Linköping University | Department of Management and Engineering
Master's Thesis, 30 credits | Mechanical Engineering
Spring 2025 | LIU-IEI-TEK-A--25/05165--SE

# Integrating Vision, Language Models, & Robotic Control for Personalised Task Execution in Virtual Environments

Author:
**Oscar Chigozie Ikechukwu**

Supervisor: Mehdi Tarkian
Examiner: Johan Persson

## Thesis Report

12 June 2025

Linköping University
SE-581 83 Linköping, Sweden
+46 13 28 10 00, www.liu.se

Linköping University | Department of Management and Engineering
Master's thesis, 30 credits| Mechanical Engineering
Spring 2025| LIU-IEI-TEK-A--25/05165--SE

# Integrating Vision, Language Models, and Robotic Control for Personalised Task Execution in Virtual Environments

**Oscar Chigozie Ikechukwu**

Supervisor: Mehdi Tarkian
Examiner: Johan Persson

Linköping University
SE-581 83 Linköping, Sweden
+46 013 28 10 00, www.liu.se

# Abstract

This thesis proposes a modular framework that integrates computer vision, large language models (LLMs), and robotic control to enable personalised task execution in a simulated robotic environment. Addressing the limitations of current systems in dynamic, user-centric contexts, the architecture combines vision-based perception, natural language understanding, and adaptive task planning to align user intent with autonomous robot behaviour.

Biometric authentication via facial and voice recognition enables individualized interaction, supported by a centralized PostgreSQL database that manages user profiles, scene data, operation sequences, and task details. Multi-modal inputs (voice and gesture) are processed using open-source LLMs (e.g., Mistral, LLaMA) and converted into structured action plans. These plans, coordinated by a LangGraph-based orchestration layer, are validated in NVIDIA Omniverse Isaac Sim and executed on an ABB YuMi robot.

Empirical evaluation across 50+ test sessions demonstrated high task fidelity (80–90% success), effective sim-to-real transfer, and improved command interpretation through personalization. The system offers a scalable approach for adaptive, context-aware human-robot collaboration in industrial domains.

**Keywords**: *Human-robot interaction, Large language models, Robotic Task planning, Vision-language integration, Personalised robotic behaviour, Omniverse Isaac Sim*

# Declaration

I hereby declare that this thesis titled "*Integrating Vision, Language Models, and Robotic Control for Personalised Task Execution in Virtual Environments*", submitted to Linköping University in partial fulfilment of the requirements for the master's degree in Mechanical Engineering, is my original work.

In preparing this thesis, I have utilized AI-assisted editing tools, specifically ChatGPT and Grammarly, to refine technical documentation, enhance structural coherence, and ensure linguistic accuracy. However, all core ideas, analyses, and conclusions presented herein are my own and reflect the original research conducted during this thesis project.

It is acknowledged that certain software components particularly the early versions of the vision-processing module, were partially developed prior to the official start of the thesis semester. Additionally, the simulation environment and robot control scripts were co-developed by colleagues, *Sanjay Nambiar* and *Rahul Chiramel*, as part of earlier related research. These modules fall outside the formal scope of this thesis but are included and described in the report to clarify how their integration was achieved. All components were subsequently revised, extended, and adapted to align with the specific goals and contributions of this thesis work.

This research has not been submitted for any other degree or examination at any other institution. Any material derived from other sources has been appropriately acknowledged and cited.

*Oscar Chigozie Ikechukwu*
*12 June 2025*

# Acknowledgements

I wish to express my deepest gratitude to my supervisor, *Mehdi Tarkian*, for his invaluable guidance, insightful feedback, and unwavering support throughout this research journey. His expertise in design automation have been instrumental in the successful completion of this thesis.

Special thanks are due to my examiner, *Johan Persson*, for his critical evaluations and constructive suggestions, which have significantly strengthened the quality of this work.

I am also sincerely grateful to my research colleagues, *Sanjay Nambiar* and *Rahul Chiramel*, as well as my friends at Linköping University, for their camaraderie, engaging discussions, and continuous encouragement. Your contributions have greatly enriched my academic experience.

I extend my heartfelt appreciation to my family for their steadfast support, patience, and constant motivation. Your belief in me has been a cornerstone of my resilience throughout this endeavour.

Finally, I acknowledge the broader research community, whose foundational work has provided essential inspiration and direction for this study.

All glory belongs to God!

# Table of Contents

# List of Figures & Tables

## List of Figures



## List of Tables

# Abbreviations

| | |
|---|---|
| AI: | Artificial Intelligence |
| BERT: | Bidirectional Encoder Representations from Transformers |
| Cobot: | Collaborative robot |
| CLIP: | Contrastive Language–Image Pretraining |
| GDPR: | General Data Protection Regulation |
| GPT: | Generative Pre-trained Transformer |
| GUI: | Graphical User Interface |
| HRI: | Human-Robot Interaction |
| LLM: | Large Language Model |
| NLP: | Natural Language Processing |
| RL: | Reinforcement Learning |
| ROS: | Robot Operating System |
| Sim2Real: | Simulation-to-Reality |
| UR: | Universal Robots (collaborative robot brand) |
| URDF: | Unified Robot Description Format |
| USD: | Universal Scene Description (Omniverse file format) |
| VLM: | Vision-Language Model |
| YOLO: | You Only Look Once (object detection framework) |

# Nomenclature

Affordance Recognition: Identifying how objects can be manipulated based on visual cues (e.g., a "handle" implies pulling).

Control Script: Code translating high-level task plans into low-level robot movements (e.g., joint trajectories).

Contrastive Language-Image Pretraining (CLIP): a vision-language model that aligns visual and textual representations, enabling robots to interpret commands like 'pick up the red cup.'

Conversational Memory: An LLM's ability to retain context from prior user interactions for adaptive dialogue.

Modular Architecture: A system design where components (vision, language, control) operate independently but integrate seamlessly.

Omniverse Simulation: A virtual environment (NVIDIA) for testing robotic systems with physics-based accuracy.

Personalization Layer: A database-driven component that tailors interactions using user-specific data (e.g., preferences, history).

Reality Gap: Discrepancies between idealized simulation and real-world robotic performance conditions.

Task Graph: A structured representation of tasks/subtasks generated from natural language commands.

Vision-Language Integration: Combining computer vision (object detection) and language models (LLMs) to interpret and execute tasks.

# 1. Introduction

This chapter introduces the motivation and problem context for the thesis. It highlights the background; key limitations in current robotic systems and presents a proposed framework that integrates vision, language, and control in simulation. The chapter further outlines the research aims, key questions, expected outcomes, and discusses the study's limitations and scope.

## 1.1. Problem Context

Imagine instructing a robot, *"Tidy the lab while I finish my coffee,"* only to find it attempting to clean by stuffing cables into a microwave. While humorous, this illustrates a serious limitation: most robotic systems excel in structured, predictable environments but struggle in dynamic, human-centred spaces.


*Figure 1-1. A robot misinterpreting the command to "tidy up" (AI generated image)*

Collaborative robots are increasingly expected to operate in such unstructured settings, where humans communicate using casual, nuanced, and often ambiguous language. Yet, current systems rely on rigid commands, possess little to no contextual awareness, and often cannot recall prior interactions. This disconnect between natural human communication and robotic execution remains a major barrier to real-world adoption.

To bridge this gap, the seamless integration of computer vision, natural language processing, and advanced robotic control is essential. This thesis introduces a unified framework (Figure 1-2) that combines vision-based perception, large language models (LLMs), and adaptive robotic control to enable collaborative robots to interpret user commands, infer contextual intent, and simulate task execution within a virtual environment. The goal is to develop an AI-driven pipeline in which a cobot—such as ABB's YuMi or a Universal Robot arm—can process high level natural language commands, perform task decomposition and planning via LLM-based reasoning, and generate virtual demonstrations of task sequences prior to physical execution.

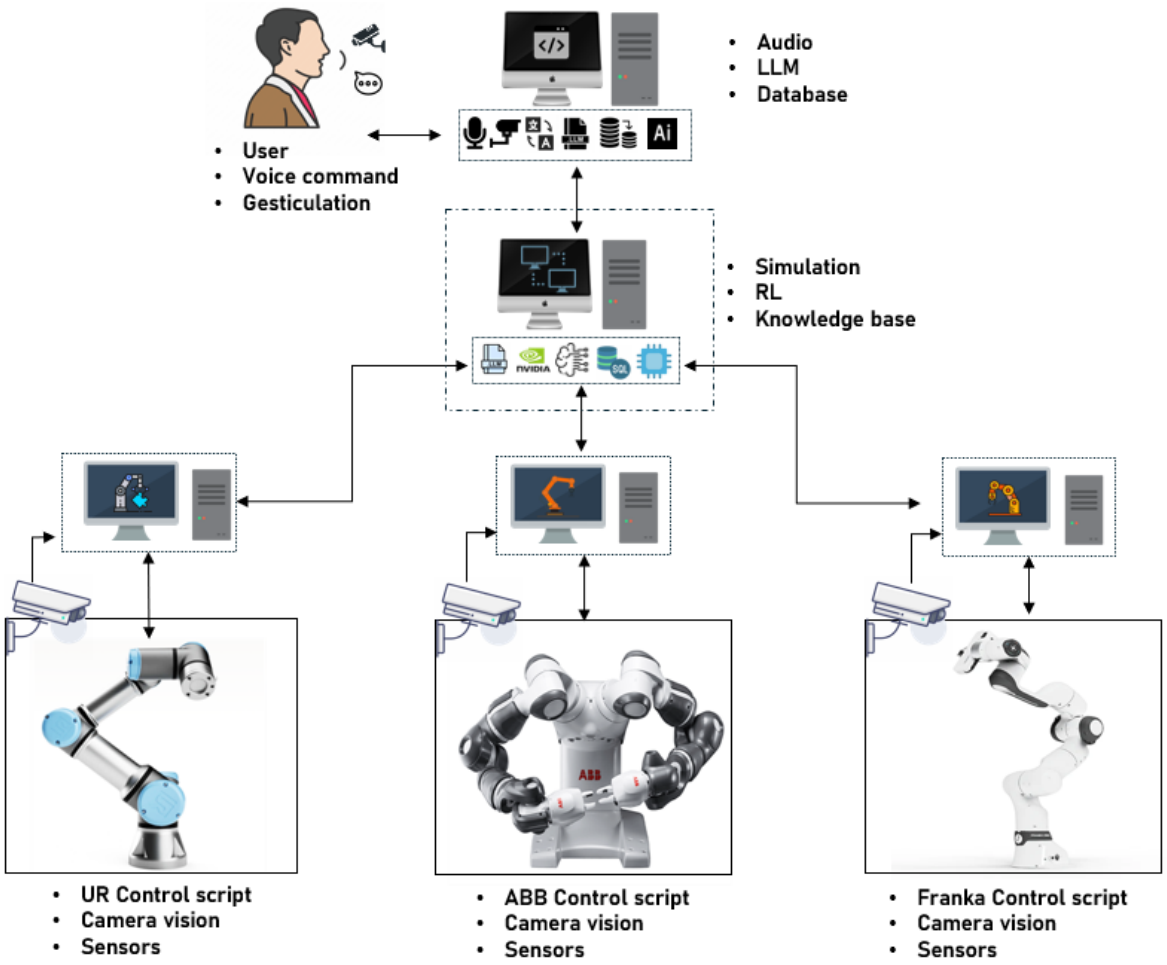



*Figure 1-2. Proposed high-level system framework*

## 1.2. Motivation

Now, consider a busy hospital ward during peak hours, where a cobot autonomously navigates hallways to deliver medications, greets nurses by name and adapts to their specific instructions. Such functionality is no longer science fiction. At CES 2025[a], prototypes of hospital-assistive robots (Figure 1-3) demonstrated significant advancements in computer vision, natural language understanding, and artificial intelligence, enabling them to learn from dynamic environments, generalize across tasks, and refine their behaviour through experience.

