investigate the raycasting logic. To validate this logic, I initially placed labels 1 meter directly in front of the camera and observed that they aligned as expected. I then extended this approach to project labels 1 meter in front of the *center* of each detected bounding box.

While the labels consistently appeared at the correct distance, they often did not visually align with the detected object, as shown in Figure 6.15. This indicated a potential error in how the bounding box coordinates were translated into Unity’s projection system. I began to suspect that the issue might lie in the conversion between viewport and screen space coordinates.

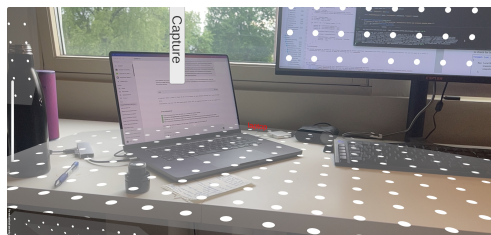


Figure 6.15: Example of a label incorrectly projected into the scene, highlighting a misalignment between the label position and the corresponding object.

From this point on, debugging efforts focused primarily on the mesh-based approach. This decision was reinforced by a meeting with the MirageXR team, who—drawing on their AR development experience—recommended this method as likely to yield more reliable results. Additionally, I considered that the 256×192 resolution of the ARKit depth map might introduce inaccuracies, and that the mesh-based method aligns with the working approach described in the Xair paper [6].

Mesh Snapshot Raycasting Experiment

Now that the bounding boxes were correctly overlaid, I wanted to investigate whether the delay between image capture and eventual raycasting could cause misplaced labels. Because the camera could have moved and the mesh might have changed, I suspected that live raycasting into the mesh might not always be accurate. To address this, I implemented functionality to store the mesh state at the time of image capture and to raycast into this stored snapshot instead of the live one.

Each snapshot was constructed by storing the geometry and transformation matrices of all ARKit mesh components in the scene. Later, I implemented a custom ray-triangle intersection test using the Möller–Trumbore algorithm [36].³

```
foreach (var snapshot in meshSnapshots)
{
    Ray localRay = new Ray(
        snapshot.localToWorld.inverse.MultiplyPoint(ray.origin),
        snapshot.localToWorld.inverse.MultiplyVector(ray.direction)
    );

    var mesh = snapshot.mesh;
    var triangles = mesh.triangles;
    var vertices = mesh.vertices;

    for (int i = 0; i < triangles.Length; i += 3)
    {
        Vector3 v0 = vertices[triangles[i]];
        Vector3 v1 = vertices[triangles[i + 1]];
        Vector3 v2 = vertices[triangles[i + 2]];

        if (IntersectRayTriangle(localRay, v0, v1, v2, out float distance))
        {
            if (distance < closestDistance)
            {
                closestDistance = distance;
                hitPoint = ray.origin + ray.direction * distance;
                hitFound = true;
            }
        }
    }
}
```

Code listing 6.2: Manual raycast through snapshot mesh using triangle iteration

This method functioned as intended but was extremely slow and did not provide significant improvements over Unity's built-in `Physics.Raycast` into the live mesh.

³For an accessible explanation of the Möller–Trumbore algorithm, see <https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection.html>.

As a result, I discontinued the snapshot approach in favor of continuing with live raycasting.

Viewport Normalization Bug

Problem: Initially, I computed screen-space rays by normalizing the bounding box center using the `lastDepthWidth` and `lastDepthHeight`—the resolution of the environment depth map. This approach was a holdover from the earlier depth-based projection method.

However, Unity's physics engine (`Physics.Raycast`) operates in screen or viewport space, not depth map space. As a result, this coordinate mismatch often caused rays to be cast in incorrect directions—especially when the resolution of the RGB image used for detection differed significantly from the depth resolution.

Fix: The solution was to use Unity's normalized viewport coordinate system, which ranges from (0,0) in the bottom-left corner to (1,1) in the top-right. Unity's `ViewportPointToRay` method expects these coordinates to generate rays correctly from the AR camera.

Corrected implementation:

```
Vector2 viewport = new Vector2(imageCenter.x, 1f - imageCenter.y);  
Ray ray = arCamera.ViewportPointToRay(viewport);
```

Code listing 6.3: Corrected viewport normalization for raycasting

Explanation:

- `imageCenter.x` and `imageCenter.y` are the bounding box center in normalized image space.
- Unity's viewport space has $Y = 0$ at the bottom, while image space has $Y = 0$ at the top, hence the flip using `1f - imageCenter.y`.
- `ViewportPointToRay` returns a ray originating from the camera position and directed through that point in the viewport.

This correction eliminated mismatches caused by using incorrect coordinate systems and aligned ray directions with Unity's internal physics and rendering logic.

Misaligned Pose-Based Projection

With the viewport normalization bug corrected, I next tackled the problem of incorrect direction rotation due to camera pose changes. The mesh isn't the only thing that can change during the delay between image capture, object detection, and final projection. The AR camera—and thus the user—will most likely move and rotate while the backend processes the detection. To address this, I implemented logic to store the AR camera's position and rotation at the exact moment a photo was taken. This saved pose would later be used to project the detection into the environment as if the projection had happened from the original camera position.

Problem: However, my initial implementation misunderstood how Unity's `ViewportPointToRay` works. I incorrectly assumed that the resulting direction vector was in local camera space, so I applied the saved rotation directly to it:

```
Vector3 viewportPoint = new Vector3((float)px / lastDepthWidth, 1f - (float)
    py / lastDepthHeight, 0f);
Vector3 direction = arCamera.ViewportPointToRay(viewportPoint).direction;
Ray rayPC = new Ray(lastCameraPosition, lastCameraRotation * direction);
```

Code listing 6.4: Incorrect pose-based ray projection (before correction)

Note: `viewportPoint` represents the normalized (u,v) coordinate in Unity's viewport space.

The direction returned by `ViewportPointToRay` is already in *world space*, based on the camera's current orientation. By applying the saved rotation on top of it, I inadvertently rotated a direction that was already rotated—effectively introducing compounded orientation errors.

Fix: I needed to reverse the current camera rotation and reapply the stored one. This required three steps:

1. Use `ViewportPointToRay` to get the current world direction.
2. Undo the current camera's rotation by applying its inverse, transforming the ray direction back into local camera space.
3. Apply the saved camera rotation to reconstruct what that direction would have been at the time of image capture.

Corrected implementation:

```
Vector3 currentRay = arCamera.ViewportPointToRay(normalizedViewport).
    direction;
Vector3 worldRayDirectionFromStoredCamera =
    lastCameraRotation * (Quaternion.Inverse(arCamera.transform.rotation) *
        currentRay);
Ray rayMesh = new Ray(lastCameraPosition, worldRayDirectionFromStoredCamera
    .normalized);
```

Code listing 6.5: Reprojecting rays using stored pose

Explanation:

- `currentRay` is the direction from the current camera in world space.
- Applying `Quaternion.Inverse(arCamera.transform.rotation)` transforms that direction into local camera space.
- Applying `lastCameraRotation` then transforms the vector as if it originated from the earlier pose.
- The ray is now cast from `lastCameraPosition` using the reconstructed direction.

This correction significantly reduced large misalignment errors when the user had moved between image capture and label projection.

6.4.5 Testing & Evaluation

Accuracy

Extensive qualitative testing demonstrated that the final pipeline consistently places labels accurately and stably on the intended objects. Even when the user moved the device between image capture and backend response, labels appeared correctly anchored to the detected objects. This consistency confirms that the final system effectively handles coordinate transformations, pose desynchronization, and real-world mesh variability.

Although it might seem logical to complement this qualitative testing with a quantitative benchmark, I decided against a formal accuracy benchmark for this development phase. The main reason is that label placement accuracy in an AR environment depends on a wide range of factors that are difficult to control and isolate in a small-scale test. For example:

- Object distance and size strongly affect mesh resolution and visibility.
- Real-world lighting conditions can affect the quality of object detection (e.g., shadows or glare).
- ARKit mesh updates and point cloud density can vary dynamically based on camera movement and scene complexity.

While I considered measuring label placement error at different distances for three object types (large, medium, and small) and classifying results as “spot on,” “slightly off,” or “way off,” such an evaluation would ultimately be redundant for this proof-of-concept. The evaluation would largely mirror the qualitative observations already documented: the system performs reliably in typical scenarios but struggles with small, distant, or cluttered scenes.

Latency

Formal latency testing was not prioritized due to tight time constraints and consistent good performance observed during development. 3D localization appeared near-instantaneous. YOLO object detection had already been evaluated in Sprint 1 and the additional raycasting step introduced in this sprint added minimal overhead, reducing the need for further performance testing at this stage given the time constraints of the project.

Remaining Limitations

Despite its robustness, the implementation still has a few known limitations:

- **Overlapping Objects:** When two objects are close together in 2D but at different depths, the ray may intersect the closer object, leading to occasional misplacement of labels.
- **Small or Distant Objects:** The AR mesh resolution can be too coarse to

reliably detect small or distant objects, sometimes resulting in missed intersections or floating labels.

- **Object Tracking:** The system does not currently track objects over time. If an object moves after detection, the label will remain at the initial position, requiring a new localization pass. Additionally, the system cannot reliably distinguish between different objects of the same class label (e.g., a water bottle vs. a small medicine bottle both classified as “bottle”), which may complicate tasks involving multiple similar items.
- **Mesh Generation Delay:** The AR mesh generation takes a few seconds to stabilize after session start or sudden camera movements. If an image is captured during this period, the mesh may be incomplete, leading to inaccurate raycasting results.

In summary, the qualitative performance testing demonstrated that the system works well under most realistic conditions and highlighted key areas for potential improvement.

6.4.6 Summary

This sprint culminated in a stable, accurate, and extensible AR object localization system that bridges 2D computer vision outputs with 3D AR environments (Figure 6.16). Initial assumptions about simple coordinate projections and depth map lookups gave way to a deeper understanding of the challenges involved in aligning 2D detections with real-world geometry.

The final implementation raycasts into Unity’s reconstructed environment mesh using the saved camera pose, ensuring accurate label placement even when the user moves between detection and projection. This method proved to be the most stable and consistent across testing scenarios.

The following improvements were key to achieving this stability:

- **Viewport Normalization:** Bounding box center coordinates are now correctly normalized to Unity’s viewport coordinate system (with a vertical flip), ensuring accurate ray generation.
- **Pose Reprojection:** Rays are cast using the stored AR camera pose at the time of image capture, which corrects for user movement and rotation during backend processing.
- **Mirroring Correction:** A horizontal image flip was removed to align YOLO-detected bounding boxes with Unity’s image coordinate system.
- **UI and Hierarchy Cleanup:** The Unity scene hierarchy was simplified, and UI canvas settings were aligned with the device’s resolution and orientation.
- **Debugging Tools:** Visual tools were introduced to render bounding boxes, rays, and labels, greatly accelerating the debugging process and helping isolate critical issues.

The next phase will focus on integrating this pipeline into a reusable module, supporting further development within the MirageXR ecosystem and beyond.

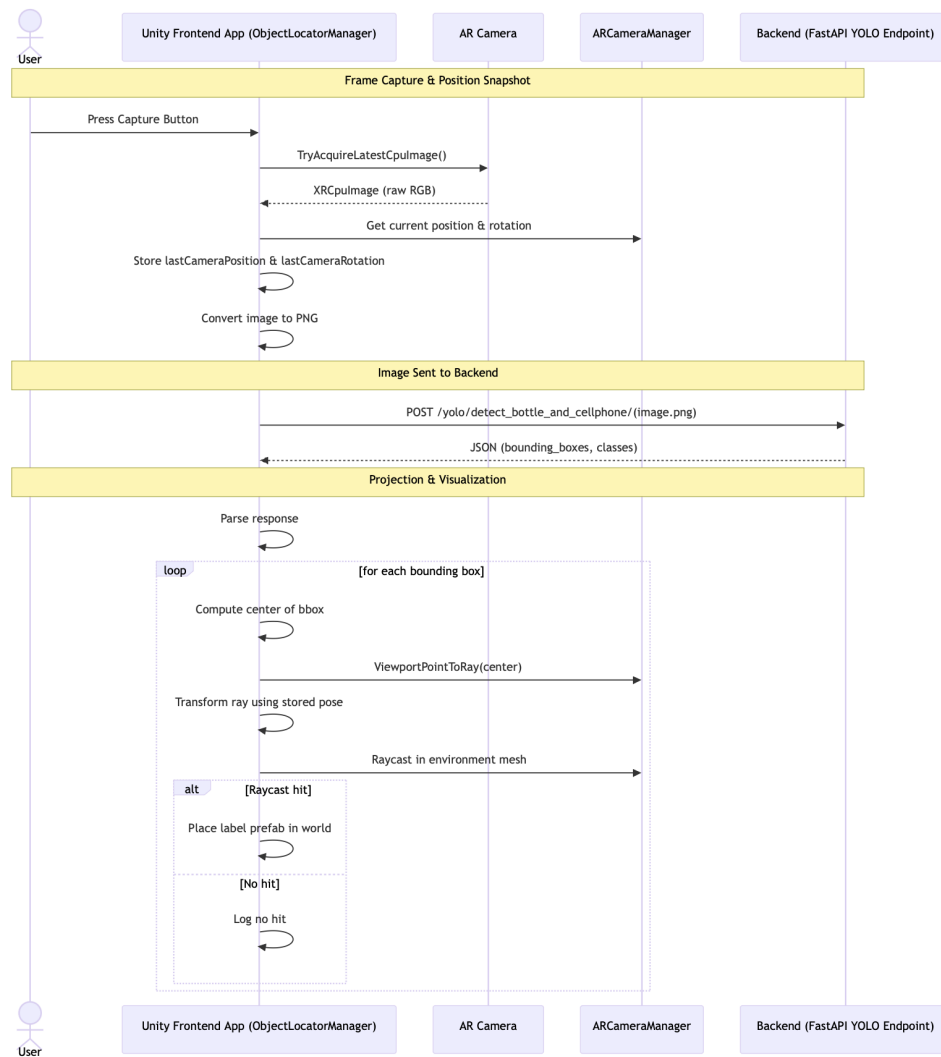


Figure 6.16: Sequence diagram: Object Localization Flow from Unity to backend and back

6.5 Sprint 4: Open-Sourcing RecognX: A Modular Unity Package for Object Localization

6.5.1 Goals and Motivation

Following the successful development of a working prototype for real-time object localization in augmented reality, it became clear that the core functionality—previously implemented within a single application—could be transformed into a reusable, open-source Unity package. Based on a recommendation of the MirageXR team, this sprint focused on modularizing the object localization logic (addressing **FR-1–FR-3**) as a manager `MonoBehaviour` script designed for easy integration into any AR system. An objective was to ensure the package would simplify integration with MirageXR’s architecture (**FR-4**) and data structure (**FR-15**; see Section 4.3). Future sprints will expand the package to include the system’s remaining functionalities.

This section describes the design and implementation of the resulting package, named *RecognX*, including packaging decisions, architectural considerations, technical challenges, and current and planned capabilities.

6.5.2 Exploration and Package Design

Unity packages are the standard format for distributing modular functionality in Unity. They encapsulate source code, assets, dependencies, and metadata in a structured and reusable format. Unlike traditional Unity projects, where logic may be scattered across scenes and scripts, packages enforce a clear boundary between reusable code and project-specific integration.

A Unity package typically includes:

- A `package.json` file for versioning, metadata, and dependencies.
- Optional `.asmdef` (assembly definition) files for defining isolated code assemblies.
- Source folders organized under `Runtime/`, `Editor/`, and `Resources/`.
- Support for Git or local distribution via Unity’s Package Manager.

Using `.asmdef` files allows the package to declare explicit dependencies (e.g., on `Unity.XR.ARFoundation`), improving build performance and enforcing modularity. Figure 6.17 shows the high-level structure of the *RecognX* package.

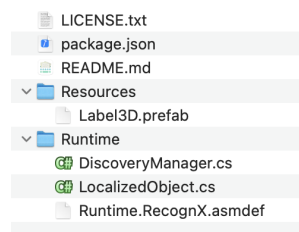


Figure 6.17: Directory structure of the *RecognX* Unity package.

6.5.3 Implementation

The system was restructured into a central `DiscoveryManager` class, responsible for coordinating all internal operations. The design goals were:

- A minimal public API for users.
- Clean abstraction of Unity ARFoundation internals.
- Extensibility for future integration (e.g., instruction tracking).

Figure 6.18 presents the class structure. All external configuration (e.g., references to Unity components like `ARCameraManager` and `ARMeshManager`) is exposed through serialized fields, enabling drag-and-drop configuration in the Unity Editor.

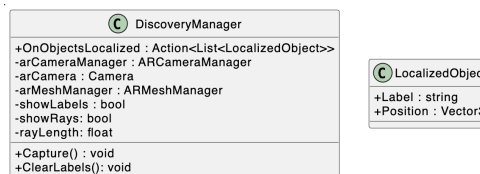


Figure 6.18: Class diagram of the core `DiscoveryManager` and the `LocalizedObject` class.

The high-level runtime flow is shown in Figure 6.19 and proceeds as follows:

1. The user invokes `Capture()`, which caches the current camera pose.
2. An image is captured and converted to PNG format.
3. The image is asynchronously posted to a backend YOLO server.
4. Upon receiving bounding boxes, the system projects them into 3D space using raycasting.
5. If intersections are found, optional visual labels are placed.
6. Results are returned via the `OnObjectsLocalized` event.

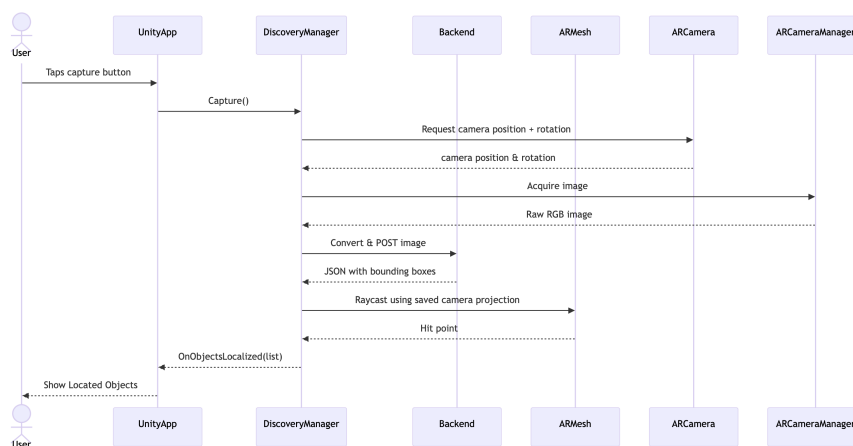


Figure 6.19: Runtime execution flow of the `DiscoveryManager`.

6.5.4 Testing and Validation

To validate the reusability of the RecognX package, a minimal Unity project was created using only the exported package. This project demonstrated correct use of the `OnObjectsLocalized` callback to place labels in the 3D scene.

The following script shows a minimal example of subscribing to the event and instantiating a label prefab at the localized position:

```
public class LabelPlacer : MonoBehaviour
{
    public GameObject labelPrefab;
    public DiscoveryManager discoveryManager;

    void Start()
    {
        discoveryManager.OnObjectsLocalized += objects =>
        {
            foreach (var obj in objects)
            {
                Instantiate(labelPrefab, obj.Position, Quaternion.identity)
                ;
            }
        };
    }
}
```

Code listing 6.6: Example integration script using the RecognX package.

This script subscribes to the `OnObjectsLocalized` event exposed by `DiscoveryManager` and instantiates a label prefab at each localized object's 3D position. Figure 6.20 shows the result of running this scene with a detected object, where the label appears precisely on the object surface.



Figure 6.20: Screenshot of label prefab instantiated at detected object location.

6.5.5 Challenges and Adjustments

Several integration challenges were encountered during packaging:

- **Assembly Definitions and Tooling:** Rider initially failed to resolve dependencies due to missing project metadata. This was addressed by regenerating project files and enabling external packages in Unity's External Tools.
- **ARFoundation Dependencies:** Assembly definitions required explicit references to ARFoundation packages. Missing these resulted in compiler errors for types like `ARMeshManager`.

These challenges were resolved by adopting a modular architecture, isolating dependencies, and thoroughly testing the package across environments.

6.5.6 Public API Overview

Table 6.5 describes the key members of the `DiscoveryManager` API.

Member	Description
<code>Capture()</code>	Starts the localization workflow. Captures image, sends it to the backend, and performs 3D projection.
<code>OnObjectsLocalized</code>	Event triggered with a list of localized objects and their positions.
<code>arCamera</code>	Reference to the main Unity AR camera, used for ray generation.
<code>arCameraManager</code>	Unity component for image acquisition from ARKit.
<code>arMeshManager</code>	Provides access to the live spatial mesh.
<code>showLabels</code>	Whether to display labels at object positions.
<code>ClearLabels()</code>	Clears all labels that were being shown.
<code>showRays</code>	Whether to display debug rays that were cast for 5 seconds.
<code>rayLength</code>	The length of the debug rays.

Table 6.5: Public members of the `DiscoveryManager` class.

6.5.7 Conclusion and Next Steps

The `RecognX` package is licensed under the MIT License, enabling both academic and commercial use.

By releasing this package, the goal is to contribute a robust and extensible foundation for researchers and developers working on spatial AR applications. Its modular design and open license could allow it to evolve with use cases beyond this thesis and will allow for easier integration within the `MirageXR` platform.

The next sprint will focus on merging the package with the instruction-tracking pipeline, creating a unified system that bridges object recognition with user task guidance.

6.6 Sprint 5: Backend Refactor for Object-Aware Instruction Tracking

6.6.1 Goals and Motivation

This sprint focused on **preparing the backend for integrating instruction tracking with object localization**—an essential step to deliver a seamless AR experience.

Initially, the system was designed with simplicity in mind: instruction tracking was implemented using a plain-text list of steps, while object localization ran as a separate module. This separation supported rapid prototyping but created a disconnect between textual instructions and real-world objects. The backend could not dynamically determine which items users needed at each step, making it difficult to guide users effectively in an AR environment.

Moreover, the old backend lacked structured step tracking—GPT alone was relied upon to keep track of the current step and completed steps, which made real-time guidance unreliable and complicated any attempt to highlight context-relevant objects.

The goal of this sprint was to **refactor the backend's data model and session management** to address these shortcomings. Specifically, this involved:

- Introducing explicit object references within tasks and steps, enabling dynamic alignment with object localization in Unity.
- Designing a more structured session manager that maintains the current step index on the backend itself.
- Leveraging GPT-based extraction to automate the process of linking objects to steps, improving scalability and usability.

This sprint further addresses **FR-5** through structured session tracking and advances **FR-1**, **FR-6–FR-8**, and **FR-9–FR-10** via an object-aware task model and cleaner APIs. On the non-functional side, it lays the foundation for **NFR-3** and **NFR-4** (MirageXR-aligned state tracking).

6.6.2 Exploration

Before implementing the refactor, I analyzed the existing backend architecture to pinpoint obstacles to integrating instruction tracking with object localization.

The original design stored tasks as a title and a list of plain-text steps. While this enabled rapid development and effective GPT-based progress tracking, it lacked structure to associate physical objects with individual steps. As a result, the backend could not determine which items to highlight at each step or align these steps with object detections from the Unity-based localization system.

This analysis confirmed that the task model needed explicit object references at the step level and that the backend had to expose this data for use by the Unity package. These insights directly informed the backend refactoring discussed next.

6.6.3 Implementation

To address these issues, the backend was refactored using an object-oriented model centered on three core abstractions as shown in Figure 6.21.

- **Task:** Represents the entire activity, uniquely identified by UUID to ensure consistent identification across updates and throughout user sessions.
- **Step:** Represents an ordered instruction within a task, referencing relevant objects.
- **Object:** Represents a real-world item, including its name from the task step (step_name), mapped YOLO class (yolo_name), and COCO class ID (yolo_class_id).