[a] *Consumer Electronics Show (CES) in Las Vegas [1]. Enchanted Tools, Mirokai Robots. (2025, January 5). CES Demonstration.*

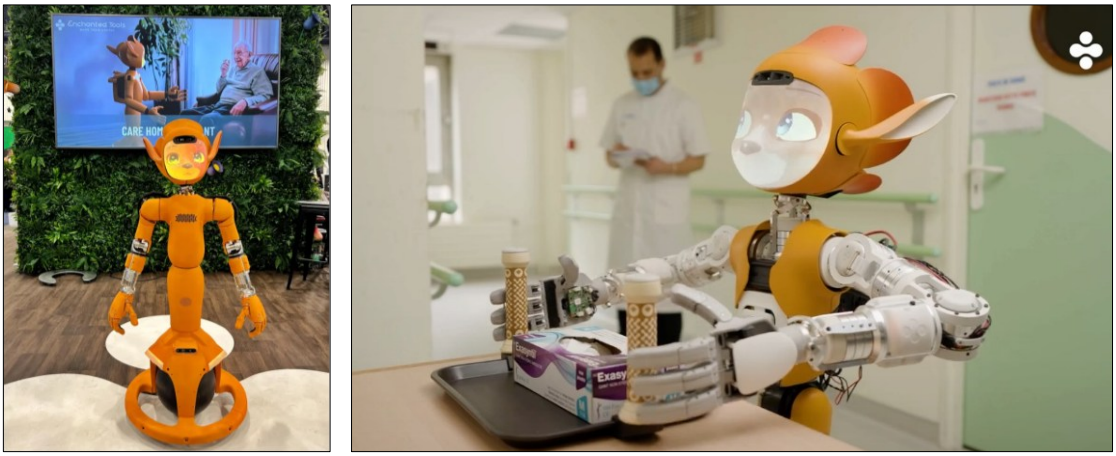


*Figure 1-3. Mirokai robot[b], CES 2025: A hospital-assistive robot*

The convergence of computer vision, natural language processing (NLP), and robotics has no doubt led to systems capable of interpreting instructions such as "*Pick up the red cup*" and autonomously executing them in physical or simulated environments [4]. As Fei-Fei Li, co-director of the Stanford Human-Centred AI Institute and creator of ImageNet, aptly puts it: *"AI is the science of making machines smart; robotics is the art of making them useful*". Her latest venture, World Labs, aims to integrate spatial intelligence into AI systems, further narrowing the gap between perception and action in robotics.

However, truly integrated, context-aware robotic systems remain rare in industrial practice. According to the International Federation of Robotics (IFR), as of 2023, collaborative robots—those designed to work alongside humans—account for only about 10% of industrial robot installations [5]. Supporting this, Eurofound (July 2024) reports that same <10% of robots installed in industrial settings are designed for human interaction, reinforcing the notion that human-interactive systems are still the exception rather than the norm [6]. Industry analysts further observe that "today, robots are still overwhelmingly programmed with traditional methods" (i.e. teach pendants or explicit code). This reveals a persistent gap between the interactive capabilities demonstrated in advanced research prototypes and the reality on factory floors, where natural language interfaces and adaptive behaviours remain largely absent.

On the research front, pioneers like Dr. Cynthia Breazeal, director of the Personal Robots Group at the MIT Media Lab, emphasize that *"the future of robotics hinges on seamless interaction with humans, both in terms of language understanding and personalised adaptation"* [7]. Her work on socially intelligent robots such as Kismet and Leonardo demonstrates the potential for robots to engage in meaningful, emotionally aware interactions through speech, facial cues, and adaptive behaviour [8], [9]. Despite this progress, most current frameworks lack personalization; robots often cannot recall individual preferences, user identities, or spatial configurations specific to a given task environment.

Moreover, recent studies suggest that approximately 73% of industrial robots remain incapable of processing unstructured, natural language commands [10]. This underscores the growing need

[b] *Source: Enchanted Tools [2], [3]*

for adaptive, user-centric robotic systems capable of learning from interactions, remembering user-specific preferences, and executing high-level instructions autonomously.

This thesis responds to that need by exploring a database-driven framework that integrates vision-based perception, LLMs, and robotic control within a virtual simulation environment. The system will be designed to interpret generalized spoken or visual instructions, personalize responses by recognizing user identity (via face or voice authentication) recall task preferences, and adjust its behaviour accordingly. By doing so, it seeks to realize the vision of a collaborative robot that is context-aware, user-centric, and capable of safely demonstrating natural language commands in simulation before real-world deployment.

## 1.3. Background

Developing intelligent robotic systems requires a multidisciplinary approach. This section outlines core technologies and concepts that underpin the proposed framework, including human-robot interaction, computer vision, language models, simulation platforms, and personalization, each contributing to enabling adaptive, user-aware robotic behaviour.

### 1.3.1. Evolution of human–robot interaction (HRI)

Industrial robots have traditionally been confined to structured environments, executing pre-programmed, repetitive tasks with high precision. In manufacturing, robotic arms have transformed assembly lines by automating operations such as welding, assembly, and material handling. Historically, these robots operated in isolation from human workers due to safety concerns and were limited to rigid motion paths, lacking any understanding of context or intent.

Today, we have transitioned into an era where robots must function in human spaces, requiring intuitive communication skills and dynamic adaptation. The past decade has seen a paradigm shift in robotics, moving from rigid automation to cobots capable of working alongside humans. This transformation has been driven by advances in three key areas: Computer vision, which enables robots to perceive their surroundings, Large Language Models (LLMs) for interpreting user commands, and Adaptive control techniques, which empower robots to adjust parameters such as grip force, trajectory precision, and timing based on user input.

Despite these advancements, and the growing capabilities of AI-powered robotics, real-world deployment remains limited. An IFR survey has reported that a meagre less than 10% of industrial robots feature advanced human interaction capabilities [5], [10]. WEF[c] has also highlighted AI-driven robotics as a central pillar for future manufacturing, forecasting a 40% increase in collaborative robotics adoption over the next two years. Yet, current systems struggle with unstructured user commands, mostly requiring rigid, structured inputs.

[c] *World Economic Forum's annual summit held in Davos-Klosters, Switzerland, in January, 2025 themed "Collaboration for the Intelligent Age" - [11]*

### 1.3.2. Vision-based perception:

For robots to interact effectively with humans then, they must be able to perceive their environment accurately. Computer vision enables robots to detect/recognize objects, track movement or dynamic changes in the environment, and infer spatial relationships.

- Object detection & tracking: Algorithms such as YOLO (You Only Look Once) can identify and locate objects in real time, enabling robots to, for instance differentiate between tools, furniture, and people [12].
- Facial & gesture recognition: Even more advanced vision models allow robots to identify users, recognize gestures or facial expressions, improving interaction and responsiveness.

However, while modern vision models can detect objects with human-level accuracy, they lack contextual understanding, which is a critical requirement for real-world robotics. For example, a robot may recognize a red cup, but does it understand that this specific red cup belongs to Alice and should be handled gently, since it's fragile? Existing vision pipelines treat all objects generically, failing to integrate personalised context.

This thesis tries to address this limitation by integrating vision models with user-specific database memory, enabling a foundational framework for robots to learn and adapt to individual users over time.

### 1.3.3. Language understanding with LLMs

Advancements in natural language processing (NLP) have enabled systems that understand and respond to human instructions, utilizing Large Language Models (LLMs) such as OpenAI's GPT, Google's BERT[d], and Meta's LLaMA.

These models can:

- Parse unstructured human commands (e.g., *"Sort the tools like last time."*)
- Translate intent into structured task plans (e.g., *"Pick the wrench and place it on the tray."*)
- Adapt responses based on context and prior interactions

Notably, GPT-4.5 from OpenAI [14] excels at recognizing patterns, drawing connections, and generating creative insights without relying on traditional reasoning. However, LLMs are not inherently grounded in physical reality. Bridging the high-level instructions and the low-level actuator commands required for robotic control is non-trivial, often likened to "*teaching a toddler calculus*" [15]. So, while LLMs excel at understanding human commands, they struggle with real-world task execution.

By exploring the integration of LLM-driven planning with real-time visual perception, this research can enable robots to generate actionable, context-aware task plans, bridging the gap between language comprehension and robotic control.

[d] *from the revolutionary paper by Google that increased the State-of-the-art performance for various NLP tasks and set the stepping stone for many other revolutionary architectures [13].*

### 1.3.4. Robotic control in simulation environments

Achieving safe and reliable robot deployment requires realistic, physics-accurate simulation environments. Modern robotic frameworks rely on:

- The Robot Operating System (ROS): A standardized communication framework for robot control and motion execution [16].
- Simulation platforms like NVIDIA Omniverse: Providing photorealistic virtual grounds for testing and validation, reducing the risks and costs of real-world deployment.

Robotic frameworks like ROS have standardized how robots receive commands and execute motions, while Simulation platforms like NVIDIA Omniverse Isaac Sim provides a high-fidelity physics-based photorealistic simulation environment for testing robotic systems before real-world execution [17]. The synergy between these two elements can be likened to a "*dress rehearsal*," ensuring that each motion plan is viable before it goes live.

Much like a flight simulator for pilots, a virtual environment serves as a risk-free testbed for robots, where errors result in iteration, not injury or equipment damage. Developers can rapidly prototype, debug, and optimize AI-driven task execution pipelines in dynamic scenarios, such as warehouse organization or delicate assembly operations, without downtime or material costs.

Reinforcement Learning (RL) [18] and sim-to-real transfer [19] further enhance the robustness of robot control schemes. Nonetheless, the "*Reality Gap*" remains a challenge we must carefully account for. Simulations do not always perfectly reflect real-world physics.

At GTC[e] 2023, NVIDIA unveiled Omniverse's "AI Gym" for training robots—an important step toward addressing the Sim2Real (simulation-to-reality) gap. By leveraging photorealistic physics, GPU-accelerated simulation, and multi-agent AI tooling, platforms like Isaac Sim enable iterative refinement of robotic behaviours. Thus, robotic systems can evolve from *"Error: Cup not found"* to personalised confirmations like *"Coffee's ready: two sugars, as usual."*

### 1.3.5. Personalization and ethical implications

With greater personalization also come ethical and societal considerations. Robots that store user data, such as facial recognition profiles and interaction histories, must be subject to robust privacy safeguards. Google's 2025 robotics privacy framework [22] explicitly recommends minimizing collection of personally identifiable information (PII), and clear transparency about sensor capabilities (cameras, microphones, etc.). As highlighted by a panel at the 2025 iREX (International Robot Exhibition) in Tokyo, "*A robot that recognizes your face shouldn't become a gossip*." This quote underscores the importance of secure data handling and respectful human-robot rapport. This thesis embeds user data in secure databases and would advise best practices for compliance, ensuring that personalization does not undermine trust or privacy in any way.

The upcoming iRex 2025[f], themed "*Sustainable Societies Through Robotics*," further highlights the growing impact of robotics in industry and society with a strong emphasis on human-robot

[e] *GPU Technology Conference 2023 hosted by NVIDIA; an annual global AI conference for developers, engineers, researchers, inventors, and IT professionals, gather to discuss the latest innovations in AI, deep learning, graphics, & high-performance computing [20], [21].*
[f] *International Robot Exhibition hosted once every two years in Tokyo, Japan since 1973, and highly regarded as one of the largest robot exhibitions in the world.*

collaboration. As experts advocate for secure data practices and ethical AI, the event reinforces a critical consensus: facial recognition-enabled robots must integrate privacy-by-design principles to safeguard user data and prevent misuse.

Existing research often focuses on isolated components—vision-only models, NLP-driven robotics, or motion planning in controlled environments. However, true AI-powered robotics requires a seamless integration of these elements. Challenges may include:

1. Multi-modal synchronization – merging vision inputs, language processing, and control commands into a cohesive system.
2. Personalization & user adaptation – ensuring robots can learn user preferences over time and recall past interactions.

This thesis tries to explore a new approach, unifying vision, language, and control into a single pipeline, while incorporating a database-driven personalization layer, turning robots from one-size-fits-all tools into collaborators that remember your name, preferences, and past interactions.

## 1.4. Aims & Goals

The overarching aim of this thesis is to develop an intelligent, user-centric framework that integrates computer vision, large language models (LLMs), and robotic control for personalised, context-aware robotic task execution in virtual environments, thereby enhancing human-robot collaboration.

**Specific goals:**

- Develop a vision pipeline for real-time object detection and user identification to enhance task execution accuracy.
- Implement an LLM-powered interface that translates natural language commands into structured, executable task plans.
- Enable the validation of the framework in NVIDIA Omniverse, simulating real-world robot tasks using an industrial robot (ABB YuMi or Universal Robot arms).
- Design a database-driven personalization module, enabling robots to store and recall user preferences and executed tasks for improved interaction quality.

## 1.5. Research Questions

This thesis aims to answer the following key research questions:

RQ1: How can computer vision and large language models be synergistically integrated to improve task planning and execution accuracy in robotic systems?

RQ2: How can virtual simulation environments (e.g., NVIDIA Omniverse) be leveraged to validate and refine AI-driven robotic control?

RQ3: To what extent does the personalised interaction, using stored user-specific data, enhance task execution efficiency and user satisfaction compared to a generic, non-personalised robotic systems?

## 1.6. Limitations & Delimitations

While this thesis aims to advance AI-driven robotics, certain limitations must be acknowledged. These limitations highlight areas where additional optimization, validation, or ethical considerations are necessary.

### 1.6.1. Limitations (external constraints)

1. Interpretability of LLM-based decision making – Large Language Models operate as black-box systems, meaning their decision-making processes are often opaque and difficult to debug. This can lead to unpredictable task plans, necessitating manual validation before execution to ensure safety and efficiency.
2. Computational constraints – Running LLMs, vision models, and real-time control systems requires significant computational resources. High inference times may impact real-time responsiveness, necessitating optimization strategies such as model pruning and distributed processing.
3. Multi-modal integration challenges – Combining computer vision, NLP, and robotic control into a single framework presents inherent synchronization challenges. Ensuring that vision-based perception, LLM-based reasoning, and robotic execution align in real time requires efficient data flow management and low-latency communication protocols.
4. Dataset bias in facial recognition – Facial recognition models, trained on specific datasets, may exhibit biases in user identification, impacting accuracy across different demographics. Addressing this requires dataset diversification and bias-mitigation techniques to ensure fairness in user recognition.
5. Ethical and privacy considerations – Personalization requires storing user interaction data, raising concerns over data privacy and security. To comply with GDPR and ethical AI principles, this research employs local data storage, access control mechanisms, and anonymization techniques to protect user information.

### 1.6.2. Delimitations (strategic choices)

To ensure focus and feasibility, this study intentionally limits its scope in the following ways:

1. Focus on pick-and-place tasks – The research prioritizes object manipulation and task planning, rather than highly dexterous or complex assembly tasks requiring fine motor skills.
2. Simulation-first approach – Instead of real-world deployment, this thesis validates robotic behaviour in NVIDIA Omniverse, leveraging virtual testing before physical trials.
3. Pre-trained LLMs instead of custom training – Rather than training new language models, the study leverages existing pre-trained models (e.g., Mistral, LLaMA, DeepSeek, or similar open-source models), applying prompt engineering and fine-tuning techniques for robotic task execution.
4. Controlled user study – The evaluation phase includes a small sample size (3-10 users) to test personalization features. A larger-scale validation study is planned as future work.

5. Privacy-centric design – While the system personalizes robotic interactions, it avoids storing sensitive biometric data long-term, ensuring compliance with ethical AI guidelines.

Despite these constraints, this thesis provides a scalable approach to explore context-aware human-robot interaction, offering a foundation for future research or real-world implementation.

## 1.7. Expected Outcomes

This research aims to deliver the following tangible and measurable outcomes:

1. A functional prototype demonstrating an AI-driven robotic task execution using natural language commands and vision-based object detection.
2. A scalable integration framework combining computer vision, large language models (LLMs), and robotic control for personalized task execution in virtual environments.
3. A database-driven personalization module (PostgreSQL-based) enabling the system to recall user preferences, biometric profiles, and historical interaction data.
4. Quantitative and/or qualitative evaluation metrics to assess the system performance.

**Planned deliverables**

- A functional pipeline combining computer vision, LLMs, and robot control scripts, capable of interpreting high-level user commands and validating task plans via Isaac Sim.
- A unified vision-language module that translates multi-modal inputs into structured task plans for simulation and execution.
- A public GitHub repository containing implementation code, documentation, and configuration scripts for reproducibility.
- A user interface supporting real-time interaction with the system through voice, text, and biometric-based user sessions, enabling practical demonstrations or remote testing.
- A GUI-integrated personalization module that allows users to manage profiles, interaction history, and task preferences via the database.
- Structured logging mechanisms for capturing task sequences and personalized vocabulary, enabling adaptive behaviour and system tuning over time.

## 1.8. Key Topics

The key research topics or search terms include:

**Vision-language integration in robotics:** Vision-language models in robotics bridge object detection (e.g., YOLO) and command parsing (e.g., GPT-4), enabling robots to interpret commands like 'pick up the red cup' both visually and linguistically.

**LLMs for robotic command interpretation (e.g., GPT, BERT):** Large Language Models (LLMs) like GPT-4 and BERT are used to parse and interpret natural language commands, translating high-level instructions (e.g., *"organize my desk"*) into structured task plans. These

models enable robots to manage ambiguous or context-dependent commands, making human-robot interaction more intuitive and efficient.

**Simulation platforms for robotic validation (Omniverse, Gazebo):** Simulation platforms like NVIDIA Omniverse and Gazebo provide virtual environments for testing and validating robotic systems. These platforms allow researchers to refine control policies, assess task execution, and reduce the reality gap (differences between simulated and real-world performance) before deploying robots in physical environments.

**Human–robot interaction (HRI) personalization techniques:** Personalization techniques in HRI involve tailoring robotic interactions to individual users based on stored profiles, preferences, and interaction histories. For example, a robot might remember a user's preferred coffee order or frequently used objects, enabling more efficient and context-aware task execution.

**Task planning and control for collaborative robots**: Task planning and control for cobots involve generating executable task plans from high-level commands and translating them into low-level control signals. This process ensures that robots like ABB YuMi or Universal Robots can safely and efficiently perform tasks in dynamic, human-centric environments.

## 1.9. Thesis Outline

This thesis is structured into six chapters:

- **Chapter 1** – Introduction:
  Defines the problem context, research motivation, objectives, and expected outcomes.
- **Chapter 2** – Literature Review:
  Reviews key work in human-robot interaction, vision, language models, simulation, and personalization.
- **Chapter 3** – Methodology:
  Describes the development approach, tools, and frameworks used in building the system.
- **Chapter 4** – System Architecture & Design:
  Details the modular system design, core components, data flow, and orchestration strategy.
- **Chapter 5** – Results & Discussion:
  Presents system integration outcomes, performance evaluation, and user interaction insights.
- **Chapter 6** – Conclusion & Future Work:
  Summarizes findings and proposes directions for further development and real-world deployment.

# 2. Literature Review

This chapter reviews existing research on human-robot interaction (HRI), computer vision, large language models (LLMs), database-driven task execution, and simulation frameworks like NVIDIA Omniverse Isaac Sim. It contextualizes the research problem, identifies gaps, and establishes a theoretical foundation for integrating vision, language processing, and robotic control into a unified framework. The review covers personalised robotics, vision-based perception, LLM advancements, database-driven execution, and simulation frameworks for validating robotic control before deployment.

## 2.1. Theoretical Framework

In recent years, the field of robotics has undergone a transformative evolution, driven by rapid advancements in artificial intelligence (AI), particularly in the domains of computer vision and natural language processing (NLP). These technologies have enabled the development of robotic systems that can perceive their environment with remarkable accuracy, understand and respond to human language, and execute complex tasks autonomously. These capabilities are essential for applications such as industrial automation and healthcare, as well as domestic assistance and education, where robots must interact naturally and efficiently with humans. However, the true potential of robotics lies not only in the individual strengths of these technologies but in their seamless integration. Combining computer vision, large language models (LLMs), and advanced robot control techniques, modern robotics can create intelligent systems that understand user intents, adapt to individual preferences, and perform tasks in a personalised and context-aware manner.

Through this theory, we aim to identify gaps in current research - such as limitations in real-time integration, scalability of personalization, and fidelity of simulation environments - and establish a theoretical foundation for the proposed thesis. By synthesizing foundational and up to recent research findings from various domains, this chapter builds a case for the proposed research, highlighting the necessity of integrating vision, language processing, and robot control in a modular framework for personalised task execution to enhance human-robot collaboration.

## 2.2. Evolution of Cobots: from Scripted to Adaptive Systems

The quest to create robots capable of natural human interaction has progressed through *three* distinct eras [23]. Early industrial robots operated in strictly structured environments following rigid pre-programmed routines of commands, with minimal human intervention [24]. While the second-generation systems incorporated basic sensor feedback for adaptive grasping [25], they

still followed preprogrammed trajectories with limited adaptability. These robotic systems were traditionally designed for efficiency in manufacturing lines, such as robotic arms used in car assembly plants.

The current cognitive robotics paradigm, marked by the emergence of collaborative robots (cobots), such as ABB's YuMi and Universal Robots' UR series, marked a significant shift in robotics. Unlike traditional industrial robots, cobots are designed to safely interact with humans, featuring real-time sensors, adaptive controls, and AI-driven decision-making. Research in HRI [26], emphasizes intuitive interactions, enabling robots to understand and respond to user intentions using speech, gestures, and vision-based cues.