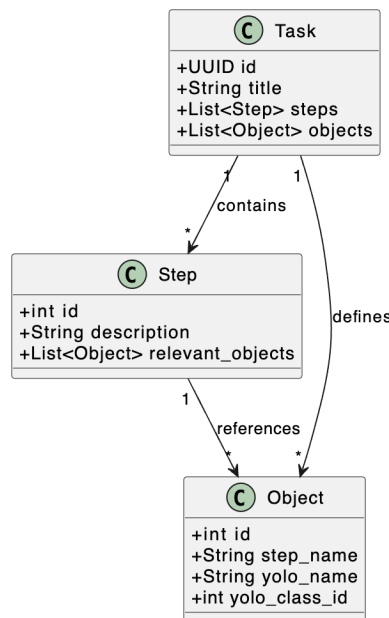


Figure 6.21: Class diagram: Task has many Steps, and each Step references one or more Objects, which are also stored globally on the task level.

This structure allows the system to:

- Provide a global object list for preparation and preview.
- Reference only relevant objects per step to aid contextual guidance.
- Seamlessly exchange structured data with the Unity package.

Automatic Object Extraction using GPT

To automate the population of object lists in tasks, the system uses GPT to analyze task descriptions and step instructions. This ensures consistent, scalable annotation of object references without manual effort.

Triggering the Extraction. When a user submits a task title and step instructions via the Next.js frontend, the backend sends a structured prompt to GPT. This prompt includes the task information and instructs GPT to extract relevant physical objects, returning the result in JSON format.

Objects must:

- Appear explicitly or implicitly in at least one instruction step.
- Belong to the 80 standard COCO classes supported by YOLOv8 or, if a custom detection model is used, to that model's defined classes.

For example, instructions like “take the bottle of water” and “pour it into the cup” result in the objects “bottle” and “cup” being extracted and mapped to their respective YOLO classes.

Prompt Structure. The prompt includes:

- A **system message** that instructs GPT to extract YOLO-detectable objects from the instructions and return valid JSON—without extra text.
- A **user message** that contains the task title, steps, and a reminder of YOLO-detectable classes, including synonyms and generalizations.

A detailed example of the prompt is provided in Appendix D.

Backend Handling. The backend parses GPT's response, strips any markdown formatting, and decodes the JSON. Each object is instantiated as an `Object` and added to the `Task.objects` and `Step.relevant_objects` list, with step references assigned based on the `used_in_steps` field. An example is shown in Listing 6.7.

```
1 {  
2   "objects": [  
3     {  
4       "step_name": "water bottle",  
5       "yolo_name": "bottle",  
6       "yolo_class_id": 39,  
7       "used_in_steps": [1, 2]  
8     },  
9     {  
10      "step_name": "wine glass",  
11      "yolo_name": "wine glass",  
12      "yolo_class_id": 40,  
13      "used_in_steps": [2, 4]  
14    }  
15  ]  
16 }
```

Code listing 6.7: Example GPT Object Extraction Output

Task Sessions and Step Tracking

To support real-time instruction tracking, the system maintains a dedicated session state for each active user. This state is initialized when a task is submitted and updated as the user progresses through each instruction step. The refactored session manager aligns backend logic with frontend requirements—enabling dynamic object-aware guidance.

Why the Refactor Was Needed. The previous session manager stored only a task title and a plain-text list of steps—sufficient for basic GPT instruction tracking but inadequate for object-aware AR experiences. It lacked:

- Tracking of the current instruction step.
- Representation of objects linked to specific steps.
- Progress tracking through completed steps.

Without these, real-time highlighting of relevant objects and adaptive task guidance were impossible.

Enhanced Session Management. The refactored session manager stores a fully structured Task, including steps and object references. Each session now maintains:

- `current_step_index`—for step-by-step progress.
- `completed_steps`—to record finished instructions.
- Which objects are relevant to which step, enabling dynamic highlighting of relevant items.

Task Lifecycle: Saving and Submitting. Tasks can be created or updated in the frontend, then saved to the backend after GPT-based object extraction. When users submit a task, the backend acknowledges receipt and initializes a session with:

- The structured task data.
- The current step index (starting at 0).
- An empty list of completed steps.

This flow ensures that sessions always start from a consistent, object-aware state.

Benefits of the Refactor. The new session manager transforms the system into a robust, object-aware task engine that:

- Provides step-level progress tracking.
- Highlights relevant objects dynamically.
- Supports resetting or repeating tasks easily.

Overall, this refactoring enables consistent, semantically meaningful guidance aligned with the AR environment.

6.6.4 Challenges and Adjustments

Several challenges arose during this sprint. Ensuring consistent, valid JSON output from GPT was crucial for parsing object references reliably. This required iterative prompt refinements and extensive testing with different task instructions. Additionally, integrating the refactored models posed risks of breaking existing functionality, especially in routes that handled task submission and instruction tracking. Careful versioning and incremental testing helped mitigate these risks. Improved error handling was also added to catch GPT failures and provide clear feedback to the user interface.

6.6.5 Testing and Evaluation

Testing and evaluation focused on verifying that the backend refactor met the sprint objectives and maintained system stability. Key testing areas included:

- **Object Extraction and Mapping:** Confirmed that GPT correctly identified and mapped objects to their corresponding YOLO classes, including edge cases like synonyms or implicit references.
- **Session Initialization and Step Tracking:** Verified that sessions started with the correct step index, tracked completed steps accurately, and aligned object references with step execution.
- **System Stability:** Ensured that existing functionality—such as task creation, submission, and instruction tracking—continued to operate as expected after the refactor.

Overall, the refactored backend performed as intended, enabling robust object-aware instruction tracking while maintaining backward compatibility with earlier features.

Qualitative Observations: Semantic Behaviors. During testing, two notable semantic behaviors emerged from GPT’s output:

1. **Cross-referencing based on pronouns.** GPT can understand when a pronoun like “it” refers to an object mentioned in a previous step. For example:

```
Step 1: Take the bottle in your hand
Step 2: Open it
```

Even though “bottle” is only mentioned in Step 1, GPT correctly understands that “it” in Step 2 refers to the same object. This allows for consistent object highlighting across multiple steps.

2. **Distinguishing between instances of the same YOLO class.** GPT also handles object disambiguation when multiple instances belong to the same YOLO class. For instance:

```
1 {
```

```
2  "objects": [  
3    {  
4      "step_name": "wine bottle",  
5      "yolo_name": "bottle",  
6      "yolo_class_id": 39,  
7      "used_in_steps": [1, 4]  
8    },  
9    {  
10     "step_name": "other bottle",  
11     "yolo_name": "bottle",  
12     "yolo_class_id": 39,  
13     "used_in_steps": [3, 4]  
14   }  
15 ]  
16 }
```

Code listing 6.8: Example of disambiguating identical YOLO classes using distinct step_names.

Despite sharing the same YOLO class, GPT assigns them distinct step_names to allow the frontend and backend to differentiate and track them individually.

These behaviors demonstrate GPT's ability to parse instructions at a deeper semantic level, rather than relying on surface-level keyword matching. The result is a more robust and scalable object extraction pipeline capable of interpreting natural, loosely structured task instructions.

6.6.6 Summary

This sprint delivered a refactored backend that bridges the gap between textual instructions and real-world object highlighting. By adopting structured task models and integrating automated GPT-based object extraction, the system now supports object-aware instruction tracking—ready for integration with the AR experience. Key backend improvements included:

- Adopting a more object-oriented design with structured Task, Step, and Object classes, improving alignment with physical objects and enabling per-step object references.
- Expanding and updating API endpoints to handle object extraction and richer task data.
- Enhancing the session manager with a current step index and object-aware steps—enabling step-by-step guidance, real-time tracking, and consistent task progress monitoring.
- Implementing robust error handling, ensuring graceful failure and clear feedback in cases of GPT errors or user issues.
- Consolidating repeated logic into reusable utility functions for better maintainability.

6.7 Sprint 6: Integration and Final System Overview

6.7.1 Goals and Motivation

Building on the backend refactoring achieved in Sprint 5, this sprint focused on **integrating object-aware instruction tracking into the Unity package**, transforming it into a fully functional AR guidance system. The aim was to bridge the gap between semantic task guidance—powered by GPT—and physical object detection—enabled by YOLO and AR mesh—by developing a modular Unity package that unifies these capabilities.

This sprint extended the package to:

- Support task sessions that incorporate object references retrieved from the backend.
- Manage the AR flow, including object localization prior to task execution.
- Guide users step-by-step, highlighting relevant objects and dynamically updating the interface based on user progress.

The implementation addressed key functional requirements, including:

- **FR-11:** Display step-by-step instructions with integrated object highlights.
- **FR-12:** Highlight relevant objects in AR.
- **FR-14:** Support dynamic interface updates.
- **FR-15:** Align with MirageXR's data structure.

6.7.2 Sprint Overview

The structure of the sprint is as follows:

Exploration (subsection 6.7.3) Discusses the initial design for of integrating object discovery and the evolution toward a dynamic, step-by-step approach.

System Design (subsection 6.7.4) Describes the modular architecture of the ARInstructionManager package, the design principles, and the internal state machine governing the package. Also includes an in-depth walk-through of component interactions during typical usage.

Usage (subsection 6.7.6) Summarizes the public API, configuration requirements, and provides a simplified example of integrating the package into an AR application.

Challenges and Adjustments (subsection 6.7.7) Presents issues encountered during development and the solutions implemented.

Testing and Evaluation (subsection 6.7.8) Details the tests conducted to validate the implementation, including a testing frontend walkthrough and user evaluation and QR-code with an example video (Figure 6.31). Links to some development repositories are also here.

Summary (subsection 6.7.9) Concludes with a summary of key achievements.

6.7.3 Exploration

Initially, the design required users to locate all relevant objects for the entire task before beginning execution (Figure 6.22). While simple to implement, this approach proved cumbersome: users had to find every object upfront, even if some were only needed later, which created friction and reduced flexibility.

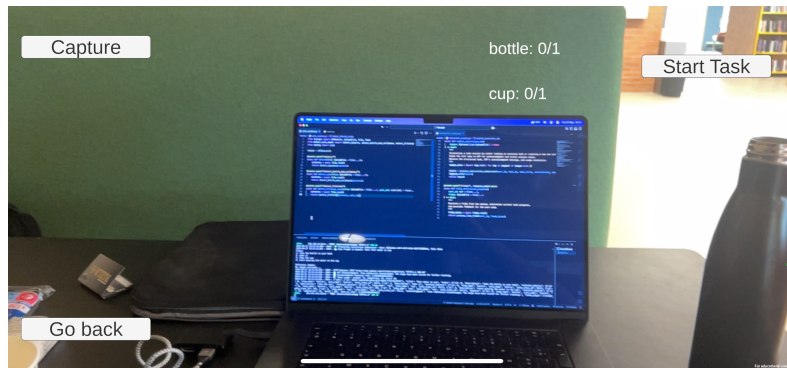


Figure 6.22: Old Approach: Users had to locate all relevant objects before beginning the task.

Through testing, I realized that allowing users to locate objects dynamically—step by step—would better match real-world workflows (Figure 6.23). Instead of forcing object discovery all at once, the system now requests only the objects relevant to the current step. This change improves usability and keeps the experience more fluid and engaging.

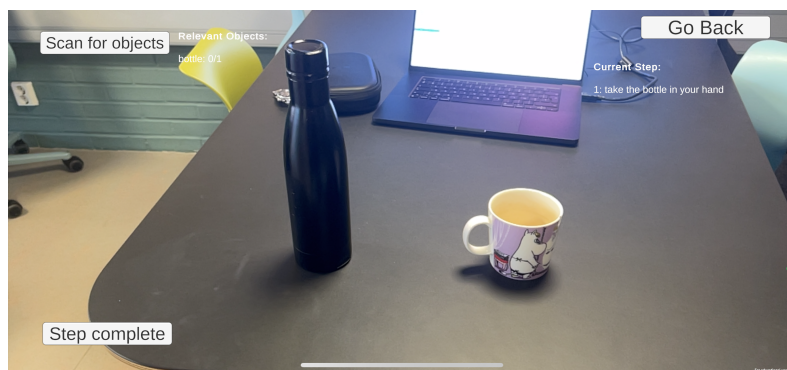


Figure 6.23: New Approach: Users can locate required objects step-by-step during execution.

This decision was refined within the sprint, demonstrating agile responsiveness to user experience challenges.

6.7.4 System Design

Modular Architecture of the `ARInstructionManager` Package

The `ARInstructionManager` Unity package adopts a highly modular design, dividing responsibilities across dedicated manager classes (see Figure 6.24). Its core components include:

- **`ARInstructionManager`:** Serves as the central coordinator of the package, initializing and connecting the other managers to orchestrate the entire AR instruction session. It manages application states, initiates processes (such as object discovery), and listens for events, ensuring that the system operates cohesively. Importantly, it is the sole entry point exposed by the package, meaning that applications interact with the system exclusively through the `ARInstructionManager`.
- **`DiscoveryManager`:** Handles object detection and localization. It captures camera frames, sends them to the backend, and raycasts the 2D detections into the AR mesh to determine 3D positions. Detected objects are converted from `YoloDetection` to `LocalizedObject` instances. This abstraction simplifies computer vision integration.
- **`SessionManager`:** Manages task progress, step tracking, and user state. It tracks the current instruction step, manages transitions, and verifies step completion with the backend. Its encapsulated logic makes it reusable for various AR instruction workflows.
- **`BackendService`:** Abstracts all backend communication, including sending data to object detection and instruction tracking endpoints. By centralizing HTTP interactions and data handling, it ensures changes to backend APIs affect only this component, preserving modularity.

During development, an initially planned `TaskController` class was removed to avoid overlapping responsibilities with `ARInstructionManager` and `SessionManager`. This decision clarified component boundaries and improved the system's maintainability.

Design Principles

In addition to its modular structure, the package's design follows several guiding principles to support extensibility, maintainability and ease of integration:

- **UI-Independent Core:** The logic layer is completely decoupled from any UI components. All interactions between the core system and the UI are event-driven, enabling developers to implement diverse user interfaces — from voice-based interactions to heads-up AR displays — without modifying the underlying logic.
- **Clear Separation of Concerns:** Each manager class handles a distinct aspect of functionality, with minimal dependencies on others. For example, `DiscoveryManager` focuses solely on object detection, unaware of instruc-

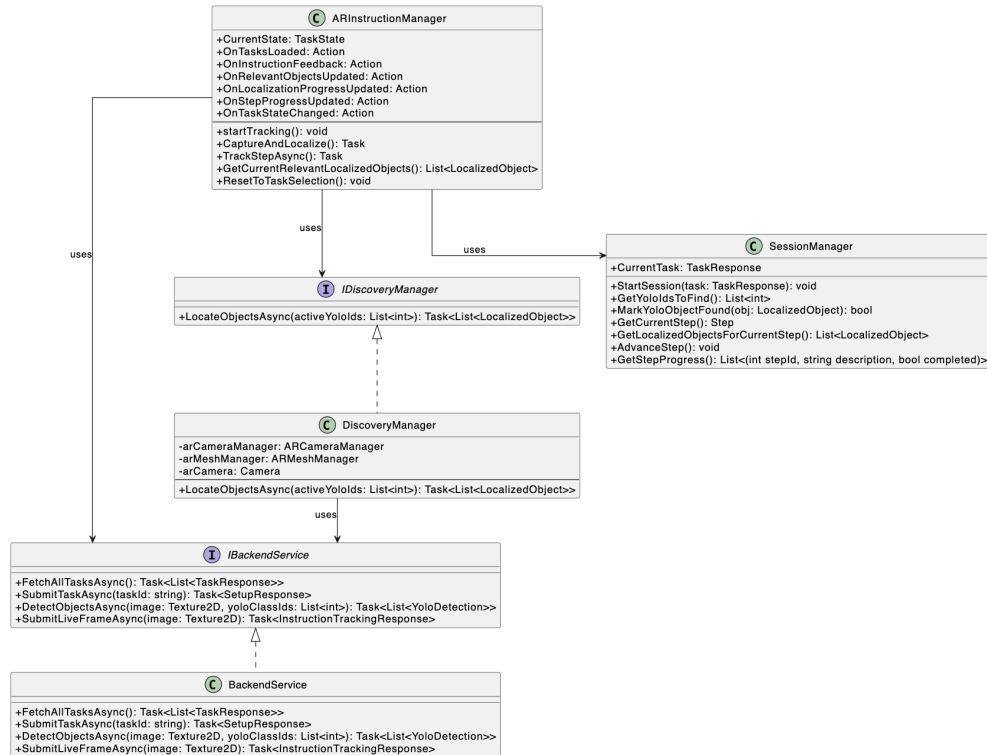


Figure 6.24: UML Class Diagram of the ARInstructionManager package. Some implementation details (such as certain private fields, internal utilities, and lower-level helper classes) have been omitted for clarity and readability.

tion steps or feedback, while SessionManager manages task progression without knowing the details of object localization. This separation minimizes the impact of changes and makes the codebase easier to maintain and extend.

- **Incremental Integration with the Frontend:** A basic UI was integrated for early testing and evolved alongside the logic layer, influencing event design and flow. Initially, the UI polled for state changes, but this was replaced with event-driven updates (e.g., OnStepProgressUpdated, OnInstructionFeedback), simplifying UI integration. Additionally, certain direct-access methods (e.g., GetCurrentRelevantLocalizedObjects()) remain available to provide flexibility for different integration patterns. The package's open-source nature further supports community contributions and customizations.

Internal State Machine for Task Execution

To manage the AR instruction flow, the package implements an internal state machine defined by the TaskState enum. This state machine governs how the system progresses through an AR session, ensuring a logical and consistent user experi-

ence. The four states in the updated design are:

- **Idle:** The system's default state, waiting for the user to select a task.
- **TaskSummary:** Activated after a task is selected. The system displays all steps and relevant objects, allowing users to review the task and prepare for execution.
- **Tracking:** The core state where the user performs each instruction step. The `SessionManager` determines which objects are relevant for the current step and collaborates with the `DiscoveryManager` to dynamically locate these objects. This allows users to discover and interact with objects step-by-step, especially when they might not recognize an object on their own. The system continuously communicates with the backend to verify step completion and highlights the relevant objects in the AR scene, ensuring smooth, guided progress.
- **Completed:** The final state reached after all steps are successfully completed and verified. The session is marked as finished, and the system can reset to `Idle` for the next task.

An overview of the state transitions is shown in Figure 6.25.

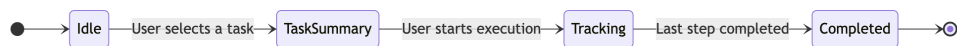


Figure 6.25: State Machine Diagram: Overview of the `TaskState` transitions showing the AR instruction session flow.

Enabling the Classic State Pattern. While the internal state management in the `ARInstructionManager` package uses a simple enum-based approach, its design intentionally exposes clear state transitions via events (such as `OnTaskStateChanged`). This design enables external applications that use the package to implement the classic State Pattern if desired. For example, an application might implement different UI overlays or AR interaction modes depending on the current state. This modular and extensible approach ensures that the package remains adaptable to a variety of AR applications.

6.7.5 Component Interaction During AR Task Execution

This section walks through a typical usage scenario of the `ARInstructionManager` package within an AR frontend application. It details how the main components interact with each other and the user during an AR instruction session. It also showcases typical use of the package by an AR frontend application. Each phase is illustrated with a sequence diagram to clarify the flow of information and control throughout the system.

App Startup

Description: When the user launches the app, the `FrontendApp` subscribes to the `OnTasksLoaded` event of the `ARInstructionManager`, which then uses `BackendService` to retrieve the list of available tasks. Once these are fetched, the `ARInstructionManager` emits the `OnTasksLoaded` event, which the frontend listens for to display the available tasks to the user (see Figure 6.26).

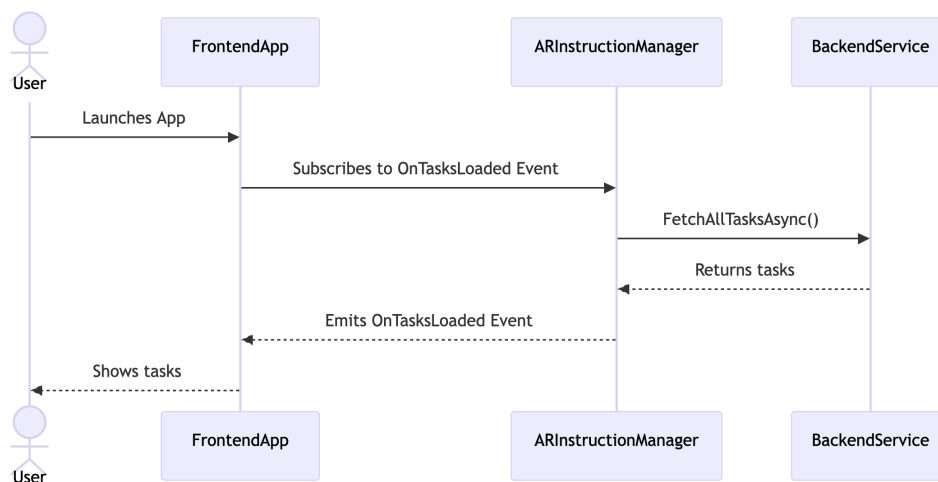


Figure 6.26: Sequence diagram illustrating interactions during the app startup.

Task Selection

Description: After the user selects a task, the `FrontendApp` calls `SelectTaskAsync()` on the `ARInstructionManager`, passing the selected task ID. The `ARInstructionManager` then sends this ID to the backend using the `BackendService` to retrieve detailed task data. Once the backend responds, the `ARInstructionManager` instructs the `SessionManager` to start the session and loads the steps and relevant objects. The `ARInstructionManager` then emits events that the frontend can use to display the task summary screen, allowing the user to review the entire task before execution (see Figure 6.27).

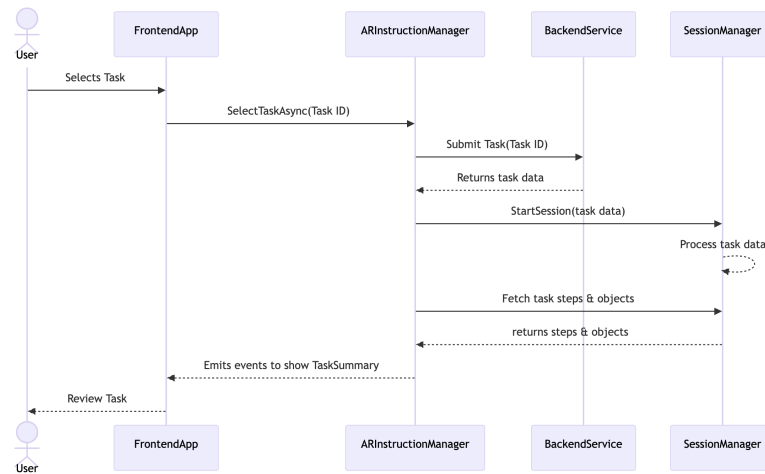


Figure 6.27: Sequence diagram illustrating interactions during the task selection phase.

Starting a Task

Description: Once the user taps “Start Task” in the UI, the FrontendApp calls `StartTracking()` on the `ARInstructionManager`. This triggers the system to transition into the Tracking state. The `ARInstructionManager` then queries the `SessionManager` for the current step information and the relevant objects required for that step.

After retrieving this data, the `ARInstructionManager` emits events to the FrontendApp, providing the step description and a list of relevant objects to be located. Allowing the user to try performing the step and get help locating objects they might not recognize themselves (see Figure 6.28).