The AI revolution is ushering in a new era in robotics. Embodied AI integrates intelligent algorithms into physical systems, enabling robots to perceive and interact with their environments through dynamic movements. Three primary robotic systems have emerged—rule-based, training-based, and context-based (Figure 2-1). These agents use sensors (e.g., cameras, microphones) to observe the world and actuators like advanced grippers to perform actions. Applied to industrial operations, these agents enable more sophisticated automation overcoming traditional challenges such as those associated with handling unstructured environments or manipulating unstable objects [27].

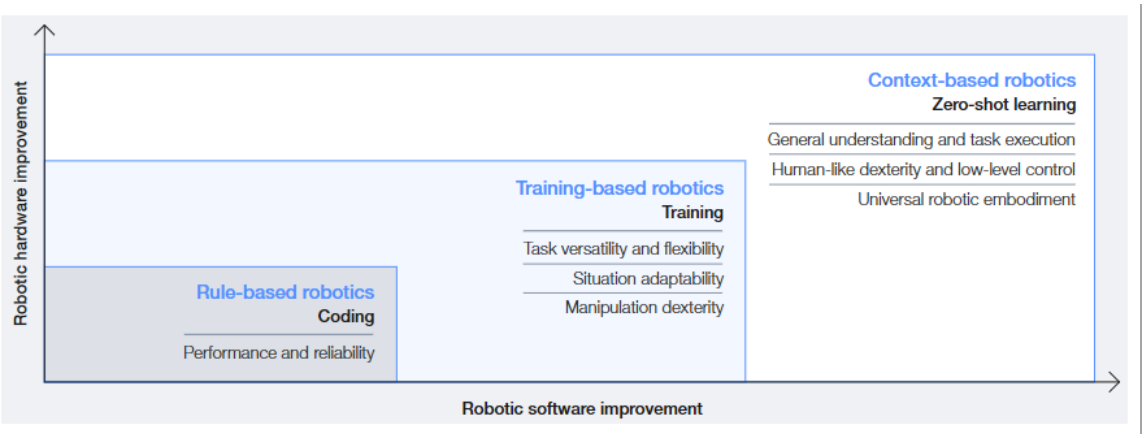



*Figure 2-1. Robot capabilities enabled by AI.* - Boston Consulting Group

Furthermore, recent studies highlight the need for personalization in HRI, where robots not only follow generalized instructions but also understand contextual nuances and adapt to individual users' preferences, roles, and past interactions [28].

The evolution from rigid industrial automation where robots performed repetitive tasks in isolated settings, to knowledge-driven personalised robotics, where cobots work alongside humans in dynamic environments have proven crucial in domains such as assistive healthcare [29], collaborative manufacturing, and household robotics.

## 2.3. Vision Systems: Object Detection to User-Centric Context

> *"A camera that sees a 'red box' is useful; one that recognizes 'John's fragile red box' is revolutionary."*

Modern robotic vision systems excel at detecting objects but remain oblivious to the *who* and *why* behind tasks. This section critiques the field's fixation on generic perception and argues for a paradigm shift toward user-centric context

### 2.3.1. Object Perception through RGB-D Cameras

Modern robotic vision systems have evolved from static object detection pipelines to dynamic scene interpreters. These advancements in real-time object detection (YOLO11) [30], [31], [32] and pose estimation (OpenCV, NVIDIA DeepStream) have transformed how robots interpret their environments. Algorithms like You Only Look Once (YOLO) have set benchmarks by processing images at remarkable speeds - *YOLOv6-3.0 achieves 142 FPS inference speeds on NVIDIA Jetson platforms while maintaining 52.3% mAP accuracy* [31]. Such real-time performance enables cobots like ABB YuMi to process 360° RGB-D streams at 30Hz, detecting objects within ±3mm positional accuracy, for identification and location of objects in applications ranging from warehouse logistics to surgical robotics [30], [33]. However, raw detection metrics only tell half the story.

Pose estimation further enhances a robot's interaction with its surroundings by determining the orientation and position of objects. Techniques utilizing OpenCV facilitate real-time pose estimation, allowing robots to infer not just *what* objects exist, but *how* they relate spatially - a prerequisite for executing tasks requiring precision and adaptability.

Despite these technological strides, many vision pipelines perceive objects as generic entities, devoid of user-specific meaning.

> *"A 'red box' detected by YOLO lacks context - is it fragile? Who owns it? How was it handled last time?".*

Amazon's Astro robot exemplifies this limitation: while it can recognize faces, it doesn't adjust its tasks based on the user's expertise or history [34].

Object detection [35] affordance recognition [36], [37] and facial recognition such as in [38] are all crucial for contextual understanding: the first helps locate items, the second infers how to interact with them, and the third enables personalised user interactions. While these capabilities have advanced individually, integrating them into a cohesive robotic vision pipeline, especially in dynamic and occluded environments, remains an ongoing frontier in research.

By integrating *scene data* with *database-driven user identity*, a system can enrich object metadata with user history. For instance, detecting '*John's red box'* triggers a pre-stored preference for low-force grip handling, avoiding the one-size-fits-all approach of existing frameworks.

### 2.3.2. User-Specific Perception via Biometrics

Biometric authentication -face recognition and voice processing- has matured into a reliable tool for user identification, from security checkpoints to collaborative robotics. Yet, some current implementations treat biometrics as mere "keys" to access systems, failing to leverage them for *collaborative* or *adaptive* applications.

By employing face and voice recognition technologies [39], robots can identify users and access individualized profiles, paving the way for customized responses. For example, Sensory Inc. has

developed embedded face and voice biometrics that provide secure and convenient user verification, enhancing the personalization of interactions [39].

By linking biometric authentication to a PostgreSQL database of user profiles, robots can recall past interactions and preferences. For instance, recognizing 'Rahul' via face authentication retrieves his stored preference for slower pick-and-place task execution, while voice commands like 'Handle carefully' update the robot's torque limits in real time. This fusion of biometrics, scene data, and historical context enables truly personalised execution.

In summary, advancing from basic object detection to user-centric context requires integrating sophisticated scene understanding with biometric-driven personalization. While existing tools like OpenCV or ROS's vision pipelines excel at *what* and *where*, a database-driven architecture could answer *for whom* - a leap from robotic perception to collaborative understanding.

## 2.4. Language Models: Translating Intent into Actionable Code

> *"What if Siri could not only set a timer but also teach a robot to assemble your IKEA furniture?"*

### 2.4.1. Natural Language Processing in Robotics

The advent of transformer-based large language models (LLMs) like GPT-4, BERT, and LLaMA for natural language processing (NLP), reasoning and language generation has redefined human-robot interaction. These models act as neural compilers, translating unstructured linguistic inputs into structured task representations through three computational phases: *syntactic parsing* (identifying verbs/nouns), *contextual grounding* (e.g., mapping "left panel" to CAD-defined part IDs), and *code generation* (producing ROS-compatible motion [40]. Recent work demonstrates their surprising adaptability: *ChatGPT*, designed for text generation, now generates robot control code [41], while domain-specific models like *PaLM-E* integrate vision and language for embodied reasoning [42]. Yet, as the *2023 LLM HRI Review* notes, these systems often operate in isolation - parsing language without grounding it in real-time sensor data or user history.

### 2.4.2. Challenges of LLMs in embodied robotic systems

Despite their prowess, LLMs face three unique challenges in embodied robotic systems:

**Ambiguity resolution in instructions**

LLMs often struggle to interpret ambiguous or underspecified human instructions in a robotics context. Different LLMs can produce divergent behaviours when given the same ambiguous command, highlighting inconsistency in understanding intent [43].

For example, an instruction like *"put the block in the bowl you think it best fits"* leaves too much subjectivity, one model might choose a different bowl than another. This issue is often described as a problem of *abstraction matching*, meaning the robot's system must align an ambiguous natural-language request with a precise, unambiguous action sequence [44].

In practice, current LLM-driven robots still find it challenging to reliably disambiguate instructions without additional contextual clues or clarifications, underscoring the need for better grounding of language in robotic perception and knowledge. Proper handling of ambiguity is crucial; otherwise, the robot's actions may not match the user's intent due to misinterpretation.

### LLMs and temporal context in robotics

LLMs often struggle with real-time processing requirements and the integration of multimodal sensory inputs, such as visual and tactile data, which are essential for accurate task execution in robotics.

> *While chatbots retain conversational memory (e.g., ChatGPT's chat history), robotic systems must also track state changes across interactions.*

Robotic tasks typically unfold over a sequence of steps and time, which poses problems for LLM-based controllers in maintaining temporal context. One limitation is that LLMs generate actions in an auto-regressive (step-by-step) manner, errors can compound over time. A mistake or misunderstanding in an early step becomes part of the context for subsequent steps, potentially derailing the entire plan.

Moreover, studies have found that LLM-generated plans sometimes execute actions out of order or with improper sequencing, reflecting difficulty in reasoning about event order. For instance, a robot might skip a necessary preparatory action or perform an irrelevant step because the model failed to connect it to the earlier context. This indicates that while LLMs can break down tasks into steps, they do not always capture the correct temporal dependencies. Ensuring that a robot understands what to do *first*, *next*, and *last* (and how to adjust if something changes mid-task) remains an open challenge. Researchers are addressing this by incorporating feedback loops and temporal logic checks, but LLM-based systems still require improvements in context retention and temporal reasoning to handle long-horizon, dynamic tasks reliably.

### Actionability and grounding of language

Another significant challenge for applying LLMs in embodied robots is translating language output into feasible, real-world actions.

> *LLMs "hallucinate" - a phenomenon where LLMs generate plausible-sounding but incorrect or unfeasible actions.*

This issue arises because LLMs, trained predominantly on text data, may lack grounding in the physical realities of robotic capabilities and environmental constraints. For example, a plan might instruct the robot to navigate to a non-existent location or pick up an object that isn't present.

Such instructions reveal a lack of grounding; the model does not fully grasp the robot's physical context or limitations. Even when the described actions are theoretically possible, they might not be *directly executable* without further translation. For instance, an LLM might output a high-level goal (*'tidy the room'*) or a vague action (*'fix the disorganized shelf'*) that the robot cannot act on without decomposition into its low-level motor commands. Researchers have noted that LLM-based planners need interfaces (such as skill libraries or code-generation layers) to convert abstract language into concrete robot behaviours, but this translation is brittle and often requires careful prompt design or model fine-tuning.

In summary, ensuring *actionability* means the robot's interpretation of an LLM's output aligns with what it can do in its environment. Overcoming this gap involves better world-model grounding for LLMs and stricter checks on the feasibility of suggested actions, so the robot doesn't attempt steps outside its capabilities or environment state.

### 2.4.3. Conversational memory: “chatbots to cobots”

In the realm of chatbots, conversational memory enables systems to maintain context over interactions, providing coherent and contextually appropriate responses. Translating this capability to collaborative robots (cobots) involves more than just retaining dialogue history; it requires the integration of past interactions, user preferences, and environmental context into the robot’s task planning and execution processes. This deep integration allows cobots to anticipate user needs, personalize task execution, and improve efficiency over time. For instance, a cobot equipped with conversational memory can recall a user’s preferred methods for assembling furniture, thereby streamlining the process in future tasks. Implementing such systems demands robust data management and real-time processing frameworks to handle the complex interplay of language, memory, and action in dynamic settings.