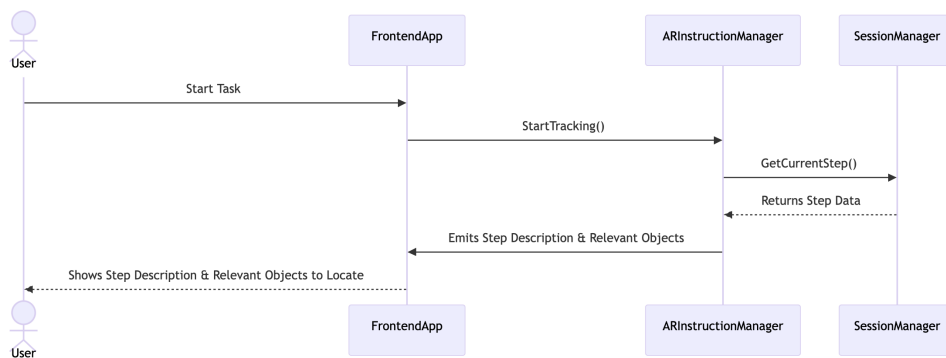


Figure 6.28: Sequence diagram illustrating interactions during task initiation, showing how step description and relevant objects are presented to the user.

Step Execution 1: Object Scanning

Description: During the Tracking phase, the user performs each instruction step, optionally guided by AR overlays that highlight relevant objects. If the user does not recognize an object or wants assistance, they can tap a “Scan for Objects” button in the UI.

This triggers the `ARInstructionManager`’s `CaptureAndLocalize()` method, which first queries the `SessionManager` for objects that are still relevant and not yet located in the current step. The `DiscoveryManager` then captures the camera view and sends it to the backend via the `BackendService` for object detection. Detected bounding boxes are projected into the AR scene using raycasting, resulting in `LocalizedObjects` that are returned to the `SessionManager`.

The `SessionManager` stores these `LocalizedObjects`, ensuring they remain available and can be highlighted again if they are required in a future step. Finally, the `ARInstructionManager` emits an event containing the newly localized objects, which the frontend app can use to display them in the AR view. The package also has an optional built-in object highlighting system that can be used.

This design supports repeated scanning at any time, ensuring users can locate objects step-by-step as needed. The complete sequence is illustrated in Figure 6.29.

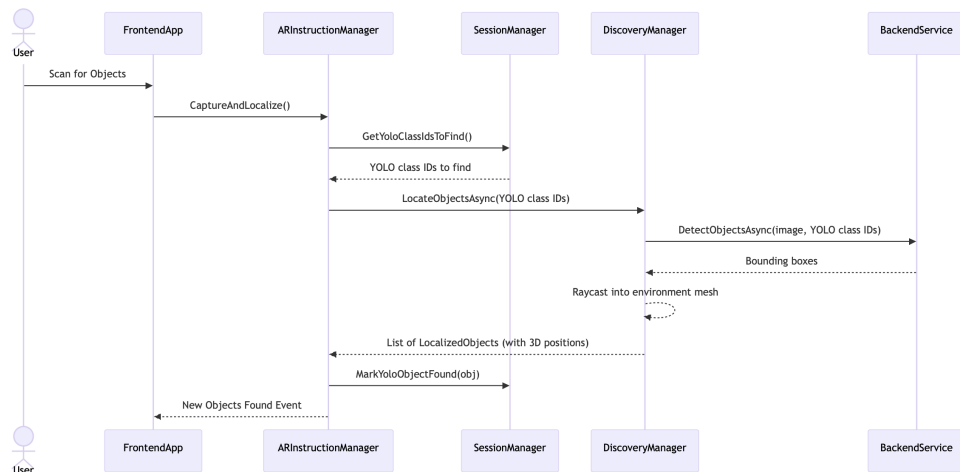


Figure 6.29: Sequence diagram for dynamic object scanning during step execution.

Step Execution 2: Step Completion

Description: When the user believes they have completed a step, they can tap a “Complete Step” button in the UI. This triggers the `ARInstructionManager`’s `TrackStepAsync()` method, which captures the user’s view and sends it to the backend for verification. The backend then returns a structured response indicating whether the step was successfully completed, along with GPT-generated feed-

back and other relevant metadata.

The system emits `OnInstructionFeedback` and `OnStepProgressUpdated` events (simplified in Figure 6.30 for clarity), enabling the frontend to update the UI. If the step is complete, the `SessionManager` advances to the next step. Additionally, if the built-in object highlighter is enabled, it updates the AR highlights using `UpdateLabelsForCurrentStep()`. If auto-scan is enabled (configurable at task summary), it is restarted to automatically discover objects relevant to the new step.

If the step is not complete, the system remains on the current step and provides feedback, allowing the user to try again as needed.

This iterative loop continues until all steps are successfully completed, enabling users to progress at their own pace. The complete sequence of interactions is illustrated in Figure 6.30.

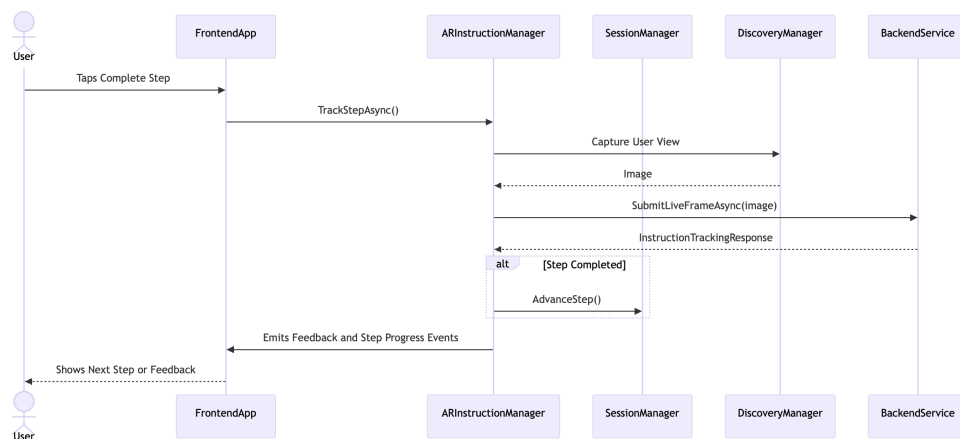


Figure 6.30: Sequence diagram for step completion and progression.

Task Completion

Description: Once all steps are successfully completed and verified, the system transitions to the Completed state, concluding the AR instruction session.

Summary This walkthrough demonstrates how the `ARInstructionManager` coordinates the AR instruction flow by integrating the `SessionManager`, `DiscoveryManager`, and `BackendService` into a seamless, interactive experience. By combining backend intelligence (YOLO and GPT) with clear UI feedback and AR overlays, the system provides users with step-by-step guidance that adapts to their real-world progress.

6.7.6 Usage

Public API Package

This section summarizes the key public API provided by the `ARInstructionManager` package.

Public Properties

Property	Description
Instance	Singleton instance of the <code>ARInstructionManager</code> . Accessed as <code>ARInstructionManager.Instance</code> .
CurrentState	The current state of the AR instruction session (<code>Idle</code> , <code>TaskSummary</code> , <code>Tracking</code> , <code>Completed</code>).
AutoScanEnabled	Whether automatic object scanning is enabled during Tracking. Can be toggled via UI.
AutoScanInterval	Interval (in seconds) between automatic scans when <code>AutoScanEnabled</code> is true.

Table 6.6: Public properties of `ARInstructionManager`.

Public Methods

Method	Description
<code>SelectTaskAsync(string taskId)</code>	Selects a task by ID and prepares the session. Should be called after a user selects a task.
<code>StartTracking()</code>	Begins task execution, transitioning the state to <code>Tracking</code> .
<code>CaptureAndLocalize()</code>	Captures the camera view and triggers object detection via the backend, returning results as events.
<code>TrackStepAsync()</code>	Submits the current view to the backend for step verification and triggers relevant feedback and progression events.
<code>GetAllStepsForCurrentTask()</code>	Returns a list of all steps with their completion status. Useful for displaying the task overview.
<code>GetAllObjectsForCurrentTask()</code>	Returns all relevant objects for the current task.
<code>GetCurrentRelevantLocalizedObjects()</code>	Returns localized objects for the current step, supporting AR overlays.
<code>ResetToTaskSelection()</code>	Resets the package to the <code>Idle</code> state, clearing all task-related data.

Table 6.7: Public methods of `ARInstructionManager`.

Public Events

Event	Description
OnTasksLoaded	Fired when tasks are fetched from the backend. Provides a list of available tasks.
OnTaskStateChanged	Fired whenever the internal state changes (e.g. Idle → Tracking). Used for UI transitions.
OnInstructionFeedback	Fired after backend verification of a step, containing GPT feedback and step completion status.
OnStepProgressUpdated	Fired when step progress is updated (e.g. step completed). Useful for updating progress bars or checklists.
OnLocalizationProgressUpdated	Fired when object progress updates (labels found per step). Useful for updating object lists.
OnRelevantObjectsUpdated	Fired after object localization, providing the latest found objects for AR overlays.

Table 6.8: Public events of ARInstructionManager.

Dependencies and Configuration

The ARInstructionManager package requires several AR components and offers configurable options to customize its behavior. These must be set by the integrating application:

- **AR Dependencies:**
 - `arCameraManager`: A reference to the ARCameraManager component for capturing camera frames.
 - `arMeshManager`: A reference to the ARMeshManager component for accessing the AR environment mesh.
 - `arCamera`: The camera used for AR rendering.
- **Visualizations:**
 - `useBuiltInLabelRenderer (bool)`: Enables or disables the built-in object label renderer, which automatically places 3D labels in the AR scene. If disabled, the application is responsible for rendering object highlights independently.
 - `labelPrefab (GameObject)`: The prefab used for rendering labels on localized objects. A default prefab is provided by the package, but developers can assign a custom prefab to change the visualization style while still using the built-in renderer.
- **Auto-Scan Options:**
 - `autoScanEnabled (bool)`: Enables automatic scanning of relevant ob-

- jects during the Tracking state.
- `autoScanInterval (float)`: Sets the interval in seconds between automatic scans.

These configurations must be assigned either in the Unity Inspector or programmatically to ensure the package operates correctly in the AR environment.

Usage Example

A typical integration of the package in a frontend AR application (see Code listing 6.9) involves subscribing to events and calling the appropriate methods as the user interacts with the UI. The required dependencies and configuration options discussed in section 6.7.6 should be properly assigned in the Unity Inspector or programmatically.

Runtime Integration: The application typically:

- Calls `SelectTaskAsync` after task selection.
- Calls `StartTracking` to begin the AR instruction session.
- Calls `CaptureAndLocalize` on user request to find objects dynamically.
- Calls `TrackStepAsync` to submit the current step for backend verification.
- Listens to events like `OnInstructionFeedback`, `OnStepProgressUpdated`, and `OnRelevantObjectsUpdated` to update the UI and provide user feedback.

```
manager.OnInstructionFeedback += response => {  
    feedbackText.text = response.response;  
};  
  
manager.OnTasksLoaded += tasks => {  
    dropdown.AddOptions(tasks.Select(t => new TMP_Dropdown.OptionData(t.  
        title)).ToList());  
};  
  
startTaskButton.onClick.AddListener(() => {  
    manager.StartTracking();  
});  
  
captureButton.onClick.AddListener(() => {  
    manager.CaptureAndLocalize();  
});  
  
completeStepButton.onClick.AddListener(() => {  
    manager.TrackStepAsync();  
});
```

Code listing 6.9: Simplified example of integrating the `ARInstructionManager` in a Unity application, demonstrating event subscriptions and essential method calls.

This example shows how the `ARInstructionManager`'s event-driven design integrates seamlessly into a typical AR frontend application, enabling dynamic user feedback and step-by-step AR guidance.

6.7.7 Challenges and Adjustments

Fixing Final Step Completion and UI Feedback Bug

During late-stage integration testing, an issue emerged where the final step of a task was completed internally, but the frontend failed to reflect this: the last step did not turn green, and no task completion feedback was shown. This pointed to a logic flaw in the step advancement and UI update pipeline.

After debugging, the root cause was identified in the session management logic. Specifically, the `AdvanceStep()` method in `SessionManager` incremented the `currentStepIndex` without verifying bounds. On the final step, this caused `currentStepIndex` to exceed the valid step range. Subsequent methods such as `GetCurrentStep()` and `GetLocalizedObjectsForCurrentStep()` assumed the index was valid, leading to silent failures or skipped updates when null steps were encountered.

To resolve this, two targeted fixes were implemented:

- A guard clause was added to `AdvanceStep()` to prevent incrementing the step index beyond the number of available steps.
- `GetLocalizedObjectsForCurrentStep()` was updated to return an empty list when the current step is invalid, avoiding any exceptions or incorrect object filtering.

These changes ensured that even after completing the final step, UI events such as feedback updates and visual object highlights continued to function as expected. Importantly, the system now handles task completion gracefully without any crashes or silent failures, improving both user experience and system robustness.

6.7.8 Testing & Evaluation

Walkthrough: Testing Frontend Application

To demonstrate the `ARInstructionManager` package in action, a testing frontend application was developed to showcase the full AR task cycle—from app launch to task completion. This walkthrough illustrates the practical usage of the package, highlighting each interaction step by step. The QR code links to a video demonstration, see Figure 6.31⁴.

⁴<https://youtu.be/277WDaRMSHE>



Figure 6.31: QR code linking to a video walkthrough demonstrating the final Unity frontend app using the developed package.

The following repositories provide access to the final system components:

RecognX Unity Package: <https://github.com/xandervanparys/RecognX-Package>

Final Unity App used for Evaluation: <https://github.com/xandervanparys/Final-eval-frontend>

Backend Service: https://github.com/xandervanparys/RecogX_backend

Initial Testing Frontend (Next.js): https://github.com/xandervanparys/RecogX_frontend

Latency of the system

Due to time constraints, a full system latency measurement could not be conducted. However, since the system's architecture composes pre-tested components (object localization and instruction tracking) and the frontend's UI rendering is lightweight, the overall latency was qualitatively evaluated in the final user evaluation discussed next.

User evaluation

Discussed in section 8.4

6.7.9 Summary

This sprint delivered a fully integrated AR guidance system, bridging backend intelligence with object localization and dynamic task execution in Unity. Through modular architecture, an event-driven design, and a user-friendly API, the RecognX package now offers step-by-step instruction tracking and object highlighting. The system's design prioritizes flexibility and extensibility, ensuring that future developments can build on this solid foundation. Overall, Sprint 6 achieved the transition from a single-functionality Unity package to a functional AR guidance system.

6.8 Future Sprints and Adjustments to Requirements

During the project planning phase, two additional sprints were envisioned—*Sprint 7* and *Sprint 8*—each addressing specific aspects of the system’s future development. However, due to time constraints and prioritization decisions, these sprints were not executed during this master thesis. This section documents the goals of these planned sprints and clarifies how their non-execution impacted the system’s functional and non-functional requirements.

6.8.1 Sprint 7: Apple Vision Pro Version

The idea of creating an Apple Vision Pro version arose from discussions with the MirageXR team, which is actively developing for that platform. The goal of this sprint was to explore how the ARInstructionManager package could be adapted to run on the Apple Vision Pro, leveraging its unique interaction model and spatial computing features. Due to time constraints, only a preliminary exploration was conducted. This revealed challenges such as capturing the user’s view programmatically, as the device does not provide straightforward low-level camera access. Since this was an exploratory goal rather than a formal requirement, no specific requirements were tied to this sprint.

6.8.2 Sprint 8: MirageXR Integration

Sprint 8 aimed to integrate the ARInstructionManager package fully with the MirageXR platform, aligning data formats, API interactions, and user workflows. Although initial discussions with the MirageXR team were positive, the full integration was deferred due to the time required to adapt the system to MirageXR’s architecture and to ensure a stable implementation. Instead, the system was developed as a standalone prototype to allow thorough testing and validation of the core functionalities before committing to a complex integration.

Adjustments to Requirements

The decision to postpone Sprint 8 resulted in several adjustments to the requirements originally defined in Section 4.3. Specifically, certain functional and non-functional requirements related to MirageXR integration were either deferred or removed from the current scope. These adjustments are summarized below.

Functional Requirements Table 6.9 presents the updated functional requirements relevant to MirageXR integration, adjusted to reflect deferred integration and future potential work. For the original requirements, see section 4.3.

Non-Functional Requirements Table 6.10 presents the non-functional requirements that were deferred from the current scope due to time constraints and the postponement of full MirageXR integration.

Table 6.9: Updated Functional Requirements for the Current Prototype

ID	Updated Requirement
FR-4	The system should be designed with a modular architecture and data formats that facilitate future integration with MirageXR's input and data flow.
FR-9	The system should allow easy future integration with MirageXR's task management and storage system.
FR-10	The system's MLLM API calls should be easily replaceable with MirageXR's MLLM calls.
FR-15	The system should be structured to support potential future integration with MirageXR's data structure.

Table 6.10: Non-Functional Requirements Deferred to Future Development

ID	Deferred Requirement
NFR-3	The system must be fully integrated into MirageXR's architecture, using its APIs for input handling, data storage, and AR visualization.
NFR-4	Task progress and state tracking must be stored and managed within MirageXR's database and backend services.
NFR-6	The system must comply with privacy and security standards relevant to MirageXR's deployment.

Documenting these planned sprints and the corresponding requirements adjustments ensures transparency about the project's scope and highlights areas for future development and integration with MirageXR.

6.9 Architectural Evolution and Design Rationale

This section reflects on how the system architecture evolved during development. It highlights key design decisions and changes made to ensure the system aligned with project goals and requirements.

6.9.1 Initial Prototype Approach

The project began with the rapid development of a standalone prototype to support fast iteration on core functionalities—such as instruction tracking and object localization—without being slowed down by the complexity of the MirageXR codebase. The initial plan was to integrate these components directly into MirageXR’s frontend, backend, and data models once they were mature enough.

6.9.2 Shift to an Open-Source Package

After presenting the object localization to the MirageXR team, the approach shifted: instead of integrating everything directly into MirageXR, all AR functionality was transformed into an open-source Unity package that would then be used in their platform. This decision aligned with MirageXR’s modular design philosophy and made the AR features reusable across multiple Unity-based AR applications.

6.9.3 Final Architecture and Evaluation

Due to time constraints, the final integration into the MirageXR platform was not completed during this thesis. As a result, the evaluation of the final functionality was conducted using the standalone prototype system. This architecture, while originally intended as a stepping stone toward integration, ultimately became the primary testing and evaluation environment for the project. The final high-level system architecture is presented in Figure 7.1, and its detailed breakdown is covered in chapter 7.

6.9.4 Other Adaptations

In addition to the packaging decision, the object localization method within Unity evolved. Initially, object positions were determined using depth maps; however, this approach was replaced with a more effective raycasting method that directly interacted with the AR mesh generated by ARFoundation. This change helped improve the accuracy of object localization, resulting in more precise placement of AR labels on the intended objects within the environment.

Most other refinements occurred in the backend component. These changes enhanced its instruction tracking capabilities to support per-step object-aware tasks and improved its data definitions, supporting easier integration with both the MirageXR platform and the Unity package. Key backend improvements included:

- Reorganizing the codebase to adopt a more object-oriented design by incorporating more classes and interfaces, improving semantic alignment between tasks, steps, and physical objects.
- Supporting additional functionalities, such as object extraction from instructions, which allows the system to dynamically show AR overlays for relevant objects at the right time.
- Enhancing the session manager by introducing structured per-user state management, including the storage of a full Task object with object-aware steps, the current step index, and completed steps.
- Implementing robust error handling, ensuring that the system can fail gracefully on GPT errors or other faults, with clear error messages logged.
- Refactoring repeated code snippets into reusable functions within utility classes.
- Iterating prompts sent to GPT based on user testing to refine instruction feedback and ensure higher relevance and accuracy.

These adjustments were driven by iterative feedback and testing, ensuring that the system evolved in line with technical requirements and user needs while maintaining its core functionality and extending its capabilities.

6.9.5 Design Rationale

A dedicated backend server was chosen instead of embedding all processing within the Unity application. This choice was primarily motivated by the desire to simulate the MirageXR platform's architecture, which includes a backend component responsible for storing tasks and managing API calls. By aligning with this architectural pattern, the system supports faster, iterative development of core functionalities, ensuring that integration with MirageXR can proceed smoothly. Centralizing services like GPT-4o instruction tracking and YOLO-based object detection in the backend reduces the computational burden on the AR client, keeping it lightweight and adaptable for deployment on various edge devices with limited processing capabilities. Consequently, the AR client focuses solely on AR functionality and user interaction.

FastAPI was selected for its speed and simplicity, which facilitated easy integration with GPT-4o and YOLO while supporting rapid development of web service endpoints.

Open-source approach: The Unity package was designed as an open-source module, allowing AR functionality to be reused across different Unity applications beyond just MirageXR. This decision aligns with the MirageXR team's request for a portable solution and encourages community collaboration.

JSON database: During development and testing, a lightweight JSON file was used as a database to support rapid iteration and easy adaptation of data formats.

This choice avoided the overhead of setting up a full database system and allowed the team to focus on testing and refining the system's core functionalities.

YOLO was selected for its quick object detection capabilities, offering a balance between speed and accuracy that is beneficial in AR scenarios requiring smooth user experiences. Additionally, YOLO models can be trained reasonably easy on different objects, enabling visual support across a wide range of tasks. For details on the comparative analysis and rationale behind this choice, see subsection 6.2.2.

GPT-4o was selected for its strong multimodal reasoning capabilities, which are essential for analyzing user images in real time and evaluating instruction completion based on the user's context. See subsection 6.3.2 for a comparative analysis of this model choice.

Conclusion

Overall, the architecture was selected and refined to align with the project's evolving requirements. The initial system design aimed for tight integration with the MirageXR platform, but stakeholder feedback and time constraints led to a more modular solution. The final system met the core functional requirements by enabling object localization, instruction tracking, and AR visualization in a responsive and extensible manner. Requirements that depended on deeper MirageXR integration (such as shared data models and platform APIs) were deferred for future work, ensuring that the current prototype remained focused and achievable within the project timeline. This approach balanced rapid development with a clear path to eventual MirageXR integration while fulfilling the most critical functional objectives of the project.

Chapter 7

System Overview

Following the architectural evolution described in the previous chapter, this section provides a technical breakdown of the final system. It details the components and how they interoperate to support instruction tracking and object localization in AR, providing the technical foundation for exploring the main research question outlined in section 1.1

7.1 High-Level Overview

The system architecture consists of a backend server, a data storage component, a modular Unity package, and an AR application that integrates the package (labeled AR Applications in Figure 7.1). Several frontends were developed for testing and validation (see subsection 7.2.3).

The backend server performs the core AI processing tasks, while the Unity package integrates these services to deliver AR functionality. This modular design ensures easy integration with other platforms and supports extensibility for future development.

For a detailed description of the backend services, see subsection 7.2.1.

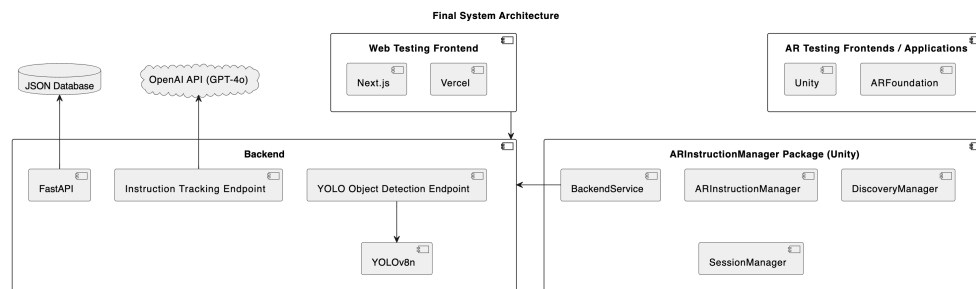


Figure 7.1: High-level system architecture illustrating the modular design and data flow between the backend service, the Unity package, and optional integration with the MirageXR platform.

7.2 Component Breakdown

7.2.1 Backend Server

A dedicated backend server was chosen to align with MirageXR's architecture and support rapid prototyping, enabling faster development of essential features while maintaining consistency with the intended final integration. The backend is implemented using FastAPI and exposes RESTful endpoints that support the following key functionalities:

- **Task Management:** The backend allows clients to create, update, and manage tasks. During task creation, instructions and optional images are processed, and relevant objects are extracted using GPT-4o prompt that tries to map task objects onto YOLO-detectable classes. These tasks are stored using a JSON file as a database.
- **Instruction Tracking:** The backend enables live step tracking by accepting real-time image frames submitted from the AR application. Each frame is processed and encoded before being sent to GPT-4o for evaluation of step completion. The session manager maintains user task history, current step index, and completed steps, ensuring that each user's progress is tracked accurately.
- **Object Detection:** To support AR object localization, the backend processes images using a YOLO detection module. Detected objects, including their class labels, confidence scores, and bounding boxes, are returned to the Unity package for 3D conversion and visualization. The object detection endpoint filters YOLO outputs based on selected class IDs and formats the results for easy integration with the Unity RecognX package.

To support these workflows, the backend architecture integrates several utility modules, including:

- **SessionManager:** Manages task state, including user-specific session data, current step index, and task history.
- **ScenarioLogger:** Logs user progress and key interactions for debugging and analysis.
- **Instruction Utilities:** Provides functions for object extraction and step evaluation using GPT-4o.

The backend logs key metrics including response times, token usage from the OpenAI API and overall system latency, to evaluate performance and ensure the system meets the responsiveness required for user-friendly AR applications. Its modular design, combining task management, instruction tracking, and object detection, ensures a scalable and adaptable system that can be integrated with various frontends and AR environments.

7.2.2 RecognX Unity Package

The open-source RecognX package is designed to integrate seamlessly into any Unity AR application, enabling developers to incorporate object localization, instruction tracking, and dynamic step-by-step guidance. Developers can leverage this package to create customized user experiences with overlays and progress tracking tailored to their AR workflow.

The package's functionality is structured around four key managers:

- **ARInstructionManager:** The main orchestrator, coordinating other managers and managing the flow of an AR instruction session. It is the only exposed entry point of the package, ensuring that applications communicate exclusively through this interface.
- **DiscoveryManager:** Handles object localization by capturing AR camera frames and sending them to the backend for YOLO detection. It processes 2D detections using raycasting into the AR mesh to identify real-world positions.
- **SessionManager:** Keeps track of task progression and object localization.
- **BackendService:** Facilitates all communication with the backend, handling HTTP requests, data serialization, and response parsing. It also ensures that each user's session is maintained on the backend via unique user IDs.

7.2.3 Testing Frontends

Three separate frontends were developed during the project to support different testing phases:

- **Instruction Tracking Testing Frontend:** A Next.js application hosted on Vercel, used to evaluate the instruction tracking and object detection functionalities of the backend independently of the AR components. This frontend (labeled Web Testing Frontend in Figure 7.1) provided a convenient interface for creating and manipulating tasks, sending images and receiving step evaluation feedback from GPT-4o. It also enabled testing of different custom YOLO models and configurations, supporting quick validation and debugging.
- **AR Object Localization Testing Frontend:** A Unity application designed to test object localization functionality in isolation. This frontend allowed for rapid experimentation with YOLO detection and AR mesh raycasting before full system integration (labeled AR Testing Frontends in Figure 7.1).
- **Final Package Testing Frontend:** A Unity application developed to evaluate the combined functionalities of the RecognX package. This frontend facilitated testing during development and is used for the final user evaluation (labeled AR Testing Frontends in Figure 7.1).

Chapter 8

Results

This chapter presents the key results obtained during the development and evaluation of the AR task guidance system.

8.1 Object Detection Performance

8.1.1 Inference speed

Inference speed benchmarks using the COCO dataset confirmed that YOLOv8n achieved real-time performance (17–21 FPS) on Apple Silicon, as documented in Section 6.2.5. This confirmed its suitability for integration into the AR task guidance system.

8.1.2 Custom YOLO Models Performance

Two YOLOv8n models were trained for the object localization component of the system: one for the VR Treadmill task and one for the Gastrosocopy task. As described in Section 5.4.6, no separate held-out test set was created due to limited annotated data and time constraints. Instead, model performance was mainly evaluated with additional qualitative testing conducted in the lab. While this limits conclusions about generalization, the results still provide a useful indication of task-specific detection performance.

Task	Precision	Recall	mAP@50	mAP@50–95
VR Treadmill	0.947	0.817	0.895	0.720
Gastrosocopy	0.907	0.991	0.995	0.845

Table 8.1: Performance of task-specific YOLOv8n models on available training and validation data.

The Gastrosocopy model showed extremely high recall and mAP@50, indicating consistent object detection across dense scenes. The VR Treadmill model exhibited slightly lower recall but maintained strong precision, suggesting reliable de-

tection with fewer false positives. In both cases, model behavior was further confirmed during live testing, where real-time detection proved accurate and responsive enough to support AR-based task guidance.

8.2 Instruction Tracking Evaluation During Development

To iteratively improve system performance, two evaluations were conducted during development: a controlled technical assessment and an exploratory user workshop. As both evaluations directly informed implementation decisions, their full descriptions remain in the Development chapter to avoid repetition.

8.2.1 Controlled Evaluation of Instruction Tracking

A controlled evaluation of instruction tracking, detailed in Section 6.3.5, informed key system design decisions—particularly regarding prompt clarity, image resolution, and backend responsiveness.

8.2.2 User Evaluation: Exploratory Evaluation of Instruction Guidance

To complement the controlled tests, a user workshop was conducted during development and is described in Section 6.3.6. While not a formal usability study, it provided valuable real-world insights into user phrasing, system feedback, and interface expectations.

8.3 OpenAI API Usage

Over the course of development and evaluation, the OpenAI API was used extensively for instruction tracking and object extraction. A total of 429,470 input tokens—spanning image and text prompts—were processed, resulting in a cost of \$1.074. Although the exact number of output tokens is unavailable due to API limitations, the associated cost was \$0.242, bringing the total to \$1.316. A visual summary is shown in Figure 8.1.

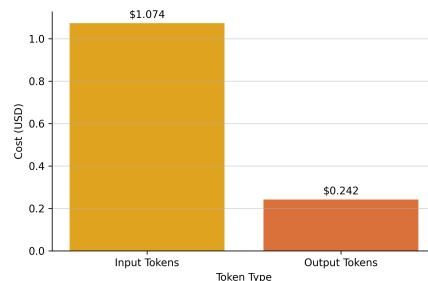


Figure 8.1: Breakdown of OpenAI API usage costs for input and output tokens.

8.4 Final Evaluation

This section presents the results of the final user evaluation, which involved 17 participants completing one of two augmented reality (AR)-guided procedural tasks. The full methodology is described extensively in section 5.4.

8.4.1 Participant Overview

Participants were recruited as a convenience sample from university and SINTEF lab environments and assigned to one of two tasks: VR Treadmill Preparation (IMTEL case) or Gastroscopy Setup (SINTEF case). Each group included one domain expert with prior experience in the respective procedure, while the remaining participants were novices. One domain expert was included in each group to provide expert-level feedback.

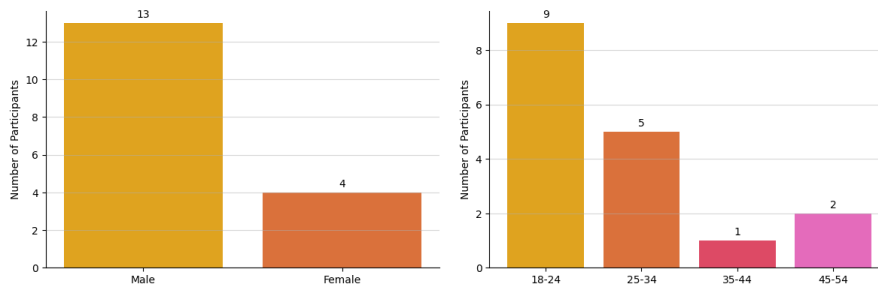


Figure 8.2: Participant demographics: gender (left) and age bracket (right).

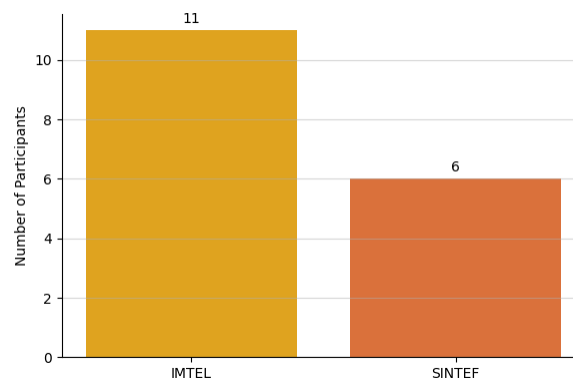


Figure 8.3: Distribution of participants across the two evaluation tasks (IMTEL vs. SINTEF).

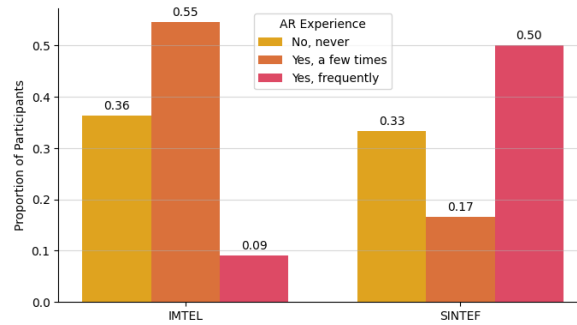


Figure 8.4: Normalized distribution of prior AR experience within each task group.

As shown in Figure 8.2, the participant group was predominantly male, with most participants falling into the 18–24 and 25–34 age brackets.

Figure 8.3 shows the number of participants assigned to each evaluation task. The IMTEL group included more participants than the SINTEF group, although this imbalance was unintentional and based on availability during the evaluation period.

Figure 8.4 shows the normalized distribution of prior AR experience within each task group. The IMTEL group had a relatively higher proportion of participants who had used AR a few times, while half of the SINTEF participants reported frequent AR usage. This variation is coincidental and not the result of targeted sampling.

8.4.2 Questionnaire Results

This subsection presents the results from the post-experiment questionnaire, which aimed to evaluate user perceptions of the system’s helpfulness, clarity, responsiveness, and overall experience. (For details on questionnaire structure, see section F.1.)

Perceived Helpfulness of the System

Participants were asked to rate their agreement with the statement: “*The system was helpful in completing the procedure*” on a 5-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree). As shown in Figure 8.5, responses were consistently positive: 64.7% of participants selected a score of 4, and 35.3% gave it the highest score of 5. No participant rated the system below 4. The mean score was 4.35 (SD = 0.49), indicating that users generally agreed the system was helpful in supporting task execution.

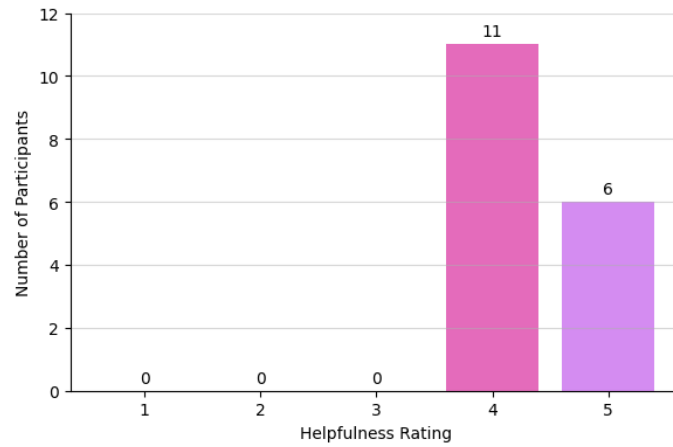


Figure 8.5: Participant responses to the statement: “The system was helpful in completing the procedure.” (1 = Strongly Disagree, 5 = Strongly Agree).

Perceived Helpfulness and Accuracy of Object Localization

Participants evaluated the object localization feature on two dimensions: its helpfulness during the task and its perceived accuracy. As shown in Figure 8.6, responses were consistently positive. A majority of participants (58.8%) **strongly agreed** that the localization feature was helpful, while the remaining 41.2% **agreed**. No participant rated the feature neutral or negative. Regarding accuracy, responses were more spread out: 52.9% **agreed** that the localizations were accurate, 17.6% **strongly agreed**, and 29.4% selected a neutral score. These results suggest that object localization was seen as a helpful feature overall, though there may still be room for improvement in accuracy.

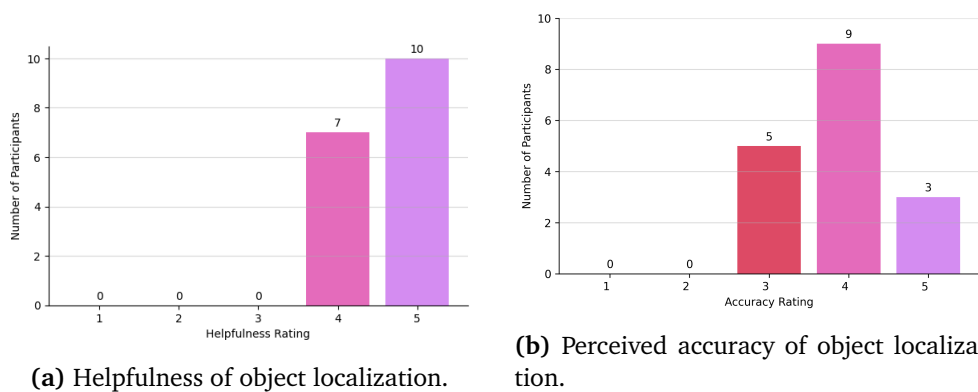
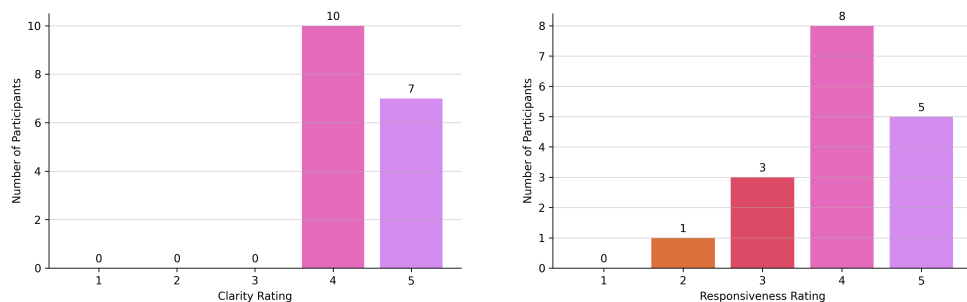


Figure 8.6: Participant perceptions of the object localization feature.

Clarity of Instructions and Timeliness of System Feedback

Participants evaluated both how easy the system instructions were to understand and how promptly the system responded during task execution. As shown in Figure 8.7, 52.9% of participants **agreed** that the instructions were easy to understand, while 41.2% **strongly agreed**. No participant selected neutral or negative options, suggesting that the system's task descriptions were consistently clear and accessible.

Regarding the system's feedback speed, participants responded to the statement: "The system responded quickly enough for me to complete each step without unnecessary delays." A majority (47.1%) **agreed**, 29.4% **strongly agreed**, and another 17.6% were neutral. One domain expert¹ (5.9%) **disagreed**. No participant **strongly disagreed**, indicating generally acceptable responsiveness in most sessions.



(a) Agreement with clarity of system instructions.

(b) Agreement with system responsiveness during the task.

Figure 8.7: Participant ratings of the system's instruction clarity and response timeliness.

Perceived Challenges and Overall Advantage of System Support

Participants were asked whether they experienced any confusion or hesitation due to the system, and whether they believed the system made the procedure easier to complete. As shown in Figure 8.8, most participants (70.6%) reported experiencing confusion **once or twice**, with two participants (23.5%) reporting **no confusion at all**. One domain expert (7.7%) indicated experiencing confusion **multiple times**. These findings suggest that while the system generally provided effective guidance, some usability issues occasionally interrupted task execution. Participants who reported confusion elaborated on specific causes, which revealed several recurring themes:

- **Step completion and progression issues:** Participants sometimes experienced uncertainty about whether a step was completed or which step they were on. This was often due to the system struggling to recognize certain

¹The domain expert was a participant with prior experience in performing the procedure.

objects—especially those with small or subtle features (e.g., green lights on the base station or the powerbox) unless shown from very close range.

- **Instruction clarity:** Occasional ambiguity in how to interpret the task wording.
- **AR tracking behavior:** Some users wanted to re-trigger object localization after moving the camera, disrupting flow.
- **Hardware unfamiliarity:** One participant noted initial confusion stemming from unfamiliarity with VR hardware.

Despite these challenges, participants overwhelmingly felt that the system was helpful. A majority (76.4%) **strongly agreed** that it was easier to complete the procedure with the system's help. Three participants (17.6%) chose **agreed**, and one participant (6%) chose **neutral**. No one reported that the system made the procedure harder. These results highlight that users found the system to be a valuable aid, even if minor usability improvements remain.

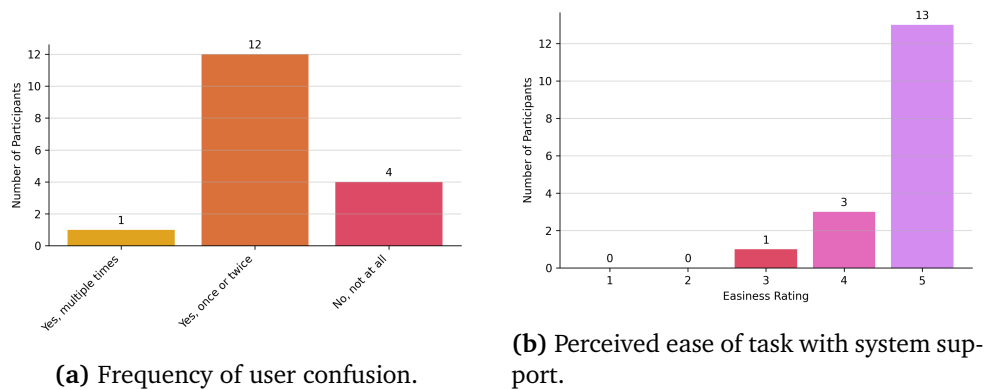


Figure 8.8: Participant responses regarding confusion and perceived helpfulness of the system.

Positive Aspects Highlighted by Participants

Participants were asked what they liked most about the system. The responses revealed several commonly appreciated features:

- **Object localization and arrows:** Many participants praised the object localization capabilities, particularly the use of arrows to guide attention. For example:
 - "The arrows pointing to the necessary object."
 - "The arrow to locate the object"
 - "The location of objects"
 - "The object localisation was accurate and fast"
- **Step-by-step guidance:** The sequential structure of the instructions was seen as helpful and easy to follow:

“Easy to follow step-by-step instructions for completing complicated tasks.”

- **Ease of use and intuitiveness:** Several participants highlighted how simple and intuitive the system felt:

“The operation is easy and just takes 2 buttons during the procedures, that is good.”

- **General enthusiasm for AR:** One participant reflected on the broader potential of using AR for procedural tasks:

“I like the idea of using AR to complete procedures, it seems potentially more helpful than reading an instruction manual or watching a video.”

Overall, participants responded positively to both the functional and experiential aspects of the system, especially its visual guidance and ease of use.

Suggestions for Improvement

Participants were asked what they would improve in the system. The suggestions covered a range of usability and clarity enhancements. Recurring themes included:

- **User interface and visual layout:** Several participants suggested refinements to the interface and layout:

“The layout of the text could be better, there was some overlap of the feedback and the necessary objects.”

“Maybe improve the wording in tasks or increase the font size.”

“The instructions are hard to read on the display.”

“I would maybe improve the HUD and the design.”

- **Object detection and step feedback:** Some felt the feedback timing or detection could be improved:

“The time delay between completing a step and getting feedback was a bit long at times.”

“The highlighting of recognised objects should take less space on the screen, it should be made clearer when you have gotten feedback or completed the step.”

- **Suggestions for added features:** Participants proposed helpful additions such as:

“I would add images/illustrations of components/devices that would help in identifying what objects I should scan.”

“Maybe a mode that could keep Locate Object activated without having to press it again.”

These insights point to minor usability issues that, if addressed, could improve the system’s effectiveness and user experience further.

Summary

In summary, participants responded positively to the system across most measures, particularly highlighting the helpfulness of the object localization and the intuitive nature of the task flow. Nonetheless, several participants offered constructive feedback on interface design and clarity of instructions, providing a basis for further refinement.

8.4.3 Observations during user execution

A research assistant noted participant behaviours, difficulties, and reactions throughout the sessions. The most frequent and noteworthy patterns are summarised below.

Instruction and Interface Clarity

A recurring challenge among participants in both tasks involved understanding the task instructions and navigating the user interface.

Text readability and layout Several participants commented on difficulties with instruction readability. In the SINTEF task, P6 attempted to reposition themselves to improve text contrast, while P8 noted that the instructions were hard to read. P5 found the overlay of instructions on the camera feed obstructive. Similarly, in the IMTEL task, P12 used their finger to create contrast behind the instruction text, reinforcing the need for better visual presentation of textual guidance.

Ambiguity in instructions Participants in both tasks encountered ambiguous or under-specified steps. For example, in the SINTEF task, P7 was unsure where the “entrance to the stomach” was located, while P6 expressed confusion about the term “controller.” In the IMTEL task, multiple users (e.g., P1, P2, and P9) were unsure how many base stations to find, or what the “lever” referred to. These issues suggest that instruction specificity and object naming could be improved to reduce reliance on guesswork or clarification from researchers.

Interaction expectations Participants also displayed uncertainty about how to progress through the task or mark steps as complete. For instance, P4 (SINTEF) was unsure if they should press “step complete” or wait for the system to register it. In the IMTEL task, P11 initially asked whether tasks needed to be remembered or could be completed sequentially.

Localization Performance

Successes and quick detections Several participants reported instances where the system localized objects correctly on the first attempt, which contributed to a

smooth experience. P2 (IMTEL) found the power box immediately, and P6 (SINTEF) confirmed multiple objects with a single scan. P9 found the shoes on the first try. These moments highlight that often, the system's localization pipeline functioned effectively and met user expectations.

Repeated localization and completion attempts Many participants had to scan multiple times before objects were correctly localized. P1 (IMTEL) struggled to localize the power box and required several attempts, even after an arrow pointed in the general direction. P11 and P16 needed four scans to localize the power box precisely. Similarly, P12 needed four scans to mark the base station box step as complete. These cases illustrate the variability in performance and the need for better guidance on scanning angle, distance, or feedback on partial detections.

Mislocalizations and false positives There were several instances where the system localized the wrong object. P2 saw the arrow pointing at the head of a person instead of the VR headset. P13 experienced the same issue, where the system misidentified the assistant's head as the headset. For P16 the base station arrow was pointing to the VR headset in front of it.

Distance and angle effects Some failures in localization were related to distance or viewing angle. P3 was initially too close to the object, and the system failed to detect it until they stepped back. P11 started scanning from far away, which initially failed, but found success when moving closer. These examples suggest that users benefit from scanning guidance—such as extra prompts to adjust distance or framing—when localization fails.