In summary, standalone LLMs remain *language-bound* , their brilliance in text generation often falters when confronted with the *physics-aware* demands of robotic systems. This gap is bridged through hybrid architectures that fuse LLM outputs with real-time visual data and user-specific context, enabling robots to adapt to dynamic environments and user-specific preferences, moving beyond generic responses to tailored solutions.

## 2.5. Robot Task Planning & Execution in Virtual Environments

### 2.5.1. Robotic control in simulated environments

Virtual environments have become indispensable tools for designing, testing, and validating robot control strategies prior to real-world [45]. These platforms offer photorealistic and physics-accurate simulations, reducing the necessity for physical prototyping and accelerating iteration cycles in robotics research. For instance, NVIDIA Omniverse Isaac Sim provides a scalable and extensible simulation environment that supports various sensors and robot models, enabling comprehensive testing of AI-driven robotics solutions.

A virtual environment is a computer-generated, interactive 3D space used for visualization, immersion, or user interaction. In contrast, a simulation environment models real-world systems over time, focusing on physics, logic, and behavioural accuracy. While virtual environments emphasize realism and interface, simulations prioritize functional modelling and sensor dynamics. In robotics, platforms like Isaac Sim combine both, enabling immersive, physics-accurate testing of robot behaviour in realistic digital settings.

In [46], the authors propose a flexible simulation framework that supports real-time validation and task adaptation for robots operating in unstructured settings. Their work demonstrates how simulation-based planning, coupled with digital twin synchronisation, can enhance system robustness and reduce execution risk—principles that are also central to the design of the current system.

#### Challenges from use of simulation tools

Simulation is the process of using a computer to approximate how a dynamic system (here a robot) evolves in time, and as has been poignantly observed, “Simulation is doomed to succeed”,

[47], [48] provides an overview of the concept and role of physics-based simulation in designing intelligent robots.

Afzal and his colleagues [49] have identified 10 high-level challenges associated with the use of simulation and outlined ideas on how to tackle these challenges to unlock the full potential of simulation-based testing. These challenges include the gap between simulation and reality, a lack of reproducibility, and considerable resource costs associated with using simulators.

### 2.5.2. Modular architectures for robotic planning

Modern robot control architectures increasingly favour modular frameworks that separate perception, planning, and actuation. Within such architectures, the control component is typically responsible for generating collision-free trajectories and closed-loop feedback corrections based on sensor inputs. Open-source software like the Robot Operating System (ROS) and its MoveIt framework exemplify this modular principle, offering a suite of libraries for motion planning, inverse kinematics, and real-time control. Recent advancements have extended these concepts into GPU-accelerated simulation environments, such as NVIDIA Isaac Sim, to support faster-than-real-time iterations of planning and control strategies.

### 2.5.3. Task decomposition & execution using hybrid frameworks

A fundamental aspect of robotics research involves ensuring that robots can break down high-level goals into discrete, executable actions - colloquially referred to as task planning and execution. Classical AI planners like STRIPS [50] laid the groundwork for symbolic planning; more recent efforts integrate classical symbolic methods with modern AI techniques, particularly Large Language Models (LLMs) to enhance task planning by translating natural language instructions into symbolic representations, thus improving the generalization to new tasks, addressing challenges in task planning and execution in unstructured environments [51].

This integration of multimodal integrated systems allows robots to perform complex actions in unstructured environments, demonstrating high success rates in various task complexities (Silva et al., 2024). A recent literature [52] proposes a TAsk Planing Agent (TaPA) in embodied tasks for grounded planning with physical scene constraint, where the agent generates executable plans according to existing objects in the scene by aligning LLMs with the visual perception models.

The role of large language models (LLMs) in interpreting user intent and generating executable plans was explored, along with the challenges of grounding natural language in physical environments. Robotic control strategies were reviewed, particularly in relation to virtual testing and planning through simulation platforms like NVIDIA Isaac Sim. Additionally, this chapter highlighted modular architectures and the growing importance of database-driven personalisation and task orchestration frameworks.

Overall, the literature underpins the architectural choices made in this project, affirming the value of multimodal input, digital twin integration, and LLM-based reasoning in building flexible, user-aware robotic systems.

# 3. Methodology

This chapter describes the methods used to develop the system. It begins with the development strategy (an LLM-assisted RAD approach) and introduces the tools, libraries, and frameworks employed across subsystems. Emphasis is placed on the modularity, data flow, and integration.

## 3.1. Development Approach

The system was developed using an iterative methodology aligned with the principles of Rapid Application Development (RAD). RAD is an adaptive software development model that emphasizes rapid prototyping, iterative user feedback and minimal upfront planning. Unlike traditional *waterfall* approaches, RAD facilitates quick creation of functional prototypes by leveraging reusable components and enabling continuous refinement based on user feedback. This involves parallel development of different modules or functionalities, often using time-boxed iterations [53].

As illustrated in Figure 3-1, RAD comprises four distinct yet iterative phases: *requirements planning*, *user design*, rapid *construction*, and *cutover* (implementation). This methodology is well-suited for projects that require flexibility and frequent iteration, especially where requirements are likely to evolve.

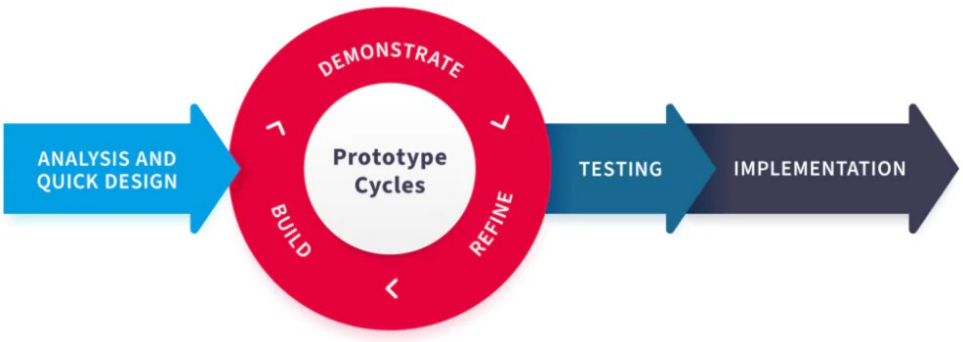



*Figure 3-1: Implementing the four phases of RAD*

Prototyping, visual feedback, and continuous testing were employed to incrementally develop core functionalities across subsystems such as vision, language understanding and robot control. Additionally, an LLM-assisted strategy (*informally termed* "*vibe coding*") was adopted to accelerate implementation and documentation. This strategy leveraged AI-powered tools including *GitHub Copilot* and *CodeGPT*, to critique, refine, and generate code from natural language prompts.

The integration of *vibe coding* into the development process enabled a fluid and dynamic interaction with the codebase, enhancing both development efficiency and the author's

engagement with the system. This combined methodology significantly improved the speed and effectiveness of the project, allowing for greater focus on system logic and architectural design rather than low-level syntax and debugging.

## 3.2. Tools and Frameworks Utilised

Table 3-1 summarizes the primary tools and libraries adopted across different subsystems.

*Table 3-1: Key tools and libraries by system component*

| System component | Tools and libraries |
|---|---|
| Scripting | Python 3.10+, Visual Studio Code/Cursor, Git for version control (GitHub) |
| Vision & perception | OpenCV, Intel RealSense D453i depth camera, face_recognition |
| Language understanding | Open-source LLMs (via Ollama), LangChain, LangGraph, porcupine, OpenWakeWord, fasterWhisper, sounddevice[g], MediaPipe[h] |
| Authentication | face encoding, voice embeddings, FAISS (vector search), Resemblyzer[i] |
| Database & memory | PostgreSQL (via psycopg2), structured schemas, multithreading support |
| Simulation & control | Omniverse Isaac Sim, ABB YuMi digital twin interface via ROS |
| GUI & Interfaces | FastAPI (for web interface), Ngrok, HTML/CSS |
| Configuration & logging | .env and YAML-based config files, centralized logging modules |

### 3.2.1. Vision & perception tools

Computer vision, a foundational component of the system, leverages *OpenCV*[j] library for real-time object detection and contour extraction, while the *face_recognition*[k] library was used for face localization. Object detection capabilities are enhanced by the *Intel RealSense D453i* depth camera (see *Figure 3-2*), which captures both RGB and depth images alongside data from an integrated inertial measurement unit (IMU) that helps track movement. The D453i's wide field of view and ability to perform well in low-light conditions make it particularly suitable for real world applications. Detected object attributes such as colour, shape, and position are continuously processed and updated on the database, enabling precise robot task planning and execution.


*Figure 3-2: RealSense D435i depth camera used for the project*

[g] *sounddevice library documentation. Available at: https://python-sounddevice.readthedocs.io/*
[h] *MediaPipe's hand tracking module for gesture recognition. Available at: MediaPipe Gesture Recognizer*
[i] *resemblyzer library documentation. Available at: https://pypi.org/project/Resemblyzer/*
[j] *OpenCV Documentation. Available at: https://opencv.org/*
[k] *A simple tool for face recognition and manipulation in Python using webcam. https://github.com/ageitgey/face_recognition*

For user recognition using user webcam, *face_recognition* is employed in conjunction with *FAISS*[l] for vector-based face embedding matching. This hybrid approach ensures fast and accurate identification even under minor appearance variations, forming a core pillar of the secure and personalized human-robot interaction flow.

### 3.2.2. Language models

The system uses open-source large language models (LLMs) as its core for understanding and generating responses. These models run through *Ollama*[m]—a simple framework for running and managing LLMs on the local machine. Different models like *llama3.3:latest*, *gemma3:4b*, *mistral:latest*, and *phi4:latest* can be selected based on task complexity, context requirements, and latency constraints. As compared in Figure 3-3, the larger models are better at handling nuanced or multi-step instructions due to their superior reasoning capabilities, while smaller models respond faster for simpler tasks or interactions.

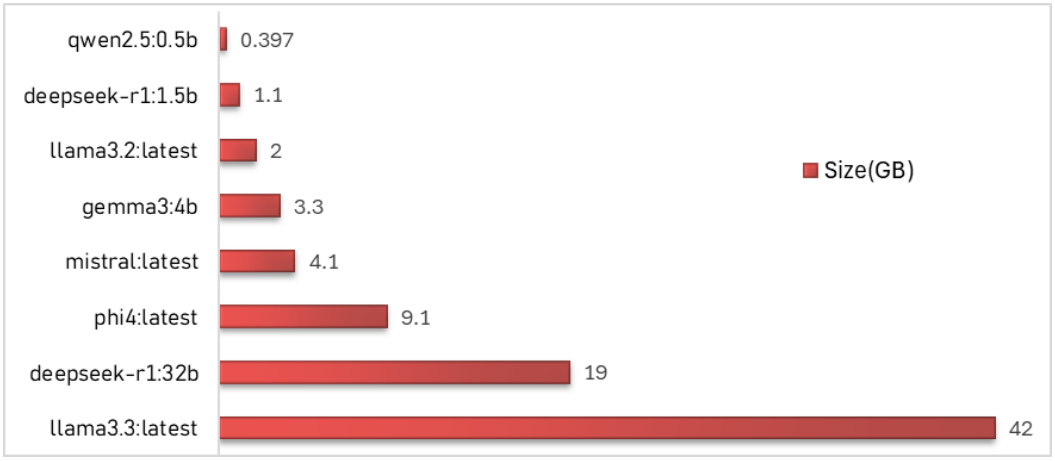



*Figure 3-3: Size comparison of some LLMs deployed*

These models parse multi-modal inputs (voice, optional gestures) into structured task plans, which are stored as executable robot actions/operations in the database.