Feedback and Completion Logic

Unclear or delayed completion confirmation Several participants expressed confusion or frustration with the system's feedback mechanisms when confirming task completion. P3 (IMTEL) was unsure whether a step was completed, and only after a second attempt did the system acknowledge it. P13 explicitly asked whether Step 1 was completed, indicating a lack of real-time, visible confirmation. In multiple cases, the system required an extra scan after the action was performed, leading users to question whether their input had been registered.

Participants needing to adapt behavior for confirmation In several situations, participants altered their behavior to satisfy the system's confirmation logic. P11 (IMTEL) turned on the power switch and had to take an extra photo showing the act explicitly for the system to register the step as completed. P13 and P14 received a hint to place a finger on the power button while scanning—this adjustment successfully triggered recognition. These adaptations reveal that the current confirmation logic may rely on subtle visual cues and does not consistently infer completion from context or object state alone.

Feedback messages aiding task progression Despite some inconsistencies, the feedback system did help guide user behavior. In the SINTEF task, multiple participants (P4, P5, P6, P7) received clarifying messages when the step was not yet complete—for example, suggesting the camera controller should be held in the hand or the screen should be turned on. P5 explicitly followed feedback when told to reattempt an incorrect action and completed all steps efficiently after interpreting feedback messages correctly.

Expectations for improved UI integration Some participants expected more visual cues or status indicators in the interface. P1 (IMTEL), experienced ambiguity whether or not feedback was given after scanning, leading them to question whether they had succeeded or needed to retry. These insights indicate a need for more intuitive and consistent step-completion indicators integrated into the interface.

Overall resilience of the system While imperfections in completion logic were noted across many sessions, all participants eventually succeeded in completing all steps, either by retrying or adapting based on feedback. This suggests that the system, even when fallible, provided enough guidance for task recovery, and users were generally able to interpret or experiment their way to success.

User Behavior and Interaction Strategies

Initiative in interpreting and navigating tasks Several participants demonstrated proactive strategies when unsure how to proceed. P11 (IMTEL), for instance, explicitly asked whether they needed to remember all the steps or could proceed incrementally, showing an effort to clarify the workflow before beginning. Multiple participants expressed uncertainty about the power box location but decided independently to scan, reflecting an effective application of trial-and-error within the system's logic.

Manual verification and double-checking Many participants adopted a cautious approach, manually verifying the state of objects or repeating actions to ensure the system acknowledged them. P6 (SINTEF) double-checked the plug selection before scanning, while P13 (IMTEL) reread instructions and resubmitted photos to confirm step completion. These behaviors show an emerging pattern of users compensating for system ambiguity through careful and repeated interaction.

Overreliance on scanning despite manual identification Interestingly, some participants used the system's scanning functionality even after they had already manually found an object. P11, for example, manually identified both the VR headset and controllers but still initiated scans to complete the steps. This suggests a dependency on the system for validation.

Interpretation of abstract terms or vague instructions A few participants struggled with unclear terminology or conceptual steps. P2 (IMTEL) found the “top corner of the room” instruction ambiguous and got a hint from the researcher. P1 did not understand what the “lever” referred to and had to locate it through trial and error. These moments highlight a potential need for more explicit language or visual examples within the instructional system.

Positive Engagement and Perceived Value

Spontaneous positive reactions Some participants expressed enthusiasm or curiosity during the task. P4 (SINTEF) spoke excitedly about the equipment and the procedure, suggesting genuine engagement with the experience. P3 (IMTEL) was positively surprised to see all steps marked as completed at the end, indicating that successful task progression can foster a sense of achievement.

Appreciation for accurate system responses Participants frequently noted when system functionality aligned with their expectations. P12 (IMTEL) praised the accurate localization of the controllers, and P9 remarked positively when shoes were found on the first attempt. These moments likely built trust in the system and contributed to perceived reliability.

Recognition of adaptive feedback Feedback messages that corrected or guided behavior were generally well received. P5 (SINTEF) acknowledged that system feedback helped them correct an initial mistake when the camera was not held in hand. P6 interpreted completion prompts clearly and completed the task smoothly as a result. When clear, such feedback enhanced the instructional quality of the system.

Perception of instructional system as useful Some participants commented directly on the value or usability of the system, even if suggestions for improvement were also given. For example, P12 (IMTEL) indicated that more obvious visual cues would be helpful, but still used the system effectively to complete all steps. This balance of critique and functional engagement suggests a generally positive view of the system’s core value.

Expert vs. Non-Expert Comparison

Task familiarity improves strategy, not immunity to system issues The expert participants—P8 (SINTEF) and P13 (IMTEL)—demonstrated clear advantages in their approach, such as confidently interpreting system feedback, and applying domain-specific knowledge to anticipate task steps. However, their expertise did not shield them entirely from localization or system-related difficulties. P8 initially failed to power on the screen correctly and had to interpret feedback to retry, while P13 encountered mislocalization of the VR headset and had to take multiple scans.

Experts use more deliberate and corrective behaviors P13, for instance, responded to unclear task completion status by rereading the instruction and deliberately including a finger in the scan to aid recognition. P8 effectively interpreted feedback and completed all steps with minimal intervention. These patterns indicate that experts might be better equipped to diagnose and adapt to failures in system recognition.

Efficiency and reduced uncertainty Expert participants tended to complete steps more quickly, with fewer instances of hesitation or scanning loops. For example, P8 completed the entire gastroscopy setup with efficient localization and correct sequencing, while P13 moved smoothly through most IMTEL steps, raising the ring and verifying its position without delay. In contrast, non-experts more frequently paused to interpret feedback, questioned step status, or scanned the same object multiple times.

Implications for system design While expertise clearly contributes to a more fluid experience, the system must still be robust and intuitive enough for novice users to succeed without external help. The fact that even expert users encountered minor breakdowns—such as visual obstructions from arrows or misclassified objects—underscores the importance of minimizing ambiguity and ensuring system feedback is consistently clear, even for those with prior task knowledge.

Summary of Findings

Overall, participants were able to complete both tasks using the AR system, but key issues emerged around instruction clarity and step-verification reliability. Users often struggled with vague terminology, UI ambiguity, and inconsistent feedback, occasionally requiring multiple scans or specific adjustments (e.g., adding a finger) for recognition.

Despite these friction points, many participants responded positively—highlighting moments of accurate localization, helpful feedback, and a general sense of achievement. Expert users progressed efficiently, and even non-experts showed adaptive behavior, confirming the system’s potential for procedural guidance.

8.4.4 Post-Experiment Interviews

To complement the behavioral observations and questionnaire, semi-structured interviews were conducted with all participants after completing the questionnaire (For questions, see Section F1). These interviews aimed to capture participant reflections on system usability, instruction clarity, object localization, and potential for improvement. Thematic analysis was conducted across all responses, grouped according to recurring themes aligned with the interview questions.

Perceived System Understanding

Participants generally reported that the system provided instructions that matched their actions and supported task completion. Most participants indicated that they felt understood by the system, even if moments of uncertainty occurred. For example, P1 noted that although they paused occasionally, they were ultimately “able to complete the task.” Similarly, P6 emphasized that the instructions were “clearly laid out,” and they “never had to guess” what to do next.

However, several participants also highlighted edge cases where the system’s feedback lagged or failed to update as expected. P13 (IMTEL, expert) mentioned that “in rare cases, the steps did not update,” suggesting limitations or bugs in state recognition. P9 also expressed that identifying certain object states, such as whether indicator lights were on or off, required manual camera adjustment, which introduced ambiguity.

A recurring point among several users was the challenge of understanding early steps due to unfamiliarity with the environment. P10 described initial confusion—“I pointed the phone not in the right direction, so the system did not help me”—but later acknowledged that the system helped them orient themselves if they scanned around more. This highlights the dependency of system understanding on user positioning and scanning behavior.

Overall, while participants largely trusted the system’s ability to guide them correctly, their responses reveal that task context, physical orientation, and UI responsiveness played key roles in shaping this perception.

Accuracy and Helpfulness of Object Highlighting

Object highlighting—through arrows and overlays—was consistently recognized as a helpful feature, particularly because most participants were unfamiliar with the task environment. Many described it as crucial for identifying relevant components quickly. For instance, P7 remarked, “I had no idea which object was what. . . it was a nice assurance having the arrow pointing to objects.”

Participants also frequently noted limitations in precision. Several observed that highlights were occasionally misaligned or placed awkwardly. P10 estimated that the highlights were “maybe. . . 87% accurate,” noting slight spatial offsets. Similarly, P11 said that “maybe 10% of them were a bit shifted,” but that repositioning and rescanning often corrected this.

A few participants expressed concern about the arrows occasionally occluding the object itself. P13 (IMTEL, expert) recommended using “smaller or transparent arrows,” as the current overlays could “block the actual object.” Others suggested that the system could improve feedback when an object was not yet detected.

Despite these limitations, many participants stated that they found the object highlighting more effective than static images or video tutorials, particularly in helping them identify and localize unfamiliar objects. P3 and P6 both emphasized how fast and responsive the system was at detecting and highlighting items, describing it as “almost instant” and “great” respectively.

Combination of Instructions and Object Highlighting

Several participants highlighted the importance of synchronizing textual instructions with the visual object highlighting. The experience was often described as clear and efficient. P6 noted, “The combination worked well. I could read, scan, and just follow the arrow.” P12 shared a similar sentiment, appreciating how “the steps were broken down, and the arrows made it easy to follow.” Several participants mentioned that they appreciated the system’s confirmation feedback, particularly in indicating whether they were using the correct object and whether the step had been completed successfully. Several noted this as something often lacking in traditional manuals or video tutorials.

A few participants wished for more explicit integration between the instruction text and the highlighted object. P4 mentioned that while the arrow was helpful, “the instruction didn’t always say what the arrow was pointing to,” which created uncertainty. P11 suggested that confirming visually which object was being highlighted before acting would have improved their confidence in the system. Despite these challenges, most participants appreciated the instructional-object highlighting pairing. P3 described it as a “strong combination,” particularly for complex or unfamiliar steps. These responses suggest that while the system already benefits from coupling instructions with visual guidance, clearer linking and real-time confirmation could further strengthen the experience.

Perceived Value for Learning Procedural Tasks

Participants often commented on the educational value of the AR guidance system for learning procedural tasks, especially in unfamiliar domains. Many described the system as useful, practical, or even preferable compared to traditional manuals or videos. The combination of stepwise instructions and object highlighting was often described as effective and intuitive, especially in guiding users through unfamiliar equipment or setups.

Several participants emphasized that the system enabled them to complete the tasks independently, reducing the need for a human instructor. This self-sufficiency was particularly valued in contexts where users lacked prior knowledge of the physical environment or tools. The system was seen not just as an instructional aid, but as a confidence-enhancing tool, especially for first-time users or novices. An expert participant, P8 (SINTEF), noted the system’s potential for formal training scenarios, such as medical procedures or student exercises, where large numbers of learners need guidance without constant supervision. They also highlighted that AR allows for scalable training, freeing up expert time.

A recurring observation was that the system outperformed static manuals by providing spatial awareness and context-specific feedback. While some users noted the benefit of combining the AR interface with supplementary materials like manuals or posters, most agreed that the interactivity and real-time guidance offered a distinct advantage. P15 and P17 stated that a system like this would be helpful when building IKEA furniture.

Overall, the interviews suggest that the AR system holds potential for procedural learning, especially in environments that demand hands-on interaction with complex equipment. Participants appreciated the immediacy and clarity of instructions and saw the system as a motivating, engaging, and efficient learning tool.

Suggestions for Improvement

Participants offered a variety of constructive suggestions for improving the system, many of which centered around clarity, feedback, and interface responsiveness. A common theme was the desire for more immediate and unambiguous confirmation when a step was completed. Users expressed uncertainty when scanning or performing actions, often due to the lack of clear visual indicators such as checkmarks, animations, or confirmation sounds. Some suggested that a more visible “step completed” notification would reduce hesitation and make the process feel more intuitive.

Another frequently mentioned point concerned the readability and layout of instructions. Participants proposed improving the contrast of the instruction text against the background and avoiding text overlap with other interface elements such as the camera feed or arrows. Dynamic repositioning or background shading for text was recommended by multiple users to improve legibility in varying lighting conditions.

In terms of object localization, users suggested adding real-time feedback to guide scanning — for example, messages instructing the user to move closer, change angle, or rescan if an object was not found. This would help mitigate the frustration associated with repeated failed scans without clear explanations.

Several participants also requested improvements to the visual design of arrows and labels. Arrows were sometimes reported to obstruct visibility or mislead users by pointing imprecisely. Recommendations included shrinking the arrow size, using transparency, or showing outlines rather than filled graphics to reduce occlusion of the target object.

Finally, participants expressed interest in additional features such as a progress overview (e.g., how many steps are left), manual override options when the system wrongly goes to the next step. These suggestions potentially indicate users’ openness to engaging deeply with the system, provided it better supports their need for clarity, responsiveness, and control.

Chapter 9

Discussion

9.0.1 Overview and Research Goals

This thesis set out to explore how state-of-the-art artificial intelligence techniques—specifically MLLMs and vision-based object detection—can be combined in an AR system to guide users through physical procedures. The research was driven by the central question:

How can an AR system be designed to effectively combine instruction tracking (using multimodal language models) and object localization, leveraging vision-based AI models, to support users in performing physical tasks?

To address this, a prototype system was developed that supports real-time instruction tracking and object localization, delivering contextual AR guidance to users. The system’s development followed an iterative, modular process. It was implemented as a standalone application for rapid prototyping.

The research approach combined:

- A technical design and implementation phase focused on fulfilling functional and non-functional requirements;
- An exploratory evaluation phase during development to refine the system through real-world usage;
- A final user evaluation with 17 participants performing two procedural tasks using the AR system, generating qualitative and quantitative feedback.

This chapter reflects on how well the system addressed the original research questions, how users perceived the integrated instruction tracking and object localization features, and what implications these findings have for future design and deployment of AR-based training tools.

9.1 RQ1: Instruction Tracking with Multimodal Language Models

9.1.1 System Design and Implementation

When a user thought to have completed a step he could send an egocentric image to the system. This then got sent to GPT-4o along with the full task description, current step, and any previous step context. The model was asked to evaluate whether the current step had been completed or not. If the step was completed, GPT was prompted to return the next step; if not, it was instructed to provide feedback to help the user progress.

9.1.2 Step Tracking Accuracy

Several participants appreciated the system's confirmation feedback, especially in verifying whether they were using the correct object and had completed a step successfully. This kind of real-time feedback was viewed as a distinct advantage over traditional training methods like manuals or video tutorials. These findings align with the system's goals of step validation (FR-7) and adaptability to diverse instruction complexities (FR-8).

Participants were often pleasantly surprised by the system's ability to correctly detect step completion. However, the reliability of this was highly dependent on how explicitly the completion conditions were defined. When instructions were vague, hallucinations or incorrect judgments occurred. For instance, during the SINTEF task, one participant picked up the camera controller as instructed. In the next step, they needed to insert the probing camera into a model stomach. Since they had to hold their phone in one hand, the participant briefly set down the controller to use the camera. GPT then failed to recognize the step as complete, because the controller was no longer in their hand, even though the participant had followed the instructions correctly.

In contrast, for the instruction "put on the platform shoes," GPT marked the step as complete even though only one shoe had been worn. A similar tendency to overpraise learner actions was observed in the VTA system [28], where GPT occasionally classified subtly incorrect gestures as "perfect" due to overly lenient thresholds. These examples show the MLLMs' tendency to either oversimplify or apply overly literal interpretation, depending on the situation. They reaffirm the importance of verbose, unambiguous step descriptions and highlight the limitations of current MLLM-based visual understanding. Similar challenges were noted in XaiR [6], where GPT-4V occasionally failed to correctly identify whether a step was complete unless the object was clearly visible and unambiguous in context.

9.1.3 Context Awareness and Visual Understanding

A particularly recurring issue arose in the VR treadmill task. Participants were instructed to verify that a VR base station was active by capturing an image of its

green LED light. Although they could easily see the light themselves, GPT often failed to recognize it—likely due to resolution or lighting limitations. Many users had to retake photos multiple times before realizing they needed a close-up. While using higher-resolution images might resolve this, it would also increase latency, which could negatively impact responsiveness (NFR-2).

Another noteworthy event involved a participant intentionally trying to deceive the system by placing the VR headset on the back of their head (see Figure 9.1). GPT correctly identified that the headset was not worn properly and responded with “Please ensure the VR headset is properly positioned over your eyes, not on your forehead.” While this was an impressive catch, the response also revealed a limitation: GPT mistakenly assumed the headset was on the forehead. Given the lighting and visual ambiguity (e.g., similar shirt and headset colors), the confusion is understandable. This demonstrates both the power and fragility of GPT’s visual reasoning. A few participants echoed this observation in interviews—acknowledging the helpfulness of the system while noting that it occasionally struggled to understand scenes the way humans do. Similar issues with visual ambiguity were also observed in TAGGAR [27], particularly when objects in the scene had repetitive or confusing elements, such as multiple identical drawers or switches.



Figure 9.1: Attempt to fool GPT by placing the VR headset incorrectly on the back of the head

9.1.4 Feedback Consistency and Prompt Reliability

A particularly strong outcome was GPT’s consistent adherence to the structured response format. The system relied on compact prompt templates with strict response expectations. During the final evaluation, GPT failed to return a valid response only a few times (around three cases across all tasks and participants), far exceeding expectations. This reliability facilitated seamless integration with the step tracking logic and underscores the effectiveness of the prompt design (FR-6 and FR-9). Similar strengths in prompt-driven reliability were reported in the VTA

message training system [28], where JSON-formatted outputs allowed consistent and actionable feedback for both learners and instructors.

9.1.5 Feedback Quality

Participants consistently reported that the system's feedback was helpful and often played a meaningful role in guiding task execution. In several cases, the feedback was not only responsive but also demonstrated a degree of contextual awareness. For example, during the IMTEL task, users were asked to verify that a device was active by capturing an image of its green LED light. When GPT initially failed to detect the light, it responded with clarifying guidance such as: "Please ensure the green light is clearly visible in the image." This indicated that the model was capable of inferring what it lacked from the input and communicating that to the user, even in the absence of successful recognition. Over time, many participants learned that successful step validation often depended on image framing—for instance, ensuring that a button press was explicitly visible.

These examples reflect a promising level of visual reasoning and adaptive interaction. However, they also reveal a key limitation: such interactions are not always intuitive. The responsibility to correctly interpret the system's expectations remained with the user. This was particularly evident in cases where users believed a step had been correctly completed but GPT failed to recognize it. Several participants suggested improvements, such as introducing an option to override or skip a step when confident in its completion. Such functionality could significantly improve usability and trust, especially in low-risk environments.

While GPT occasionally delivered rich and actionable feedback, most initial responses tended to simply restate the original instruction. However, after repeated recognition failures, the model often adjusted its feedback, offering more detailed suggestions—such as specific object orientations or elements to highlight in the frame. This adaptability was positively received, with several participants noting that it made the system feel more interactive and supportive, rather than rigid or passive.

The quality of GPT's feedback was closely tied to the clarity and specificity of the step definitions used in the prompts. When these definitions clearly described the criteria for step completion, GPT was able to enforce them effectively and guide users appropriately. In contrast, vague or underspecified steps led to confusion, occasional hallucinations, or overly rigid interpretations. These findings underscore the importance of well-structured prompt design and highlight both the potential and the current limitations of using MLLMs as feedback engines in procedural systems.

Despite occasional inconsistencies or misunderstandings, the feedback mechanism enabled users to complete tasks with minimal external support. With improvements such as clearer prompts, multimodal clarification options, or the ability to override steps, future systems could further enhance the user experience. Overall, the results affirm that GPT-4o can serve as a capable feedback provider in

AR-based procedural training, provided its outputs are scaffolded with thoughtful task design and user-centric interaction strategies.

9.1.6 Responsiveness and Latency Tolerance

In an early development latency test (see section 6.3.5), the instruction tracking feature achieved an average response time of 2.63 seconds. This is already a good improvement from the latency reported in XaiR [6], where end-to-end response times exceeded 4 seconds. Despite not being instant, this latency was generally well tolerated: 76.5% of participants agreed or strongly agreed that they did not experience unnecessary delays when using the system (Figure 8.7b).

However, response times are still subject to external factors, particularly the load on OpenAI's servers. While internet speed contributes to some variability, early development tests—conducted under average modern network conditions—showed that backend processing and network transmission accounted for only 8.1% of the total latency. This suggests that the bulk of the delay stems from GPT-4o's internal processing and response generation time. This may explain why 17.7% of participants gave a neutral rating for responsiveness.

These results mirror broader challenges reported in systems like XaiR [6], where reliance on remote MLLMs introduces latency that affects real-time interactivity. Nonetheless, continued improvements in model efficiency and inference speed—such as those seen in the shift from GPT-4V to GPT-4o—are likely to further reduce delays in future systems.

Only one domain expert expressed dissatisfaction. However, this appeared to stem more from the added overhead of using the system compared to completing the task unaided, rather than from actual delays in GPT-4o's responses. This was confirmed during the semi-structured interview following task completion. Given the expert's prior familiarity with the procedure, the system introduced additional steps and occasional confusion, which would not affect novice users in the same way. Still, this highlights the importance of improving the intuitiveness of the interface for users.

During development and testing in the lab, it became clear that simply displaying a loading popup while the API call was in progress significantly improved the user experience. Without it, users would probably become impatient and repeatedly press buttons. This again confirms that shorter response times are desirable, but UI feedback mechanisms can help maintain a sense of fluidity even when processing is delayed.

However, if the system were to be deployed at scale—with many users interacting simultaneously—backend throughput and load on OpenAI's servers could introduce additional delays. The backend would definitely need improvements such as request queuing, load balancing and many more robustness infrastructure in order to maintain responsiveness in high-demand scenarios. OpenAI's backend is already used to enormous loads and will probably continue to handle that better and better. During the course of this project I never noticed any significant devi-

ations from the average response times, indicating that OpenAI's servers are very reliable.

9.1.7 Usage Cost

As detailed in section 8.3, the system processed a relatively high volume of image and prompt queries during development and evaluation. Despite this, the total cost remained low (\$1.32), demonstrating the cost-efficiency of using advanced MLLM APIs such as GPT. This supports their feasibility not only in research prototypes but also in scalable, commercial applications.

The low cost—combined with the high-quality feedback reported by users—suggests that the prompt structure was efficient in both design and token usage. These results highlight the potential of MLLMs in interactive AR systems with minimal operational expense.

Conclusion

Overall, the findings show that MLLM-based instruction tracking holds significant promise. Its accuracy, adaptability, and ability to provide task-specific feedback impressed many participants. However, its effectiveness remains tied to step clarity, visual context, and input framing. Further improvements in task design, user input options, or model robustness could significantly enhance performance. As a prototype, the system already highlights both the potential and the current constraints of integrating MLLMs into procedural AR applications.

9.2 RQ2: Object Localization in AR using Vision-Based Models

9.2.1 System Design and Implementation

When users can't find an object, they can scan the scene. A YOLO model then detects only the relevant objects for the current step and returns bounding boxes. In Unity, the center of each bounding box is used to raycast into the LiDAR-generated environment mesh. The point where the ray hits the mesh is taken as the estimated 3D location. A red arrow and the object name are then displayed at this position.

9.2.2 User Perception and Feature Integration

The evaluation showed that all participants appreciated the object localization feature. It was seen as useful not only for identifying unfamiliar objects, but also as a confirmation mechanism when users were unsure. Several participants commented on the speed of the system and were surprised at how well it functioned. These findings align with results from García et al. [20], who found that AR overlays in the Holorailway system significantly improved spatial accuracy, reduced errors,

and enabled even inexperienced users to complete assembly tasks efficiently. Like in our system, visual AR cues were especially appreciated for their role in confirming actions and reducing uncertainty. The final user evaluation of this thesis also showed that the majority of participants found the AR overlays to be accurate. This confirms the successful integration of FR-1, FR-2, FR-3, and FR-12, and indicates that NFR-1 (accuracy) and NFR-2 (speed) were fulfilled.

These results support the conclusion that fast and accurate computer vision models like YOLO can be valuable tools in AR applications where real-time environment awareness is required. These results align with findings from TAGGAR [27], a system that also uses GPT-4V for semantic parsing and vision-language alignment. However, while TAGGAR uses GroundingDINO for object localization, our system combines GPT-4V with a YOLO-based model to prioritize real-time performance. Participants in both studies appreciated the ability to receive visual guidance without prior task knowledge. In our case, the faster inference times of YOLO enabled immediate feedback during exploration, and multiple users explicitly commented on the system’s responsiveness. This suggests that pairing vision-language reasoning with a fast object detection backbone is an effective strategy for instruction tracking in real-world AR use cases. Similarly, the XaiR system [6] demonstrated that users benefited from overlaid AR annotations during task execution, particularly when generated from expert walkthroughs. While XaiR focused on reconstructing instructions from egocentric video and relied on server-side models like Ferret for object localization, our approach emphasizes responsiveness using client-side YOLO inference.

9.2.3 Challenges and Limitations

Despite its effectiveness, several limitations were observed. To create the final two evaluation tasks, small custom datasets had to be annotated and used to fine-tune YOLO models for detecting relevant objects. While the datasets were kept minimal to allow for rapid annotation and training, this approach introduced constraints: the models only recognized objects from specific angles or under certain lighting conditions. As a result, objects were occasionally missed during use.

This highlights the broader challenge of dataset preparation: improving robustness often requires large, diverse datasets, which demand significant annotation effort and training time.

Another challenge arose when the system needed to distinguish between multiple identical or similar-looking objects positioned close together. Since object detection models like YOLO return bounding boxes without contextual differentiation, it was impossible to specify which instance was correct. In the SINTEF task, for example, I had to physically cover a smaller, similar-looking socket with paper to prevent the system from selecting it instead of the intended one.

Similar challenges were reported in TAGGAR [27] and XaiR [6], particularly regarding spatial accuracy and detection robustness. TAGGAR occasionally produced ambiguous results when objects had multiple similar parts (e.g., identical

drawers or switches), while XaiR noted inaccuracies in AR content placement due to incomplete or slowly updating (1Hz) environment meshes—especially during the initial moments of use. All systems using raycasting shared a common limitation in projecting 2D detections into 3D space: occlusion, unstable meshes, or missing depth data often led to inaccurate anchoring. Furthermore, like XaiR, our projections are fixed after placement and not actively tracked, which can result in misalignment if the scene or objects change.

Another limitation involved object occlusion. When users were unsure what an object looked like, they often scanned the room multiple times until a red arrow eventually appeared. In some cases, the object was occluded or only partially visible, preventing detection. This led to trial-and-error behavior, particularly when the model failed to return results due to viewing angles or partial visibility.

Similar issues were addressed in the ELLMER system by Mon-Williams et al. [25], which also tackles object localization—but in a robotics setting instead of AR. Their setup combines GPT-4 with vision models and even force feedback to help a robot complete tasks. While their system is obviously more complex (using things like an Azure Kinect and Grounding DINO to create a voxel-based 3D view of the scene), they ran into some of the same problems we did. For example, when objects were heavily occluded—like 80–90% blocked—their detection success dropped to about 20%. It's clear that occlusion and viewing angles are big challenges for any object localization system, whether it's in AR or robotics.

An experimental auto-scanning feature was introduced during development to address this. It continuously performed object detection as users moved their phone around. Although this allowed more fluid exploration, the feature was excluded from the final evaluation due to poor positional accuracy. This was likely caused by the delay in Unity's environment mesh updates, which could not keep up with rapid motion. While manual framing allows time for the mesh to settle, dynamic scanning often resulted in outdated or misaligned raycast collisions.

Even manual picture-taking sometimes suffered from mesh limitations. If an object was partially visible at the edge of the image, the object detection model would still create a bounding box, and the system would raycast through it. This often produced inaccurate positions. This may be due to the iPhone 13 Pro's LiDAR sensor, whose depth accuracy diminishes toward the edges of the field of view. Denser meshes are generated in the center, making peripheral detections more error-prone.

9.2.4 Proposed Improvements

Future systems could benefit from several enhancements:

- **Higher-end hardware:** Devices like the Apple Vision Pro may provide denser and more accurate LiDAR meshes across the entire frame, reducing spatial inaccuracies.
- **Partial detection handling:** Strategies to ignore or flag partially visible detections could prevent misleading feedback, though this may also suppress

valid results and frustrate users.

- **Preloading object positions:** Instead of discarding detections of future-step objects if not relevant for the current step, the system could store their positions silently and reveal them later, as is already done for objects in the current step that are also referenced again in a later step. This would reduce the amount of necessary scanning.
- **Generalist vision-language models:** Studies such as XAIr [6] have shown that models like Apple Ferret can perform object localization directly from natural language prompts, without requiring task-specific training or manual dataset creation. However, these models currently have slower inference times, making them less suitable for real-time applications. This direction also aligns with TAGGAR [27], which uses GroundingDINO for open-vocabulary detection and proposes integrating it with GPT-4V to streamline semantic grounding. In contrast, our system relies on fast, task-specific YOLO models, offering better responsiveness at the cost of flexibility. Future work could explore balancing these trade-offs by integrating more efficient generalist models.

Conclusion

In summary, the object localization system proved to be a valuable feature, effectively assisting users in identifying task-relevant items with surprising speed and accuracy. Its integration demonstrated the feasibility of combining fast computer vision models like YOLO with AR mesh-based spatial reasoning. Nonetheless, several limitations emerged, particularly related to dataset dependency, partial occlusion, and positional accuracy tied to LiDAR resolution and mesh delays. These findings underscore the need for richer training data or generalist object detectors, better occlusion handling, and potentially more advanced AR hardware. Future work could explore the integration of generalist vision-language models and continuous scanning techniques to create more robust, hands-free, and user-friendly object localization experiences in procedural AR systems.

9.3 RQ3: Integration and Impact on Procedural Learning

Due to time constraints, this project did not include a dedicated training phase or formal measurement of user performance improvements over time. No longitudinal data were collected, and no controlled comparison was made between users trained with this system and those using traditional methods such as manuals or video tutorials. As a result, it is not possible to draw empirical conclusions about training outcomes. Nonetheless, insights from the final user evaluation offer meaningful indications of how the integration of instruction tracking and object localization may impact procedural learning in AR environments.

Participants were asked to complete a full procedural task using the integrated system, which combined instruction tracking via GPT-4o with object localization

based on a YOLO model and LiDAR-generated environment mesh. Despite the prototype nature of the implementation, all participants completed the tasks with minimal external assistance, highlighting the system's usability and functional reliability even for first-time users. This aligns with prior research on AR-guided procedural training systems such as WEKIT, where step-by-step overlays enabled domain-independent learning across different domains [22].

Quantitative results support this impression. As shown in Figure 8.5, the system received consistently high ratings for helpfulness, with no participant rating it below 4 on a 5-point Likert scale. The mean score was 4.35 (SD = 0.49), indicating that the system successfully supported task execution for most users. This reflects earlier findings by Richardson et al., where in-situ AR overlays reduced errors and improved task efficiency in technical assembly tasks [21]. While WEKIT did not observe statistically significant learning gains over traditional methods, it confirmed that AR systems can deliver effective and clear instruction without impeding learning [22].

Object localization was similarly well received. A majority of participants (58.8%) **strongly agreed** that the localization feature was helpful, while the remaining 41.2% **agreed** (Figure 8.6a). Perceived accuracy, while slightly more varied, was still positive overall, with 70.5% of participants agreeing or strongly agreeing that the localization results were accurate (Figure 8.6b). These findings are consistent with the WEKIT system's success in helping users localize physical tools and objects via AR markers [22], and with the high visual grounding accuracy reported in XaiR's use of Ferret and 3D raycasting for spatial overlays [6].

Participants also reported that the system's real-time feedback helped them feel more confident during task execution. As one user put it, the system provided a "sense of progress" by actively confirming completed steps and guiding users to the next ones. This aligns with results from the multimodal massage instruction study, where GPT-generated, step-specific feedback improved learners' confidence and execution quality [28]. While that system relied on motion and pressure data, the feedback function served a similar role to this project's GPT-based confirmation prompts—underscoring the value of timely and personalized guidance during procedural tasks.

Crucially, participants emphasized the value of the system's visual guidance features. The red arrows pointing to relevant objects helped reduce time spent searching or guessing, and many participants mentioned that the combination of visual and verbal cues clarified not just what to do, but also where and how to do it. This combination of object localization and instruction clarity resembles the dual-channel support found in XaiR, where 3D AR overlays and step-specific prompts were used to guide users through tasks [6]. While 70.6% of participants reported experiencing confusion at least once (Figure 8.8a), this was typically due to minor usability issues—such as unclear step transitions, difficulty recognizing small objects like LEDs, or the need to actually show your finger on a switch rather than it just being flipped on. Similar issues of perceptual ambiguity and step misclassification were noted in XaiR, where users sometimes had to repeat actions or repos-

ition objects to clarify intent for the MLLM system [6]. This highlights a broader challenge in integrating visual understanding with instruction logic—a limitation shared by recent systems.

Despite these challenges, 94.1% of participants agreed or strongly agreed that the system made the procedure easier to complete (Figure 8.8b). Taken together, these results suggest that the integrated system successfully supported procedural execution and fostered an interactive, confidence-boosting learning experience. Although no formal training metrics were collected, the low number of clarification requests, 100% task completion rate, and consistent engagement with system feedback indicate that the system may support procedural learning in real time. This aligns with previous findings that real-time AR guidance can reduce cognitive load and enhance user engagement [21, 28].

In summary, while this project did not conduct a formal study of training outcomes, the user evaluation results demonstrate the feasibility and value of integrating instruction tracking with object localization in AR-based procedural tasks. Participants responded positively to both the functional capabilities and the experiential aspects of the system. These findings point to strong potential for future training applications—particularly when paired with more rigorous evaluation methods and extended user exposure. The cross-domain effectiveness observed in WEKIT [22] and the general-purpose applicability demonstrated in recent MLLM-based systems [6, 28] support the idea that such systems can evolve into robust, domain-flexible platforms for procedural training.

9.3.1 UI Feedback and Usability Observations

The most frequent critiques during the user evaluation related to the user interface—specifically the clarity of the visual layout, text readability, and overall intuitiveness. These issues occasionally disrupted the flow of the task, such as when instruction text visually overlapped with feedback or when users struggled to read small fonts.

Interestingly, these were the dominant sources of friction, rather than complaints about the core functionality. This suggests that the instruction tracking and object localization features were well accepted, even with a minimal UI layer. It also points to a clear opportunity: improving the UI could significantly enhance user experience without major architectural changes.

Several participants also expressed a need for clearer guidance during use. For example, some were unsure how to frame objects to trigger detection or how to confirm that a step was completed. Including subtle onboarding messages, tool-tips, or visual hints (e.g., framing suggestions or camera angle tips) could prevent such confusion and reduce hesitation during first-time use.

9.3.2 Implications for Future AR Training Tools

This system demonstrated that integrating instruction tracking and object localization can enable autonomous task completion in AR-based learning scenarios.

However, the evaluation also surfaced limitations that should inform future developments.

Multimodal LLMs such as GPT-4o were effective at contextual tracking if carefully prompted but they remain prone to hallucinations and can produce ambiguous or misleading outputs. Their response times have improved but are still not instantaneous, which can interrupt the flow of real-time guidance.

YOLO-based detectors enabled real-time localization but required custom training to work reliably for each task. This limits scalability. Generalist detectors like GroundingDINO or Apple Ferret are promising for broader use, but their slower inference makes them less suitable for current mobile AR experiences.

Raycasting-based 3D localization worked well under ideal conditions, but mesh update delays, occlusions, and peripheral detections often resulted in inaccurate placements. Higher-end hardware (e.g., Apple Vision Pro) and smarter mesh sampling may mitigate these issues.

Overall, the key lesson from the evaluation is that technical performance was not the main barrier—rather, small UX and guidance improvements would have made the system significantly more intuitive and effective. Future tools should definitely enhance these aspects as well. This would make future AR training tools more accessible, intuitive, and robust for real-world use.

9.4 Limitations

9.4.1 Project Timeline and Scope

One of the primary limitations of this study was the highly constrained timeline. The entire project—ranging from initial background research and related work analysis to system design, implementation, iterative testing, user evaluation, and thesis writing—was conducted between February and mid-June. This short time frame necessitated a strong focus on creating a functioning prototype, often at the expense of deeper empirical testing or extended validation of individual system components.

As a result, many design decisions were evaluated based on informal observations during development rather than rigorous, quantifiable metrics. While this iterative and exploratory approach provided valuable insights, it limited the generalizability of certain findings.

Due to time constraints, some technical avenues could not be fully explored. For instance, the integration of Apple Ferret—a general-purpose vision-language model capable of open-vocabulary object localization—was initially planned as a more flexible alternative to task-specific YOLO models. However, setting up and deploying Ferret proved non-trivial, and limited development time meant this direction had to be abandoned. As a result, the final system remained reliant on manually trained, task-specific detectors, limiting scalability and adaptability to new scenarios.

Similarly, an experimental auto-scan feature was developed to continuously de-

tect objects as users moved through the environment. While early tests showed promise, there was insufficient time to optimize mesh synchronization or evaluate its real-world performance in depth. Consequently, this feature was excluded from the final evaluation. This omission may have impacted user experience, as several participants explicitly expressed a desire for a more dynamic, passive scanning approach to object localization. A longer development period could have allowed the feature to mature and potentially improved system usability and intuitiveness. Moreover, due to time pressure, some aspects of the codebase remained under-documented or less refined than desired. A more structured development phase with dedicated time for refactoring and documentation would be desirable for future iterations.

Finally, although the system was designed to be integrated with the MirageXR platform and to support devices such as the Apple Vision Pro, these goals were not realized within the project timeline. Such integration would likely have enhanced usability—particularly by enabling hands-free operation and access to more robust LiDAR sensing and spatial computing capabilities.

9.4.2 Final User Evaluation

The final user evaluation, while yielding useful insights, also presented several limitations. First, participants were recruited through convenience sampling, resulting in a group that—although varied in terms of prior AR experience—was relatively uniform in age and likely shared a high degree of digital familiarity. This limits the generalizability of findings to broader user populations. For example, the SINTEF task simulated a medical training scenario (gastroscopy), but no medical students or healthcare trainees participated in the evaluation.

Second, the number of participants was relatively small ($n = 17$), which limits statistical power and the ability to generalize findings. Also, there was an imbalance in task assignment: more participants completed the IMTEL (VR treadmill) task than the SINTEF (gastroscopy) task. This uneven distribution was due to availability constraints but may have introduced bias, reducing the robustness of cross-task comparisons. Moreover, no control group using traditional training materials (e.g., manuals or video tutorials) was included. This limits the ability to draw comparative conclusions about the system's effectiveness relative to baseline approaches.

Third, while the evaluation tasks were designed to simulate realistic procedures, it became apparent during testing—particularly for the SINTEF task—that they may have been too simple to fully showcase the system's benefits. In fact, some participants noted they could have completed the procedure even without assistance. In future studies, using more complex, domain-specific tasks with unfamiliar or specialized equipment could better reveal the added value of the system in real-world training scenarios. Furthermore, as participants only interacted with the system once, no conclusions can be drawn about long-term learning effects, retention, or improvements over repeated usage.

Fourth, the object localization feature was powered by YOLO models fine-tuned on very small datasets. While this enabled fast prototyping, the sparsity of training data likely contributed to occasional detection failures and reduced the model's robustness across lighting conditions or camera angles. A more rigorous data collection and annotation process would likely have yielded improved performance. Finally, the system was tested using a smartphone (iPhone 13 Pro Max), requiring participants to hold the device while performing physical tasks. This two-handed interaction burden was consistently noted as a usability issue, especially during more intricate steps. Running the system on head-mounted devices such as the Apple Vision Pro would not only improve comfort and intuitiveness but would most likely improve performance due to a better LiDAR sensor and spatial mapping capabilities.

Conclusion

While these limitations impacted the breadth and generalizability of the study, they are typical of early-stage prototyping and system exploration. The lessons learned nonetheless provide a strong foundation for future work, particularly in integrating broader participant sampling and extended evaluation. Future studies should aim for more controlled comparisons, domain-specific user testing, and longitudinal follow-ups to fully assess the educational impact of instruction tracking and object localization in augmented reality systems.

9.5 Contribution to Sustainable Development Goals

This thesis contributes to several of the United Nations Sustainable Development Goals (SDGs) by advancing the responsible use of digital technologies to support learning, accessibility, and innovation.

SDG 4 – Quality Education

The core objective of this research is to enhance **instruction-based learning** using **AI-powered assistance in AR environments**. By providing real-time, context-aware guidance and visual feedback, the system supports active learning in procedural and skill-based domains. This aligns with SDG 4, which emphasizes the importance of “increasing the number of youth and adults who have relevant skills, including technical and vocational skills, for employment, decent jobs and entrepreneurship”.

SDG 9 – Industry, Innovation and Infrastructure

This work explores the integration of cutting-edge technologies — including **vision-language models (GPT-4o)** and **real-time object detection (YOLOv8)** — in a modular system architecture. The resulting Unity-based package promotes

modularity and scalability, contributing to **sustainable digital infrastructure for future AR learning systems**. By open-sourcing the core components, the project also promotes knowledge sharing.

SDG 12 – Responsible Consumption and Production

Although primarily software-based, the project also considers **sustainable computing practices**. Object detection models were trained and benchmarked on **energy-efficient Apple M-series chips**, helping reduce the need for high-power GPUs. Additionally, the use of large language models (LLMs) was carefully monitored and bounded by a **token usage cap** during development and evaluation. This constraint encouraged more efficient prompt engineering and reduced unnecessary model calls, contributing to more sustainable and cost-effective use of AI resources.

In future iterations, further attention could be given to reducing the carbon footprint of LLM interactions.

SDG 17 – Partnerships for the Goals

The project was developed in collaboration with a **university spin-off team** (MirageXR), illustrating a partnership between academic research and applied development in learning technologies. Although the project was led at the Norwegian University of Science and Technology (NTNU), it was carried out **during an Erasmus exchange from Ghent University (UGent)**, reflecting meaningful international academic cooperation. This underscores the importance of cross-border collaboration and joint research efforts in developing scalable educational technologies that can be adapted across contexts and institutions.

9.6 Data Privacy and Security

Since the prototype was tested in a closed setting without user accounts or external deployment, dedicated data privacy, security, and user management features were not required for this phase. The system was tested solely by the developer and participants in a supervised lab setting. During the user evaluation, instruction tracking data was collected for evaluation purposes, which occasionally included photos taken by participants. No personally identifiable photos were included in the thesis or shared externally without consent, and all data handling adhered to ethical research practices. Future integration with MirageXR would leverage its existing infrastructure to address these concerns, ensuring secure storage, privacy compliance, and proper user management aligned with the platform's standards.

Chapter 10

Conclusion

This thesis set out to explore how an AR system could effectively integrate instruction tracking using MLLMs with object localization through vision-based AI models to support users in performing physical tasks.

To investigate this, a prototype system was iteratively developed and evaluated through a series of design sprints, culminating in a final user study. The analysis was structured around three subquestions, each addressing a key system component. Together, the findings provide insights into the feasibility, implementation, and perceived impact of combining instruction tracking and object localization in AR-based procedural support.

10.1 RQ1: MLLMs for Task Guidance

How can MLLMs be leveraged to track and guide users through multi-step tasks using egocentric visual input and structured prompts?

The proposed and implemented system design together with the research findings demonstrated how MLLMs can be used for real-time instruction tracking by evaluating egocentric images against structured, step-specific prompts. The system used GPT-4o, which was aware of the full procedure and assessed whether each step had been completed based on user-submitted visual input. It then either confirmed completion and advanced to the next step, or provided targeted feedback on what was still missing.

Participant behavior and feedback confirmed that MLLMs can successfully interpret egocentric images and structured prompts to track multi-step tasks. Response times were generally fast enough to support real-time interaction without disrupting the user experience. Although occasional inconsistencies in recognition were observed—likely due to known hallucination issues with LLMs—users often received helpful, context-aware feedback when a step was not recognized. In several cases, the system’s ability to infer what was missing enabled corrective prompts that helped users resolve issues without external assistance.

While the evaluation tasks were not complex enough to fully test the limits of

MLLM reliability in high-stakes or ambiguous tasks, the results strongly suggest that they hold promise. Further research and development into leveraging MLLMs for task support and learning in AR environments is clearly worthwhile—especially as these models continue to improve in spatial and visual reasoning.

These results confirm the feasibility of using MLLMs for real-time step verification and guidance in procedural AR systems, particularly when supported by well-structured prompts and clearly defined steps.

10.2 RQ2: Vision-based AI for AR Feedback

How can a robust object localization pipeline be designed and implemented in AR systems using vision-based AI models and AR devices to provide visual feedback during task execution?

This thesis presents a robust object localization pipeline that combines vision-based AI, spatial mapping, and LLM reasoning to support users during physical tasks in AR. At its core, the system used GPT-4o to process task instructions and extract relevant objects for each step. Since instructional language often doesn't match YOLO's class labels, GPT-4o served as a semantic bridge by translating natural-language references into structured label lists. These were stored step-by-step and used by the backend to guide object detection dynamically throughout the task.

For localization, user-submitted images were sent to the backend where a YOLO model identified instances of the relevant objects. The 2D center of each detection was returned to the AR device (an iPhone 13 Pro Max) and raycasted into a LiDAR-generated spatial mesh to estimate 3D position. These coordinates were used to overlay arrows and labels within the AR view, allowing users to visually locate required objects within their environment.

Evaluation results showed that participants overwhelmingly found the localization feature helpful, with responses indicating improved clarity, reduced uncertainty, and smoother task execution. This supports the general consensus in recent literature that real-time visual feedback grounded in task context can significantly enhance procedural support in AR, demonstrated here even with lightweight detection models and accessible hardware.

Nevertheless, several technical limitations emerged. Because the YOLO model was trained on small, custom datasets, it occasionally failed to recognize objects in unfamiliar poses or lighting conditions. Raycasting was also sensitive to occlusion and to depth inaccuracies near the edges of the camera view, which sometimes resulted in misleading 3D placements. While participants often compensated by repositioning their camera or retrying detection, these issues highlight the need for more robust depth sensing and more generalizable object detectors. An experimental auto-scanning feature—intended to detect objects continuously during camera motion—was promising but suffered from mesh-update lag, leading to

inaccurate placements.

In summary, the prototype demonstrates how a modular and robust object localization pipeline for AR systems can be designed. It leverages the strengths of LLMs for semantic mapping, the efficiency of YOLO for detection, and the spatial understanding provided by ARKit and LiDAR. Future improvements could explore more generalist vision-language models (e.g., Grounding DINO or Apple Ferret) and faster, denser mesh generation, but the current approach already offers strong evidence that such pipelines can meaningfully enhance real-time AR interactions in training tools.

10.3 RQ3: User Performance and Training Value

To what extent does the integration of instruction tracking and object localization improve user performance and training outcomes in AR-based procedural learning?

While this project did not include formal user performance testing or comparisons with traditional training methods, the final user evaluation offers valuable indications of the system's potential. All participants completed the procedural tasks with minimal external assistance, and many reported that the combination of real-time instruction tracking and object localization improved clarity, reduced uncertainty, and supported independent task execution.