Voice commands are captured and processed using *fasterWhisper*[n], which transcribes audio to English using beam search and fallback logic. Faster-Whisper is an implementation four times faster than *OpenAI/whisper* for the same accuracy while using less memory, enabling real-time vocal command parsing even in noisy industrial environments.

Wake-word detection is handled using *Porcupine* or *OpenWakeWord*, enabling passive listening across sessions. Prompt templates for task classification, scene description, and conversational responses are dynamically constructed using a custom *PromptBuilder* module powered by *LangChain*.

*LangGraph*[o] serves as the orchestration engine, routing user intent through dedicated subgraphs, such as *ActionTask*, *SceneQuery*, or *GeneralQuery* based on a classifier node. Custom prompt templates ensure reliable grounding and precision in command-to-task translation.

[l] *Facebook AI Similarity Search. Available at: https://github.com/facebookresearch/faiss*
[m] *Ollama: A lightweight, open-source framework for running and building large language models locally. https://ollama.com/*
[n] *Faster Whisper: Faster implementation of OpenAI's Whisper model. Available at: https://github.com/openai/whisper*
[o] *LangGraph is an open-source framework for building stateful AI agentic systems. https://github.com/langchain-ai/langgraph*

### 3.2.3. Simulation tools

Nvidia Isaac Sim is used as the simulation environment, with a ROS-integrated ABB YuMi digital twin enabling real-time testing of object manipulation tasks. Its physically accurate engine allows for validation of pick and place tasks in controlled environments.

This simulation layer not only supports safe pre-deployment testing but also assists in synchronizing the digital and physical robot states, ensuring consistency in execution. Its versatility and extensibility make it an attractive choice for research and development.

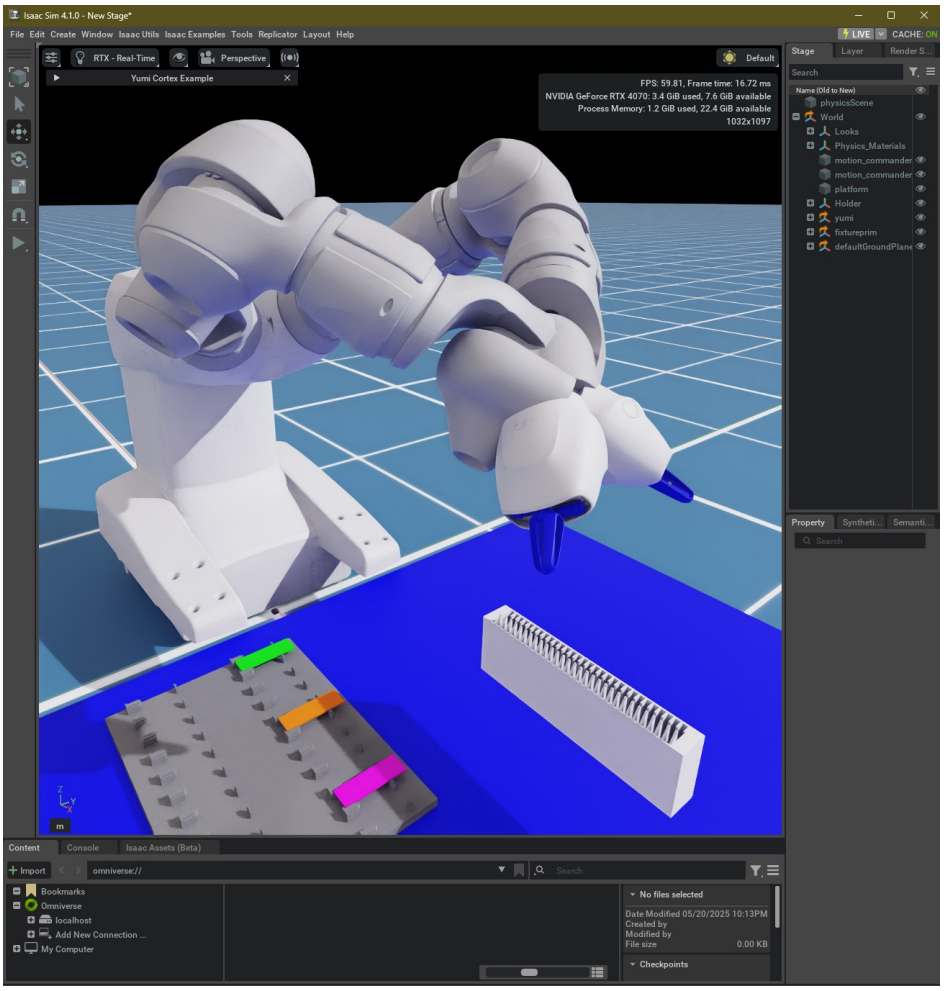



*Figure 3-4: Snapshot of the robot and objects loaded in Nvidia Isaac Sim™*

The *ROS*[p] framework plays a pivotal role in controlling robotic movements and task execution. It interfaces seamlessly with Isaac Sim through robot scripts, which translates the generated task plans into low-level robotic actions, such as trajectory planning, object manipulation, and task execution monitoring. ROS's standardized communication channels facilitate effective integration with other system components, ensuring accurate and reliable task performance.

### 3.2.4. Database tools

*PostgreSQL*[q] is the central storage backbone, interfaced via *psycopg2* (the most popular database adapter for the Python) and *pgAdmin4* (an open-source management tool for PostgreSQL). Carefully structured schemas support:

[p] *Robot Operation System https://www.ros.org/*
[q] *PostgreSQL - an open-source database with a well-deserved reputation for speed, reliability, flexibility and support of open standards.*

- *users*: identity embeddings for voice/face
- *camera_vision*: live object detection logs
- *operation_sequence*: planned robotic actions
- *unified_instructions*: multi-modal command logging etc.

Transactions are designed to be ACID-compliant with concurrent upserts and indexed lookups for real-time performance. *FAISS* indexes are also periodically refreshed and stored alongside face/voice embeddings for authentication.

Database queries are structured using parameterized SQL to guard against injection, and caching strategies are applied where applicable to reduce latency during task planning.

### 3.2.5. Scripting and orchestration tools

All subsystems are implemented in *Python 3.10+,* modularized across scripts (e.g., *face_auth.py*, *voice_auth.py, camera_vision_pgSQL_rs.py*, *task_manager.py*) to ensure readability and testability.

Key dev. practices include:

- Version control using Git (via *GitHub*[r]) for collaborative development and rollback.
- Configuration management through *.env* files and modular YAML settings to support deployment flexibility.
- Centralised logging via *logging_config.yaml* to track events, errors, and system behaviour across modules.
- Orchestration entry points like *task_manager.py*, which coordinates LLM inference, user input capture, robot planning, and feedback delivery.

Each module runs independently but interacts through session states or via the PostgreSQL database, enabling traceability and modular debugging.

Parallelism is handled through Python threading, especially across critical modules like *voice_processor.py*, *gesture_processor.py*, and *session_manager.py*, allowing asynchronous handling of wake-word monitoring, audio transcription, gesture detection, and GUI feedback. Wake-word triggers, input handling, and robotic task activation operate in parallel threads, contributing to a responsive, event-driven interaction model.

### 3.2.6. User interface tools

The graphical user interface (GUI) is developed using *FastAPI,* providing a lightweight, interactive web server that supports real-time voice and text input, status monitoring via integrated tables, and user session feedback. *HTML* and *CSS* are used for layout and styling, while *Ngrok*[s] and *uvicorn* library is employed during development to expose the local server application to the internet for remote testing and interaction. The GUI allows users to initiate sessions, view task progress, and interact with the system without needing direct access to the codebase.

[r] *GitHub link to this project for public access: https://github.com/oscik559/mini_project_repo*
[s] *Ngrok - a tool that securely exposes local servers to the internet, often used for testing APIs, webhooks, or locally hosted apps during development. https://ngrok.com/*

### 3.2.7. Configuration & hardware capabilities

The system was developed and tested on a high-performance workstation running *Windows 11 Education (24H2)*, equipped with an *Intel Core i5-14600KF processor (3.5 GHz)*, *64 GB of DDR5 RAM*, and an *NVIDIA GeForce RTX 4070 (12 GB) GPU* with *CUDA 12.1* support. The machine includes a *1.82 TB SSD*, ensuring sufficient storage and fast I/O operations. This configuration enabled real-time object detection, concurrent voice and gesture processing, and simulation in Isaac Sim with minimal latency. The GPU-accelerated environment also supported LLM inference and deep learning workloads, making it suitable for rapid prototyping and performance benchmarking throughout development.

## 3.3. Integration Strategy

The system's different modules were connected through a central PostgreSQL database, which served as the main communication hub. It kept track of the system state, handled user authentication, and recorded task execution. Each module operated independently but shared updates through database triggers to stay in sync. Python threading was employed to handle concurrency, enabling multimodal inputs such as voice, gestures, and GUI actions.

Tools across modules were chosen based on ease of use, compatibility, performance, and community support. This setup allowed each part to work on its own or together as a complete system, while the overall design enable flexibility and scalability of the subsystems.

# 4. System Architecture & Design

This chapter outlines the design and structure of the proposed system. It introduces the core modules i.e. authentication, perception, language understanding, planning, simulation, and interface, and explains how they interact within the modular, LangGraph-driven architecture. The chapter also presents the project structure and describes how data flows between components via a shared PostgreSQL database to support real-time.

## 4.1. High-level Design Overview

The proposed system operates as a closed-loop pipeline (see Figure 4-1): beginning with user authentication and scene perception, followed by multimodal input capture, language-based task planning, and simulation-validated robotic execution. Each module functions independently but communicates through a shared PostgreSQL database, ensuring decoupled operation and data consistency.

The interaction flow proceeds as follows:

1. *User authentication*: Facial and voice biometrics are processed to retrieve the user profile, task preferences, and prior interaction history.
2. *Input processing*: Voice, gesture, or GUI input is captured and synchronised to form a unified command context.
3. *Language understanding*: The command is parsed by an open-source LLM (e.g., Mistral) and routed via LangGraph to an appropriate task planner or query handler.
4. *Task planning*: Commands are decomposed into executable plans referencing live scene data and stored in the operation_sequence table.
5. *Simulation and execution*: The plan is validated in Isaac Sim using a digital twin of the ABB YuMi robot before synchronisation with the physical cobot.
6. *Feedback and logging*: Outcomes are relayed through the GUI and TTS, while logs and updates are stored centrally for auditing and debugging.

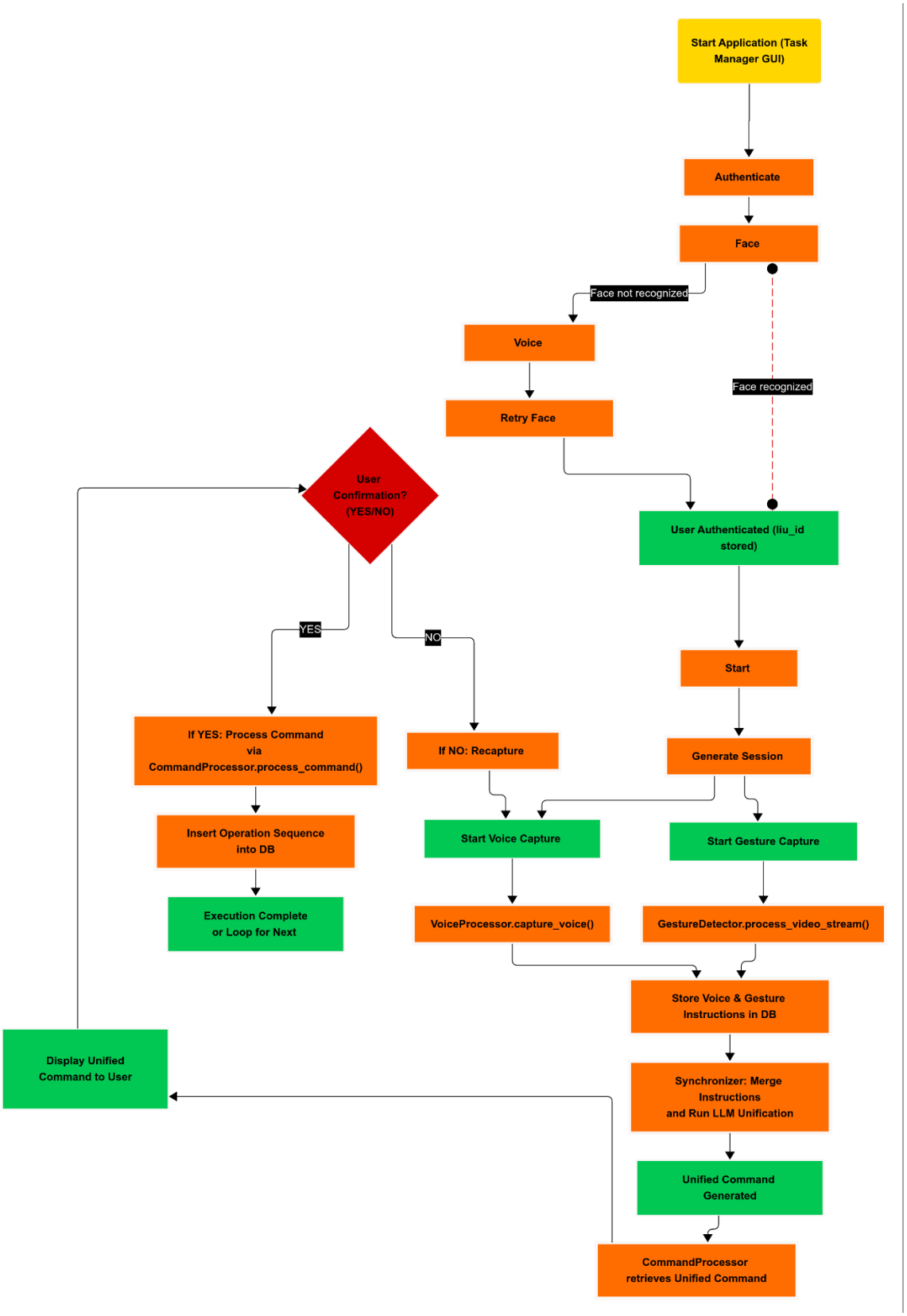



*Figure 4-1: Overview of operational workflow – Design Approach I*[t]

This initial system design prioritises modularity, concurrency, and sim-to-real consistency. Each subsystem is implemented as a Python module with clearly defined inputs and outputs and orchestrated through *LangGraph* nodes based on user intent and system state.

[t] *The other explored Approach II can be found in the appendix section of this report*

## 4.2. Project Structure

This project is structured as a modular Python package under the root directory *mini_project_repo/* (see Figure 4-2). This structure follows a layered functional decomposition aligned with the system's architecture. Each major module (e.g., authentication, perception, language processing, task control, simulation) is captured in its own subdirectory with related scripts, configuration files, and assets. The project *GitHub* repository hosts the project documentation and scripts for public access.

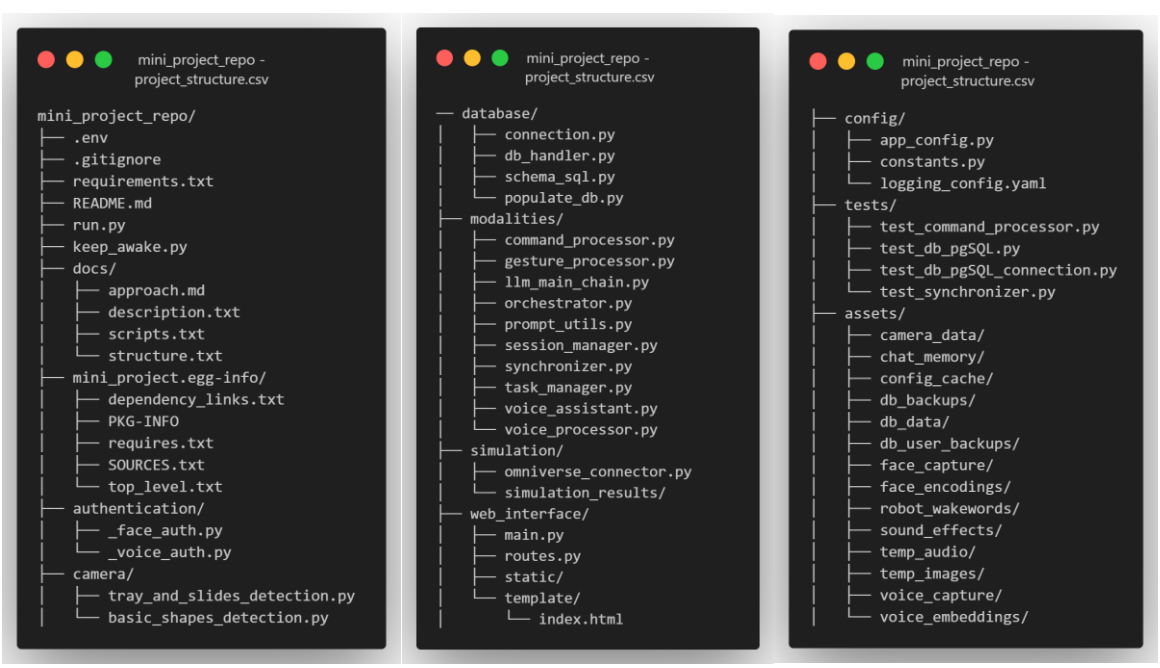



*Figure 4-2: Simplified tree structure showing core modules of the project*

This structure supports readability, reusability, and testability, allowing independent development, unit testing, and integration of each system component. below is a breakdown of the major folders and their core responsibilities:

*Table 4-1: Project folders and responsibilities*

| Folder / Module | Responsibilities |
|---|---|
| *authentication/* | Face & voice recognition scripts for user verification and access control. |
| *camera/* | Real-time object detection and shape analysis using OpenCV and depth sensing. |
| *database/* | PostgreSQL schema definitions, connection handling, and data operations. |
| *modalities/* | Voice & gesture processing, LangGraph orchestration, session management. |
| *simulation/* | Isaac Sim interface and digital twin integration for task execution validation |
| *web_interface/* | FastAPI backend and HTML frontend for Web GUI interaction and feedback. |
| *config/* | Centralised configuration for constants, environment variables, and logging. |
| *assets/* | User-generated inputs: audio, video, embeddings, temp files, backups. |
| *tests/* | Unit & integration tests across database, command, and orchestration modules. |

Supporting scripts such as run.py, keep_awake.py, and .env files reside in the root directory and serve as execution entry points or environment initialisation utilities.

## 4.3. Design of Key Modules

The system integrates six (6) key modules:

### 4.3.1. Authentication Module

This is responsible for verifying the identity of users through facial and voice recognition. This module ensures secure, role-based access to the system by retrieving personalised preferences and usage history from the database before permitting task execution.

The authentication system supports two biometric modalities:

**a) Facial recognition**

Facial recognition is implemented in *_face_auth.py* using the *face_recognition* library, which extracts 128-dimensional facial embeddings from webcam images. These embeddings are matched against a vector store of pre-encoded faces using *FAISS* (Facebook AI Similarity Search) for efficient similarity search. If the distance threshold between the captured and stored embeddings is below a configured value, the user is authenticated. The match result, including timestamp and image, is logged to the users table in *PostgreSQL*.

- Script: *_face_auth.py*
- Libraries: *face_recognition*, *cv2*, *faiss*, *psycopg2*
- Database table: *users*

**b) Voice recognition**

Voice authentication is handled by *_voice_auth.py*. Voice signals are captured through the system microphone and converted into embeddings using *Resemblyzer*. These embeddings are likewise matched against pre-enrolled user profiles via *FAISS*. The combination of facial and voice verification enables robust multi-factor authentication when required. All access attempts are stored for auditing.

- Script: *_voice_auth.py*
- Libraries: *Resemblyzer*, *faiss*, *sounddevice*, *psycopg2*
- Database tables: *users, access_logs*

**c) Integration and execution flow**

At runtime, the system first attempts face recognition via webcam. If no match is found or if the user has no stored face profile, voice authentication is used as a fallback. Upon successful authentication, the user session is initialised, and their profile is loaded for context-aware task planning. Authentication results are used to:

- Personalise language model behaviour (*admin/operator/guest*)
- Link instructions to specific users
- Record activity in the *access_logs* table

```
Authentication Module Summary

1. Capture face or voice input
2. Generate embedding (face_recognition or Resemblyzer)
3. Compare against stored embeddings using FAISS
4. Authenticate user and fetch their profile
5. Log authentication attempt to database
```

*Figure 4-3: Code summary: User authentication via face and voice.*

This module forms the entry point to the broader system pipeline, ensuring that only recognised users may issue task commands or receive system feedback.

### 4.3.2. Vision & Perception Module

This module enables the system to detect objects, compute their positions and orientations, and maintain a digital twin of the live world model that supports task planning and robot execution.

The module is implemented using *OpenCV* for 2D image processing and the *Intel RealSense D453i camera* for RGB and depth data. The output is continuously logged to the camera_vision table in the PostgreSQL database.

Vision scripts for two use pick-and-place applications were developed for this purpose:

#### a) Basic shape & colour detection (for shape stacking case)

The script *basic_shapes_detection.py* handles general-purpose object detection within a predefined region of interest (ROI) on the table. It uses OpenCV's contour and hierarchy analysis to identify standard geometric shapes such as circles, rectangles, and triangles. Detected objects are filtered by colour using HSV masks, and each object's properties—including shape type, dominant colour, centroid coordinates, and area—are calculated and stored.