The system's feedback loops—confirmation of completed steps, guidance on incomplete ones, and visual localization of unfamiliar objects—helped participants stay engaged and progress confidently. Several users described the system as “actively helping” rather than merely presenting static instructions, suggesting that the integration fostered a stronger sense of support and learning.

Although no longitudinal effects or knowledge retention were measured, questionnaire results, observed behavior, and qualitative feedback suggest that such a system could meaningfully enhance procedural training. The evaluation of this prototype confirms that the integration of instruction tracking and object localization can deliver valuable support in real-time AR-supported physical task performance and procedural learning scenarios. Future work could build on this by conducting controlled comparisons, measuring training efficiency, and evaluating long-term learning outcomes.

10.4 Main Question

How can an AR system be designed to effectively combine instruction tracking using Multimodal Large Language Models and object localization, leveraging vision-based AI models, to support users in performing physical tasks?

This thesis shows the feasibility of using an AR system to support users in physical task execution by integrating instruction tracking via MLLMs with real-time object localization powered by vision-based AI and AR spatial mapping.

The proposed design relies on a modular backend–frontend architecture. Instruction tracking is enabled through structured prompts that represent each task step and are processed by GPT-4o. This model evaluates egocentric images submitted by the user to determine whether a step has been completed and provides corrective feedback if not. Object localization is driven by a YOLO-based detector, which identifies relevant objects in the user’s environment and projects their positions into 3D space using LiDAR-based mesh raycasting. To make this possible, GPT-4o also semantically maps natural-language object references from instructions to recognizable YOLO class labels, allowing the system to extract and localize only task-relevant items.

User evaluation confirms the feasibility and usefulness of this integrated approach. Participants successfully completed tasks with minimal external help, consistently reporting that the system’s visual cues and real-time feedback enhanced clarity and confidence. Despite some technical limitations—such as occasional detection misses or mesh inaccuracies—users adapted to the system and found it supportive throughout task execution.

In summary, an effective AR system can be designed by combining:

1. **Instruction tracking** through MLLMs using structured, per-step prompts and egocentric visual input.
2. **Dynamic object localization** using vision-based detection and AR mesh projection.
3. **A contextual bridge** between task semantics and model input formats, provided by the reasoning capabilities of large language models.

This integrated architecture enables a resource-efficient yet effective approach to procedural task support, leveraging current AI models and consumer-grade AR hardware. Future work should explore scaling to more complex tasks, increasing robustness, and evaluating long-term training benefits.

10.5 Future Work

While this thesis demonstrated the feasibility of combining instruction tracking and object localization in an AR system, several directions remain for further exploration and development:

Integration of Advanced AR Devices such as Apple Vision Pro

Future work could explore integrating the system into advanced AR headsets like the Apple Vision Pro. These devices offer superior spatial tracking, higher-resolution passthrough, and more immersive interaction capabilities. Their hands-free nature makes them especially suitable for procedural training tasks, enabling users to follow instructions and receive feedback without needing to hold a device. Combined with spatial computing features and advanced eye and hand tracking, this integration could significantly enhance both usability and performance in real-world environments.

Automatic Object Scanning and Passive Step Monitoring

Future studies should investigate whether continuous background scanning for object localization and step recognition improves task efficiency and user experience compared to user-triggered interactions. This includes exploring how different user preferences—such as the desire for control versus convenience—affect the perceived usefulness and acceptance of passive monitoring systems.

Formal User Performance Evaluation

To quantify the learning benefits of this system, future research should include controlled studies comparing user performance, task efficiency, and error rates against traditional methods, both in the lab setting and in operational environments. Additionally, studies could measure retention over time to assess the long-term value of AI-supported AR guidance.

Scalability to Complex Tasks and Domains

The current system was tested on relatively short and self-contained procedures. Scaling to more complex tasks—such as multi-phase industrial workflows, medical procedures with branching paths, or collaborative field operations—will require improvements in system robustness and adaptability. This includes handling a larger number of steps, a broader range of object types, and more ambiguous or domain-specific instructions.

Future research should explore how the instruction tracking and object localization pipeline performs under increased task complexity and domain diversity. This may involve extending prompt structures to support conditional logic, integrating additional task metadata (e.g., dependencies or timing), and validating system behavior in real-world operational contexts. Supporting more specialized hardware or workflows (e.g., sterile environments or safety-critical procedures) may also require tailoring the system's UI and feedback mechanisms to meet domain-specific constraints.

Personalization and Adaptive Learning

Integrating learner models could allow instruction tailoring based on user background, skill level, or task performance. Adaptation in feedback style, pacing, or modality (e.g., switching from text to voice) would enhance learning outcomes and engagement.

Improved UI/UX and Onboarding Support

Several participants indicated areas for UI improvement. Future iterations should refine interaction design, instruction readability, and onboarding support. This includes visual clarity of overlays, user pacing controls, and context-aware tooltips.

Integration with MirageXR

A promising direction for future work is the integration of this system into the MirageXR platform—a functionality-rich, open-source AR learning environment that already supports features such as step-based guidance, motion capture, and remote collaboration. Embedding instruction tracking and object localization into MirageXR would not only extend its capabilities but also allow for broader testing in educational and industrial settings.

Moreover, MirageXR's support for real-time collaboration opens the door to remote assistance scenarios, where experts can monitor a user's progress, offer live feedback, or highlight specific objects in the AR view. This would further elevate the system from a standalone training tool to a fully collaborative and adaptive learning experience. Integrating modern AI-driven capabilities into such a platform represents a significant step toward comprehensive, intelligent AR-based training solutions.

Edge Deployment and Offline Support

To ensure functionality in low-connectivity environments, future systems could investigate local execution of detection and feedback models. Leveraging device-native accelerators like Apple's Neural Engine would support on-device inference and reduce reliance on remote servers.

Improved Feedback Quality and Explainability

LLM responses can sometimes be vague or overly verbose. Fine-tuning prompt formats or integrating uncertainty estimates would improve interpretability. Clearer guidance on why a step failed and how to fix it would enhance trust and effectiveness.

Use of Generalist Detectors and Open-Vocabulary Models

Instead of training custom YOLO models, integrating models like Grounding DINO or Apple Ferret could allow object referencing across tasks without retraining. These models offer open-vocabulary recognition and better transferability across domains.

Multimodal Feedback and Voice Interaction

Combining visual overlays with voice interaction would increase immersion and reduce reliance on screen-based UI. Voice-triggered commands, spoken instructions, and audio feedback would particularly benefit hands-busy scenarios.

Authoring Tools for Task Creation

Future work could explore how content authoring systems can be designed to effectively support domain experts in creating procedural tasks without requiring technical expertise. This includes investigating which interfaces, automation features (e.g., object extraction), and simulation capabilities best facilitate accurate, intuitive, and scalable task creation and validation across different domains.

In summary, the prototype presented in this thesis provides a promising foundation. Extending its capabilities through the directions outlined above could significantly advance the impact and usability of AI-powered AR training systems.

Bibliography

- [1] R. T. Azuma, 'A survey of augmented reality,' *Presence: Teleoperators & Virtual Environments*, vol. 6, no. 4, pp. 355–385, 1997. DOI: 10.1162/pres.1997.6.4.355.
- [2] M. Billinghurst, A. Clark and G. Lee, 'A survey of augmented reality,' *Foundations and Trends in Human–Computer Interaction*, vol. 8, no. 2-3, pp. 73–272, 2015. DOI: 10.1561/11000000049.
- [3] M. Dunleavy, C. Dede and R. Mitchell, 'Affordances and limitations of immersive participatory augmented reality simulations for teaching and learning,' *Journal of Science Education and Technology*, vol. 18, no. 1, pp. 7–22, 2009. DOI: 10.1007/s10956-008-9119-1.
- [4] M. B. Ibáñez and C. Delgado-Kloos, 'Augmented reality for stem learning: A systematic review,' *Computers & Education*, vol. 123, pp. 109–123, 2018. DOI: 10.1016/j.compedu.2018.05.002.
- [5] I. Radu, 'Augmented reality in education: A meta-review and cross-media analysis,' *Personal and Ubiquitous Computing*, vol. 18, no. 6, pp. 1533–1543, 2014. DOI: 10.1007/s00779-013-0747-y.
- [6] S. Srinidhi, E. Lu and A. Rowe, 'Xair: An xr platform that integrates large language models with the physical world,' in *2024 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2024, pp. 759–767. DOI: 10.1109/ISMAR62088.2024.00091.
- [7] S. Ren, K. He, R. Girshick and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. DOI: 10.48550/ARXIV.1506.01497. arXiv: 1506.01497 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1506.01497>.
- [8] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, 'You only look once: Unified, real-time object detection,' in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.
- [9] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson et al., 'Flamingo: A visual language model for few-shot learning,' in *Advances in Neural Information Processing Systems*, 2022, pp. 360–380. [Online]. Available: <https://arxiv.org/abs/2204.14198>.

- [10] OpenAI, *Hello gpt-4o*, <https://openai.com/index/hello-gpt-4o/>, Accessed 2025-06-22, May 2024.
- [11] P Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal and A. Chadha, *A systematic survey of prompt engineering in large language models: Techniques and applications*, 2025. DOI: 10.48550/ARXIV.2402.07927. arXiv: 2402.07927 [cs.AI]. [Online]. Available: <https://arxiv.org/abs/2402.07927>.
- [12] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le and D. Zhou, 'Chain-of-thought prompting elicits reasoning in large language models,' *arXiv preprint arXiv:2201.11903*, 2022. DOI: 10.48550/arXiv.2201.11903. [Online]. Available: <https://arxiv.org/abs/2201.11903>.
- [13] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., 'Language models are few-shot learners,' *arXiv preprint arXiv:2005.14165*, 2020. DOI: 10.48550/arXiv.2005.14165. [Online]. Available: <https://arxiv.org/abs/2005.14165>.
- [14] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery and D. Zhou, 'Self-consistency improves chain of thought reasoning in language models,' *arXiv preprint arXiv:2203.11171*, 2022. DOI: 10.48550/arXiv.2203.11171. [Online]. Available: <https://arxiv.org/abs/2203.11171>.
- [15] T. Van Gog, F. Paas, J. J. Van Merriënboer and P. Witte, 'Uncovering the problem-solving process: Cued retrospective reporting versus concurrent and retrospective reporting,' *Journal of Experimental Psychology: Applied*, vol. 11, no. 4, pp. 237–244, 2005. DOI: 10.1037/1076-898X.11.4.237.
- [16] WEKIT ECS Project, *Miragexr platform*, <https://github.com/WEKIT-ECS/MIRAGE-XR>, Accessed: 2025-06-21, 2024.
- [17] Unity Technologies, *Unity documentation*, <https://docs.unity3d.com/>, Accessed: 2025-06-21, 2024.
- [18] Unity Technologies, *Unity arfoundation documentation*, <https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.1/manual/index.html>, Accessed: 2025-06-21, 2024.
- [19] G. Evans, J. Miller, M. I. Pena, A. MacAllister and E. Winer, 'Evaluating the Microsoft HoloLens through an augmented reality assembly application,' in *Degraded Environments: Sensing, Processing, and Display 2017*, J. (N. Sanders-Reed and J. (J. A. III, Eds., International Society for Optics and Photonics, vol. 10197, SPIE, 2017, p. 101970V. DOI: 10.1117/12.2262626. [Online]. Available: <https://doi.org/10.1117/12.2262626>.

- [20] C. Garcia, M. Ortega, E. Ivorra, M. Contero, P. Mora and M. L. Alcañiz, 'Hol-orailway: An augmented reality system to support assembly operations in the railway industry,' *Advances in Manufacturing*, vol. 12, no. 4, pp. 764–783, 2024, ISSN: 2195-3597. DOI: 10.1007/s40436-023-00479-5. [Online]. Available: <https://doi.org/10.1007/s40436-023-00479-5>.
- [21] T. Richardson, S. Gilbert, J. Holub, F. Thompson, A. MacAllister, R. Radkowski, E. Winer, P. Davies and S. Terry, 'Fusing self-reported and sensor data from mixed-reality training,' in *Proceedings of the Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, Paper No. 14158; posted with permission, Arlington, VA: National Training and Simulation Association (NTSA), 2014. [Online]. Available: <https://dr.lib.iastate.edu/handle/20.500.12876/44354>.
- [22] B. Limbu, A. Vovk, H. Jarodzka, R. Klemke, F. Wild and M. Specht, 'Wekit.one: A sensor-based augmented reality system for experience capture and re-enactment,' in *Transforming Learning with Meaningful Technologies*, M. Scheffel, J. Broisin, V. Pammer-Schindler, A. Ioannou and J. Schneider, Eds., ser. Lecture Notes in Computer Science, vol. 11722, Cham: Springer International Publishing, 2019, pp. 158–171, ISBN: 978-3-030-29736-7. DOI: 10.1007/978-3-030-29736-7_12. [Online]. Available: https://doi.org/10.1007/978-3-030-29736-7_12.
- [23] M. Shridhar, L. Manuelli and D. Fox, 'Cliport: What and where pathways for robotic manipulation,' 2021. arXiv: 2109.12098 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2109.12098>.
- [24] D. Gao, D. Driess, T. Nguyen, R. Mason, F. Fei, M. Toussaint et al., 'Palm-e: An embodied multimodal language model,' *arXiv preprint arXiv:2303.03378*, 2023. DOI: 10.48550/arXiv.2303.03378. [Online]. Available: <https://arxiv.org/abs/2303.03378>.
- [25] R. Mon-Williams, G. Li, R. Long, W. Du et al., 'Embodied large language models enable robots to complete complex tasks in unpredictable environments,' *Nature Machine Intelligence*, vol. 7, no. 6, pp. 592–601, 2025. DOI: 10.1038/s42256-025-01005-x.
- [26] X. Liang, J. Ye, Y. Wang, H. He, Y. Tang, J. Sun et al., 'Taskmatrix.ai: Completing tasks by connecting foundation models with millions of apis,' *arXiv preprint arXiv:2303.16434*, 2023. DOI: 10.48550/arXiv.2303.16434. [Online]. Available: <https://arxiv.org/abs/2303.16434>.
- [27] D. Stover and D. A. Bowman, 'TAGGAR: General-purpose task guidance from natural language in augmented reality using vision-language models,' in *Proceedings of the ACM Symposium on Spatial User Interaction (SUI)*, 2024. DOI: 10.1145/3677386.3682095.

- [28] C. Jiang, Y. Fan, J. Xie, E. Kuang, B. Feng, K. Zhang and M. Fan, ‘Designing llm-powered multimodal instructions to support rich hands-on skills remote learning: A case study with massage instructors and learners,’ in *Proc. CHI 2025*, 2025. DOI: 10.1145/3706598.3713677. [Online]. Available: <https://dl.acm.org/doi/10.1145/3706598.3713677>.
- [29] N. Hussain, *Multimodal language models for domain-specific procedural video summarization*, 2024. arXiv: 2407.05419 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2407.05419>.
- [30] Ultralytics, *Yolo model comparison*, <https://docs.ultralytics.com/compare/>, Accessed on 2025-06-05, 2024.
- [31] H. You, H. Zhang, Z. Gan, X. Du, B. Zhang, Z. Wang, L. Cao, S.-F. Chang and Y. Yang, *Ferret: Refer and ground anything anywhere at any granularity*, 2023. DOI: 10.48550/ARXIV.2310.07704. arXiv: 2310.07704 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2310.07704>.
- [32] Ultralytics, *Yolov8 documentation (model usage and tutorials)*, <https://docs.ultralytics.com/yolov8/>, Accessed on 2025-05-31, 2023.
- [33] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, ‘Ssd: Single shot multibox detector,’ in *European Conference on Computer Vision (ECCV)*, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2.
- [34] S. Ramírez, *Fastapi*, <https://fastapi.tiangolo.com/>, Accessed on 2025-06-05, 2025.
- [35] OpenAI, *Model comparison*, <https://platform.openai.com/docs/models/compare>, Accessed on 2025-06-05, 2024.
- [36] T. Möller and B. Trumbore, ‘Fast, minimum storage ray-triangle intersection,’ *Journal of graphics tools*, vol. 2, no. 1, pp. 21–28, 1997. DOI: 10.1080/10867651.1997.10487468.

Appendix A

Extended YOLO Performance Results

Table A.1: Full results of YOLOv8n inference benchmark on 29 test images.

Image #	Avg Inference Time (s)	FPS
1	0.053	18.78
2	0.054	18.48
3	0.057	17.49
4	0.054	18.62
5	0.054	18.51
6	0.052	19.27
7	0.057	17.61
8	0.052	19.18
9	0.056	18.01
10	0.054	18.57
11	0.052	19.20
12	0.057	17.56
13	0.055	18.27
14	0.050	19.96
15	0.052	19.26
16	0.054	18.59
17	0.058	17.20
18	0.048	20.78
19	0.054	18.51
20	0.057	17.62
21	0.059	17.02
22	0.059	17.08
23	0.052	19.22
24	0.053	19.03
25	0.053	18.99
26	0.053	19.01
27	0.060	16.74
28	0.054	18.63
29	0.053	18.91

Appendix B

User evaluation: Workshop

B.1 User Workshop Notes: 2 April 2025

B.1.1 General Impressions and Task Successes

- The system worked better than expected overall, though some errors were observed.
- Tasks requiring “finger on the button” or other clear physical cues were reliably recognized.
- For tasks like wiping the VR headset, the system detected contextually relevant actions (e.g., focusing on commonly touched areas). Disposing the wipe was only accepted when placed in the correct trash bin.
- Multilingual robustness was observed: one user wrote an exercise in German (“yoga pose”), which the system tracked correctly.
- Attempted to “fool” the system by placing the VR headset in front of a black box (instead of inside); the system sometimes accepted such borderline completions.
- Physical tasks such as charging a Quest device, changing batteries, and tying shoes (including switching left/right) were generally recognized.
- Recognition of spatial relationships (e.g., Quest controllers arranged at the sides of the headset in front of the computer) worked well.
- Writing “Hello” on paper was tracked step by step without issue.
- Simple wearable tasks (putting on glasses, headphones) were reliably tracked.

B.1.2 Observations and Edge Cases

- Instructions need to be precise; vague steps led to inconsistent completion.
- In some cases, test recognition output was “clunky,” with repeated or mixed-up instructions in a single message.
- The frontend could be improved: users suggested step reordering, step numbers, completed step indicators, and clearer “submit”/“start” separation.

- Some users wrote instructions in Spanish but always received system responses in English, indicating a need for multilingual output.
- Task completion sometimes depended on clear visual cues—e.g., turning on a monitor was only detected if a finger visibly pressed the button.
- System recognized whether the Quest was ON (via LED) and whether Unity was open; closing apps on the desktop was less reliable.
- “Override” or manual “skip step” features would be valuable for testing.
- “Discovery” mode (continuous feed) was seen as potentially dangerous if used unsupervised.

B.1.3 Feature Suggestions

- Provide users with guidance on writing structured, precise prompts; consider an LLM-powered assistant for prompt creation.
- Add a search/discovery function for existing activities/tasks.
- Implement a star rating or feedback system for tasks.
- Capture user email addresses for learning analytics.
- Add a “disable photo” button when task is finished.
- Distinguish between “task evaluation” and “task assistance” modes.
- Allow users to select the preferred language for system responses.
- Consider integration of “override” or “skip step” buttons.

B.1.4 Limitations and Concerns

- The use of OpenAI models may not be suitable for industrial contexts (privacy and compliance concerns).
- Manual button pressing is counter to the goal of seamless task tracking.
- There are only two states for feedback (success or repeat instructions); suggest including visual cues for “what was detected” in the “repeat” case.
- Consider filtering out irrelevant images from the task feed before sending to the model.
- Potential liability: e.g., if the system incorrectly validates a critical safety step (“changing a tire”).

B.1.5 Potential Application Domains (As Identified by Participants)

- Assembly tasks (e.g., IKEA furniture).
- Engine repair and general maintenance.
- Changing car tires.
- Cooking assistance for those who struggle with recipes.
- Climbing gyms (checking safety equipment).
- Using gym machines correctly.
- Rehabilitation exercises.
- Step-by-step confirmation for users with low vision or blindness.
- Assisting people with cognitive impairments (e.g., using a printer).

B.1.6 What Users Liked Most

- The system recognized many things users did not expect it to (“amazing”).
- Most tasks and steps were reliably recognized.
- For 5 out of 12 users, the accuracy was described as “scary good.”

B.2 Representative Workshop Logs

This appendix provides a representative, edited sample of user-system interaction logs recorded during the hands-on workshop. The examples are selected and edited for clarity, with images omitted for privacy and length. These logs illustrate typical user behavior, step-wise system feedback, and recurring interaction patterns. They do not represent the full workshop dataset, but give a cross-section of key cases and outcomes.

B.2.1 Scenario: Prepare to Go to Lunch (Lab to Canteen)

Task Steps:

1. Put the access badge around your neck.
2. Take the bottle in your hand.
3. Turn off the laptop.
4. Close the laptop.
5. Leave the laptop on the desk.
6. Plug in the tablet.
7. Leave the backpack on the chair.
8. Turn off the lights in the lab.
9. Close the door.

User-System Interaction Log:

- **Step 1: Put the access badge around your neck.**
 - System: Step Complete: Yes.
Next step: Take the bottle in your hand.
- **Step 2: Take the bottle in your hand.**
 - System: Step Complete: Yes.
Next step: Turn off the laptop.
- **Step 3: Turn off the laptop.**
 - System: Step Complete: Yes.
Next step: Close the laptop.
- **Step 4: Close the laptop.**
 - System: Step Complete: Yes.
Next step: Leave the laptop on the desk.

- **Step 5: Leave the laptop on the desk.**
 - System: Step Complete: Yes.
Next step: Plug in the tablet.
- **Step 6: Plug in the tablet.**
 - System: Step Complete: Yes.
Next step: Leave the backpack on the chair.
- **Step 7: Leave the backpack on the chair.**
 - System: Step Complete: Yes.
Next step: Turn off the lights in the lab.
- **Step 8: Turn off the lights in the lab.**
 - *Attempts 1-6:* System responded: Step Complete: No.
Example prompts: “Please turn off the lights in the lab.”
“Ensure the lights are turned off before proceeding.”
 - *Attempt 7:* System: Step Complete: Yes.
Next step: Close the door.
- **Step 9: Close the door.**
 - System: Step Complete: Yes.
All steps to prepare for lunch are completed. You’re ready to go to the canteen!

Notes:

- Most steps were recognized and marked as complete on the first attempt, except for “turn off the lights,” which required multiple attempts and clear visual confirmation.
- The system consistently prompted for observable evidence before marking steps as complete, especially for actions with subtle or less visible outcomes.

B.2.2 Scenario: Put an Earphone in Your Ear

Task Steps:

1. Pick up the headphone box.
2. Open the earphone box.
3. Pick up a headset.
4. Put the earphone in your ear.

User-System Interaction Log:

- **Step 1: Pick up the headphone box.**
 - System: Step Complete: Yes.
Next step: Open the earphone box.
- **Step 2: Open the earphone box.**

- System: Step Complete: Yes.
Next step: Pick up a headset.
- **Step 3: Pick up a headset.**
 - Attempt 1: System: Step Complete: No.
Prompt: “Please pick up the headset from the earphone box.”
 - Attempt 2: System: Step Complete: Yes.
Next step: Put the earphone in your ear.
- **Step 4: Put the earphone in your ear.**
 - System: It seems you’ve completed this step as well. All steps have been completed successfully!

Notes:

- The system successfully recognized all physical actions, requiring an extra attempt for “pick up a headset” before confirming completion.
- The workflow shows the model’s insistence on clear, observable actions before marking a step as complete.

B.2.3 Scenario: Set Up My VR Work Station**Task Steps:**

1. Turn on computer
2. Turn on monitor
3. Get cardboard box containing the Meta Quest 2
4. Put Quest 2 and its controllers on table
5. Turn on the Quest 2
6. Connect Quest 2 to the computer using a USB cable
7. Open the Unity game engine on the computer
8. Open an Integrated Development Environment (code editor) on the computer

User-System Interaction Log:

- **Step 1: Turn on computer**
 - Attempt 1: *System*: Step Complete: No. “Please turn on your computer. Once it’s on, let me know!”
 - Attempt 2: *System*: Step Complete: Yes. Next step: Turn on monitor.
- **Step 2: Turn on monitor**
 - Attempt 1: *System*: Step Complete: No. Prompted for correct cable connection and to press power button.
 - Attempt 2: *System*: Step Complete: Yes. Next step: Get cardboard box containing the Meta Quest 2.
- **Step 3: Get cardboard box containing the Meta Quest 2**

- Attempt 1: *System*: Step Complete: Yes. Next step: Put Quest 2 and its controllers on table.
- **Step 4: Put Quest 2 and its controllers on table**
 - Attempt 1: *System*: Step Complete: Yes. Next step: Turn on the Quest 2.
- **Step 5: Turn on the Quest 2**
 - Attempt 1: *System*: Step Complete: Yes. Next step: Connect Quest 2 to the computer using a USB cable.
- **Step 6: Connect Quest 2 to the computer using a USB cable**
 - Attempt 1: *System*: Step Complete: No. Prompted user to connect cable securely.
 - Attempt 2: *System*: Step Complete: Yes. Next step: Open the Unity game engine on the computer.
- **Step 7: Open the Unity game engine on the computer**
 - Attempt 1: *System*: Step Complete: Yes. Next step: Open a code editor on the computer.
- **Step 8: Open an Integrated Development Environment (code editor) on the computer**
 - Attempt 1: *System*: Step Complete: Yes. All steps completed. Your VR workstation setup is finished!

Notes:

- The system generally tracked the physical and digital steps well.
- Steps involving visible state change (e.g., “turn on monitor”) required clear user actions—such as pressing the power button or ensuring cables were connected.
- Multiple attempts were sometimes needed when the system did not initially recognize a completed action.
- Complex tasks combining hardware setup and software launch were handled robustly after sufficient visual evidence.

B.2.4 Scenario: Tie Your Shoe

Task Steps:

1. Take a shoelace in each hand.
2. Cross both laces halfway up the laces.
3. Take the end of the left lace and put it over the right lace, then poke it underneath the right lace.
4. Hold on to the ends of both laces and pull tight to make a knot. The knot should be close to the shoe.
5. Take the right lace and make a small loop, or ‘bunny ear’. This bunny ear

should have a long tail.

6. Take the left lace and make a small loop, or 'bunny ear'. This bunny ear should have a long tail.
7. Cross the middle of the right bunny ear over the middle of the left bunny ear.
8. Take the top of the left bunny ear and put it over the right bunny ear. Now poke it through the hole.
9. Put your fingers inside the ends of the bunny ears, then hold on and pull tight. You've now tied the loops in a knot. The shoelaces are tied.

User-System Interaction Log:

- **Step 1:** *System:* Step Complete: Yes. Next step: Cross both laces halfway up the laces.
- **Step 2:** *System:* Step Complete: Yes. Next step: Take the end of the left lace and put it over the right lace, then poke it underneath.
- **Step 3:**
 - Attempt 1: *System:* Step Complete: No. Prompted for correct action.
 - Attempt 2: *System:* Step Complete: Yes. Next step: Pull tight to make a knot.
- **Step 4:** *System:* Step Complete: Yes. Next step: Make a small loop with the right lace.
- **Step 5:**
 - Attempt 1: *System:* Step Complete: No. Prompted for correct action.
 - Attempt 2: *System:* Step Complete: Yes. Next step: Make a small loop with the left lace.
- **Step 6:** *System:* Step Complete: Yes. Next step: Cross the bunny ears.
- **Step 7:** *System:* Step Complete: Yes. Next step: Put top of the left bunny ear over the right and poke it through the hole.
- **Step 8:** *System:* Step Complete: Yes. Next step: Pull tight.
- **Step 9:** *System:* Step Complete: Yes. All steps completed. Shoelaces are tied.

Notes:

- Most steps were recognized on the first attempt, but for fine-grained manipulations (steps 3 and 5), the system required multiple attempts and prompted for clarification.
- The system could distinguish between micro-actions (e.g., making bunny ears with each hand) and track the sequence reliably.
- This scenario demonstrates the model's robustness for detailed procedural instructions, while also surfacing the importance of precise hand/object positioning for recognition.

B.2.5 Scenario: Write “Hello” on Paper

Task Steps:

1. Take a paper.
2. Take a pencil.
3. Grab the pencil.
4. Stick the pencil to the paper.
5. Write “H”.
6. Write “E”.
7. Write “LL”.
8. Write “O”.

User-System Interaction Log:

- **Step 1:** *System:* Step Complete: Yes. Next step: Take a pencil.
- **Step 2:** *System:* Step Complete: Yes. Next step: Grab the pencil.
- **Step 3:** *System:* Step Complete: Yes. Next step: Stick the pencil to the paper.
- **Step 4:** *System:* Step Complete: Yes. Next step: Write “H”.
- **Step 5:** *System:* Step Complete: Yes. Next step: Write “E”.
- **Step 6:** *System:* Step Complete: Yes. Next step: Write “LL”.
- **Step 7:** *System:* Step Complete: Yes. Next step: Write “O”.
- **Step 8:** *System:* Step Complete: Yes. All steps completed. “Hello” successfully written on paper.

Notes:

- All micro-steps were recognized on the first attempt.
- The system distinguished between sequential writing actions, confirming completion at each stage.
- Minor grouping ambiguity (e.g., “LL” as a single step) was handled gracefully.
- This scenario highlights reliable tracking of fine-grained, sequential manual actions.

Appendix C

Unity Debugging Extras

This appendix gathers supplementary screenshots that document some of the issues encountered—and their fixes—during early development of the object-localization pipeline:

- **Subsection C.1** shows how a cluttered Unity hierarchy with redundant sub-components caused bounding-box mis-alignments.
- **Subsection C.1.2** details how incorrect canvas-resolution settings led to orientation and scaling problems on iPhone.
- **Section C.2** presents additional bounding-box and ray-cast overlay screenshots used to validate each fix.

C.1 Unity Hierarchy and Canvas Debugging

Figures C.1–C.4 illustrate the cluttered Unity hierarchy with redundant sub-components, while Figures C.5–C.7 compare the initial and corrected canvas configurations.

C.1.1 Unity Hierarchy Screenshots

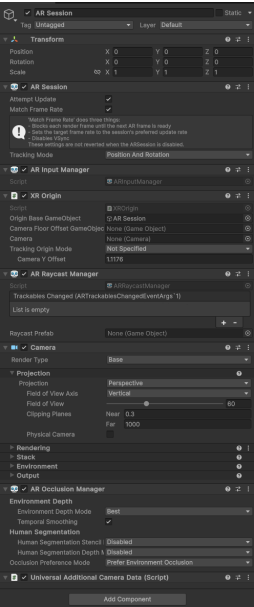


Figure C.1: AR Session object hierarchy with redundant subcomponents.

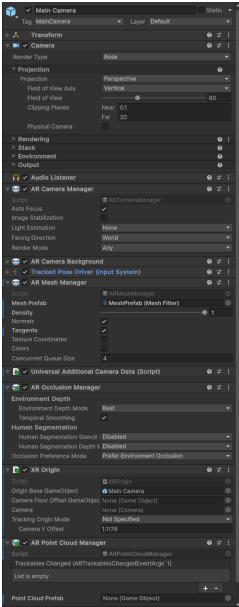


Figure C.2: Main Camera hierarchy view showing overlapping components.

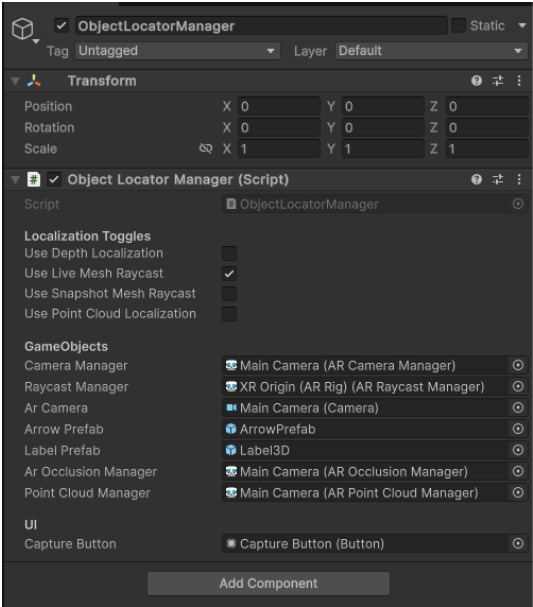


Figure C.3: ObjectLocatorManager component contributing to hierarchy clutter.

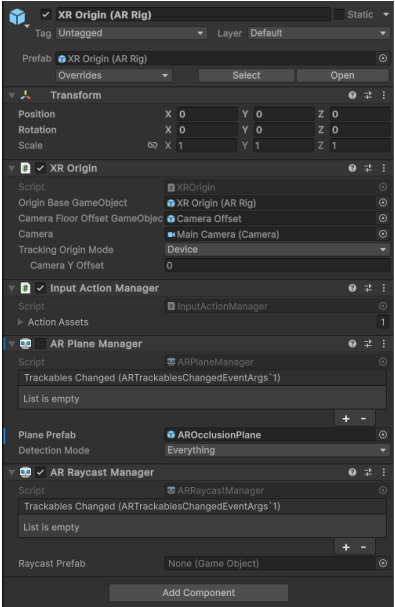


Figure C.4: XR Origin configuration with redundant subcomponents.

C.1.2 Unity Canvas Resolution Debugging

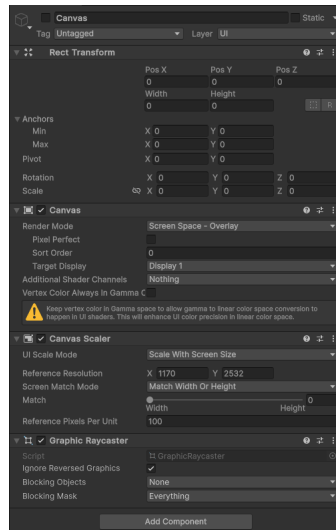


Figure C.5: Root canvas structure with incorrect portrait resolution.

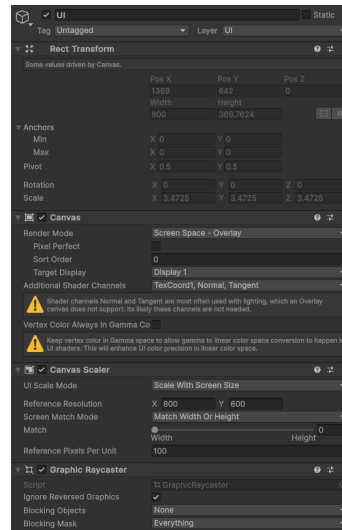


Figure C.6: UI canvas with inconsistent resolution.

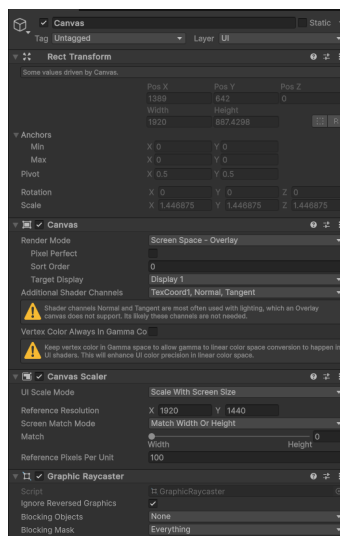


Figure C.7: Corrected canvas configuration matching iPhone screen resolution and landscape orientation.

C.2 Additional Debugging Visualizations

The screenshots in Figures C.8–C.13 illustrate bounding-box overlays and ray-cast visualizations used to verify detector alignment and identify errors.



Figure C.8: Bounding-box debugging visualization example.

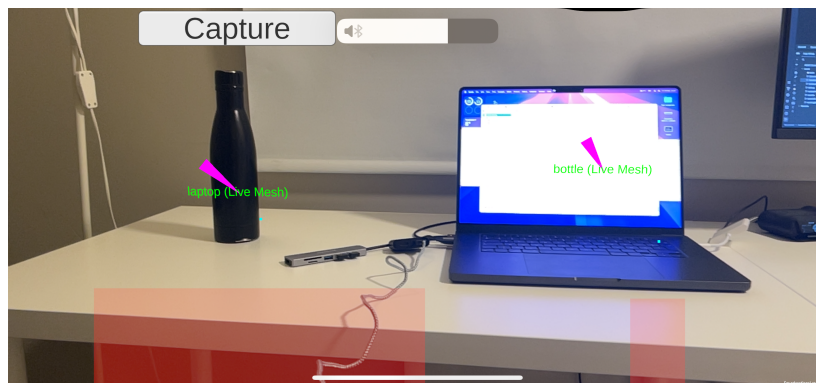


Figure C.9: Bounding-box debugging visualization example.

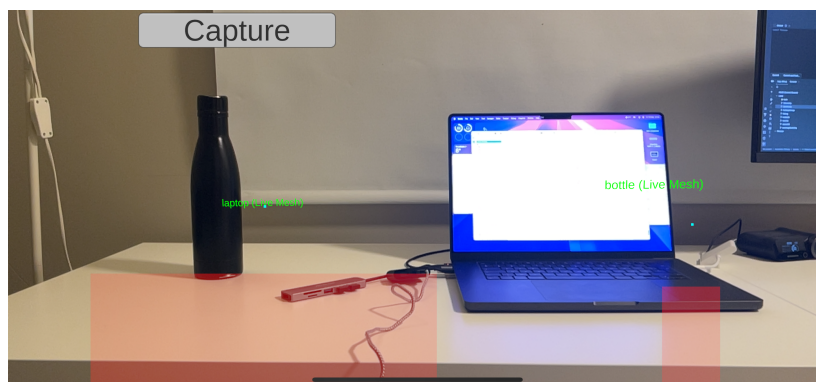


Figure C.10: Bounding-box debugging visualization example.

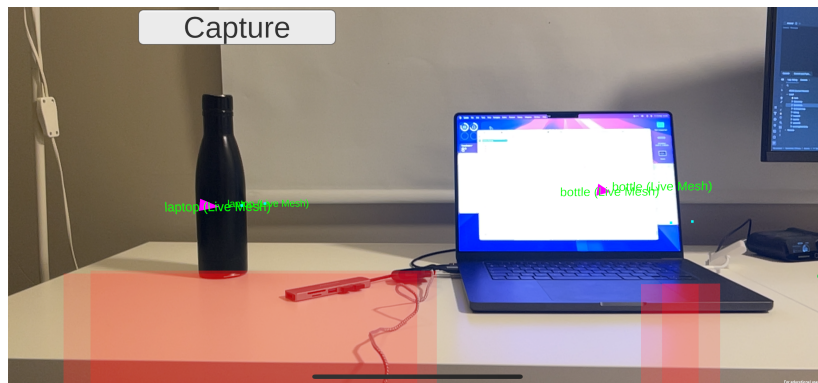


Figure C.11: Bounding-box debugging visualization example.

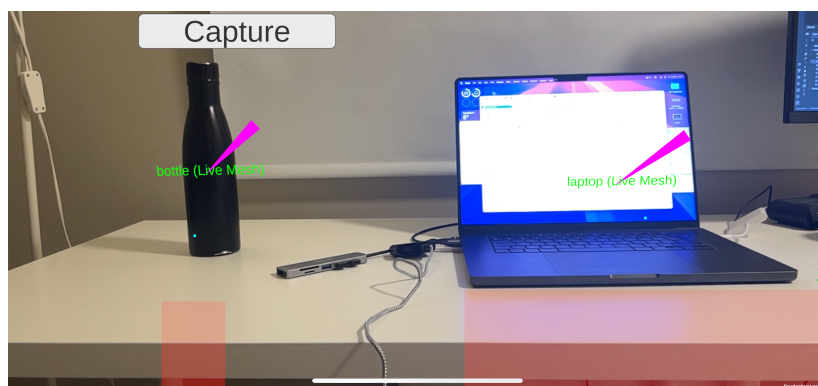


Figure C.12: Bounding-box debugging visualization example.

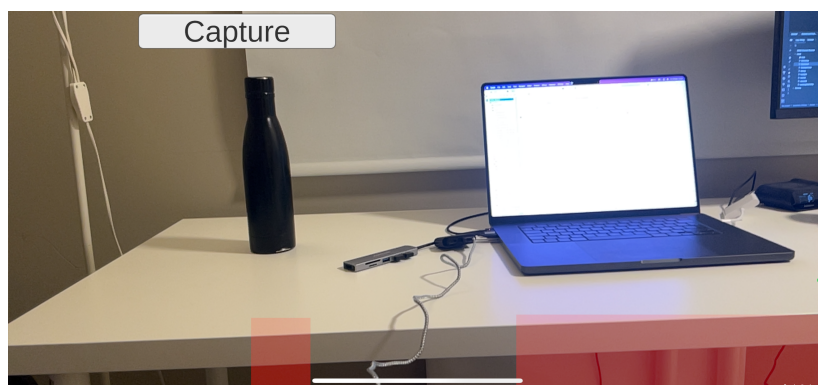


Figure C.13: Bounding-box debugging visualization example.

Appendix D

GPT Prompt for Object Extraction

This appendix contains the exact GPT prompt used to extract YOLO-detectable physical objects from task descriptions. It consists of a system message and a user message example, formatted for programmatic use.

D.0.1 System Message

Extract physical objects relevant to the task that are also detectable by YOLO. Always respond with valid JSON and nothing else. No explanation, no comments – just the JSON.

D.0.2 User Message Example

You are given a task titled: "Drink water"

Here are the steps of the task:

1. Take the bottle of water
2. Pour it into the cup

Your goal is to assist in building an AR-guided task assistant. To help with this:

1. Carefully read the task steps and identify all physical objects that users interact with or that are useful to visualize during task completion.
2. From these, map each object to a corresponding class from the following YOLO-detectable COCO classes (synonyms and generalizations are acceptable):
["bottle", "cup", ..., "cell phone", "remote", ...]
3. Only include objects that are genuinely helpful to localize during the task (e.g., tools, containers, interactive items).
4. For each relevant object, output:
 - The name as it appears in the task step (step_name)
 - The YOLO class name (yolo_name)
 - The YOLO class ID (yolo_class_id)
 - The list of step indices (1-indexed) where the object is used or should be shown

Output the result as valid JSON using this structure:

```
{
  "objects": [
    {
      "step_name": "water bottle",
      "yolo_name": "bottle",
      "yolo_class_id": 39,
      "used_in_steps": [1, 2]
    }
  ]
}
```

Appendix E

Final Evaluation Tasks

This appendix provides a detailed overview of the procedural tasks used during the final evaluation of the system. Each task was designed to involve multiple objects and actions, enabling a comprehensive test of instruction tracking, object localization, and user interaction.

E.1 VR Treadmill Setup

The *VR Treadmill* task simulates the setup process for a virtual reality treadmill system. It guides the user through a sequence of steps involving hardware activation and wearable equipment placement. This task was used to evaluate the system's ability to support sequential object-driven workflows in an AR-enhanced environment.

Steps

1. **Check VR Base Station 1:** Check if the first VR base station is on (Object ID: 0). A green light near the top, in the middle, should be visible.
2. **Check VR Base Station 2:** Check if the other VR base station is on (Object ID: 1). You should see a green light near the top, in the middle.
3. **Turn on Powerbox:** Find the powerbox (Object ID: 2) and turn on the switch. Wait for the platform to become horizontal before continuing.
4. **Wear Platform Shoes:** Take the two platform shoes (Object IDs: 3, 4) and put them on.
5. **Enter Treadmill Ring:** Go and stand inside the ring on the platform (Object ID: 5).
6. **Raise the Ring:** Raise the ring (Object ID: 5) to hip level using the raising lever (Object ID: 6) located at the side of the ring.
7. **Put on VR Headset:** Put on the VR headset (Object ID: 7).
8. **Take VR Controllers:** Take both VR controllers (Object IDs: 8, 9).

Relevant Objects

Object ID	Name
0	VR base station 1
1	VR base station 2
2	Powerbox
3	Platform shoe 1
4	Platform shoe 2
5	Treadmill ring
6	Raising lever
7	VR headset
8	VR controller 1
9	VR controller 2

Table E.1: Objects used in the VR Treadmill task.

E.2 Gastroscopy Setup

The *Gastroscopy* task simulates the basic setup and initial operation of a gastroscopy procedure using a medical training station. The steps involve interaction with electronic components and simulated anatomy, providing a compact yet effective test case for instruction tracking and object highlighting.

Steps

1. **Power On Viewing Screen:** Turn on the viewing screen by pressing the power button on the front (Object ID: 0). Show that the screen is on.
2. **Connect Camera Controller:** Plug the camera controller's cable (Object ID: 1) into the socket (Object ID: 2). Show that it is plugged in.
3. **Pick Up Camera Controller:** Take the camera controller (Object ID: 1) in your hand. Show that it is in your hand.
4. **Insert Camera into Stomach:** Take the camera (Object ID: 3) and put it in the entrance of the stomach (Object ID: 4).

Relevant Objects

Object ID	Name
0	Viewing screen
1	Camera controller
2	Socket
3	Camera
4	Stomach

Table E.2: Objects used in the Gastroscopy task.

Appendix F

Final Evaluation: Questions

F.1 Participant Demographics and Questionnaire

To support a thorough evaluation of the system, demographic data and both quantitative and qualitative feedback were collected from participants during the final user evaluation.

Demographic Information

Before beginning their assigned task, participants were asked to provide the following background details:

- **Gender**
- **Age Bracket** (e.g., 18–24, 25–34, etc.)
- **Assigned Task Case** (VR Treadmill or Gastroscopy)
- **Expertise on the Case Domain** (Yes/No)
- **Previous Experience with Augmented Reality Systems** (Yes/No)

Post-Task Questionnaire

After completing the task, participants answered the following structured questions. Unless otherwise indicated, responses were collected using a 5-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree):

1. The system was helpful in completing the procedure.
2. The ability to locate objects was helpful during the procedure.
3. The object localizations were accurate.
4. The instructions provided by the system were easy to understand.
5. The system responded quickly enough to complete each step without unnecessary delays.
6. I felt confused or hesitant at one or more steps due to the system. (*Response options: No, not at all / Yes, once or twice / Multiple times*)
7. If yes, describe what happened and at which step.

8. The system made the procedure easier to complete compared to doing it without assistance.
9. What did you like most about the system?
10. What would you improve in the system?
11. Any other comments or suggestions?

*Note: Some questions were rephrased to match a uniform Likert-style agreement format while preserving their original intent. For example, the original question “Was the ability to locate objects helpful to you during the procedure?” (response options: not helpful at all, not really helpful, neutral, somewhat helpful, yes, very helpful) was rephrased as “**The ability to locate objects was helpful during the procedure.**” to fit the agreement scale.*

Final Interview Questions

At the end of the final user evaluation, participants took part in a brief semi-structured interview designed to gather deeper qualitative insights. The interview questions were crafted to align with the updated research questions and aimed to explore participants’ perceptions of system understanding, instruction quality, and educational potential. The following five questions were asked:

1. Did you feel that the system understood what you were doing and provided instructions that matched what you needed to do?
2. How helpful and accurate was the system’s object highlighting (e.g., arrows, overlays)?
3. Did the combination of step-by-step instructions and object highlighting help you complete the task effectively?
4. Do you see the value in using this type of AR system to help people learn or perform physical tasks?
5. If you could change one thing about the system to make it better, what would it be?