- Script: *basic_shapes_detection.py*
- Techniques: HSV segmentation, contour detection, shape classification
- Output: Object label, (x, y), colour, area
- Database table: *camera_vision*

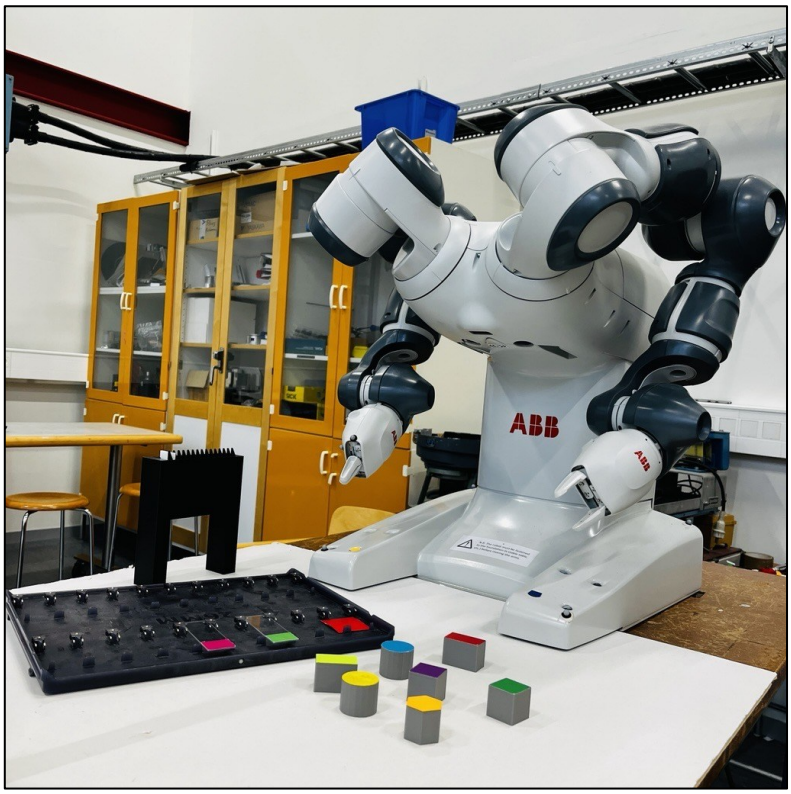



*Figure 4-4: Objects placed on the workspace in front of the robot*

### b) Tray and slide detection (for slide sorting case)

The file *tray_and_slides_detection.py* focuses on identifying trays, slides, and holders as discrete workspace objects. It performs colour-based segmentation and bounding box estimation to extract each objects attribute. Each object's angle and orientation are computed based on relative positioning to static references or bounding rectangle geometry. Multiple object classes are handled concurrently in real time.

- Script: *tray_and_slides_detection.py*
- Output: Class label (e.g., tray, holder, slide), bounding box, angle, position etc
- Integration: Detection objects are logged in the *camera_vision* in real time (see Table 4-2)

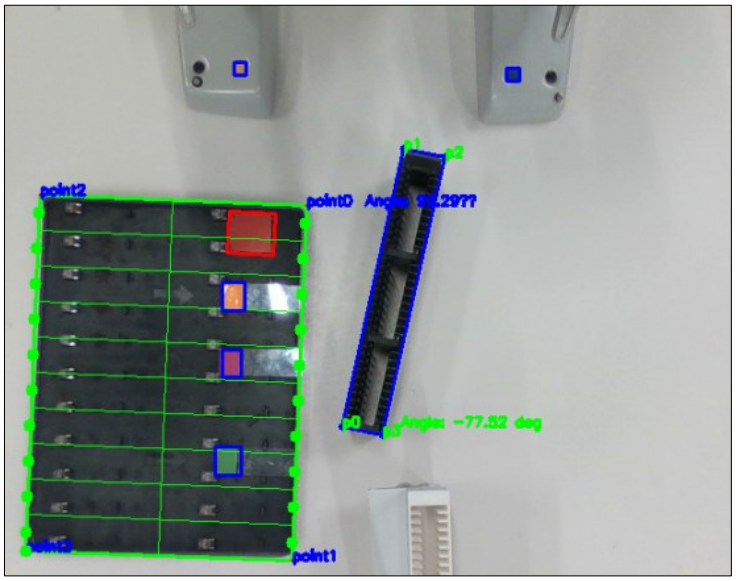



*Figure 4-5: Boundary boxes showing trays and microscope slides detected*

*Table 4-2: Sample table logging objects detected from scene*

| object_id | object_name | object_color | color_code | pos_x | pos_y | pos_z | rot_x | rot_y | rot_z |
|---|---|---|---|---|---|---|---|---|---|
| 24891 | Fixture | black | {0,0,0} | 59.68560393 | 123.6344653 | 0 | 0 | 0 | 92.29061004 |
| 24892 | Holder | black | {0,0,0} | 91.66003461 | 322.9417499 | 0 | 0 | 0 | -77.52283336 |
| 24893 | Slide_1 | green | {0.28,0.49,0.32} | 69.68560393 | 128.6344653 | 0 | 0 | 0 | 92.29061004 |
| 24894 | Slide_2 | orange | {0.81,0.61,0.4} | 79.68560393 | 133.6344653 | 0 | 0 | 0 | 92.29061004 |
| 24895 | Slide_3 | pink | {0.61,0.27,0.42} | 89.68560393 | 138.6344653 | 0 | 0 | 0 | 92.29061004 |

### c) Coordinate calibration via 'screw' reference

The system leverages known positions of two reference screws at the foot of the test robot (the origin on the USD model in Isaac Sim)—an orange circle and a blue square—placed on the workspace to calculate a coordinate transformation from pixel space to real-world coordinates. This calibration logic is embedded in both detection scripts. The reference screws are detected first, and all subsequent object positions are computed relative to them using Euclidean geometry and scaling ratios. This technique ensures:

- Consistency in object localisation even with slight camera displacements
- Accurate pose estimation for robotic pick-and-place planning

### d) Depth data and 3D positioning

While the *RealSense D453i* camera provides RGB-D frames, the current system uses depth data selectively. When enabled, depth channels are accessed to compute z-coordinates for 3D object positioning. This depth data is fused with 2D image coordinates for scene reconstruction in simulation.

```
Vision & Perception Module Summary

1. Stream video input from RealSense camera
2. Apply colour filtering and contour detection (OpenCV)
3. Identify shape, colour, and compute centroid
4. Use reference screws for coordinate calibration
5. Log detected objects to the camera_vision table
```

*Figure 4-6: Code summary: Object detection in perception module.*

> The vision module serves as the visual backbone of the system, allowing real-time scene interpretation for both simulation and physical execution.

## 4.3.3. Multimodal Input and Language Understanding

This module enables the system to interpret user instructions from multiple input channels—namely voice, gesture, and GUI—and convert them into structured task plans using large language models (LLMs). It acts as the cognitive layer of the system, handling input fusion, semantic interpretation, and intent classification through a LangGraph-based orchestration pipeline.

### a) Voice input and transcription

Voice input is captured using the *sounddevice* library and transcribed using *fasterWhisper*, a real-time variant of OpenAI's *Whisper* model. Beam search decoding with language fallback ensures robustness to environmental noise. Wake-word detection is powered by *Porcupine* or *OpenWakeWord*, allowing passive listening and activation on specific trigger phrases (e.g., "Hey Yumi").

- Script: *voice_processor.py*
- Wake-word Model: *Porcupine* / *OpenWakeWord*
- Transcriber: *fasterWhisper*
- Output: Raw command text stored in *voice_instructions* table

### b) Gesture recognition

Hand gestures are captured using a webcam and interpreted using *MediaPipe*. The *gesture_processor.py* script identifies gestures (e.g., thumbs-up, pointing, stop) and maps them to predefined symbolic actions or confirmations. These gestures can augment or disambiguate voice commands when both are captured in the same interaction cycle.

- Script: *gesture_processor.py*
- Tool: *MediaPipe Hands*
- Output: Symbolic gesture label stored in *gesture_instructions* table

### c) Command fusion and interpretation

Captured inputs (voice, gesture, GUI) are combined into a unified representation via the *session_manager.py*. This unified input is forwarded to an LLM via LangGraph, which parses the instruction, determines the user's intent (e.g., task request, scene query, or general question), and routes it to the appropriate subgraph for further reasoning.

- Intent classification node in LangGraph selects one of:
  - *ActionTask* → generates structured task plan
  - *SceneQuery* → responds with contextual information
  - *GeneralQuery* → handles small talk, help, or generic dialogue
- Script: *llm_main_chain.py* / *orchestrator.py*
- Models used: *mistral:latest*, *llama3.3:latest*, *phi4:latest* (via *Ollama*)
- Prompt templates: Constructed dynamically using *prompt_utils.py*

### d) LangGraph orchestration pipeline

LangGraph provides a directed workflow where each node represents a cognitive subroutine. It maintains state, logs transitions, and enables session-specific memory. For example, a voice command like "Stack the red slides after the blue ones" would pass through:

1. *Input node* (transcribed voice)
2. *Classifier node* (identifies task intent)
3. *ActionPlanner* node (calls LLM + scene data)
4. Final *response* node (plans task and updates *operation_sequence*)

LangGraph's node-modular design enables extensibility and fault tolerance, allowing fallback logic or clarification prompts when input is ambiguous.

```
Multimodal Input Module Summary

1. Listen for wake-word using Porcupine/OpenWakeWord
2. Record audio and transcribe using fasterWhisper
3. Optionally capture gesture using MediaPipe
4. Store voice and gesture data to database
5. Merge into unified instruction for LLM processing
```

*Figure 4-7: Code summary: Multimodal input capture and command fusion.*

> This module bridges raw user input and robot-executable task logic through a combination of real-time sensing, symbolic mapping, and natural language reasoning. By fusing multiple input modalities and grounding them via LLMs, the system achieves adaptive, context-aware task understanding.

### 4.3.4. Task Planning & Orchestration

This module handles the transformation of user instructions into executable robot tasks, coordinating the flow from semantic intent to physical or simulated action. At its core is a rule- and graph-based orchestration system powered by LangGraph, which integrates planning logic, database interaction, and submodule coordination.

#### a) Command processing

The command_processor.py script parses structured user commands (generated from the LLM layer) and maps them to task-specific routines. This mapping uses both LLM output and domain-specific templates defined in the *operation_library* database table. The output is a set of discrete steps (e.g., pick, place, sort) stored in operation_sequence.

- Script: *command_processor.py*
- Inputs: JSON-formatted commands, object metadata from *camera_vision*
- Outputs: Ordered task plan written to *operation_sequence*

#### b) Task manager and orchestration

*task_manager.py* acts as the runtime entry point for full pipeline execution. It supervises wake-word listening, input capture, LangGraph triggering, and feedback coordination. Once the operation_sequence is written, this module initiates execution either in simulation or on the real robot.

- Script: *task_manager.py*
- Responsibilities:
  - Input monitoring
  - Pipeline orchestration
  - Progress feedback and error recovery

#### c) LangGraph integration and control flow

LangGraph acts as a high-level semantic controller:

- Nodes: Authentication → Command Input → Classifier → Planner → Executor → Feedback

- Each node in the LangGraph pipeline holds a scoped memory and state dictionary
- Planning nodes use prompt-engineered LLM queries to determine actions, fallback logic, or queries to the camera_vision table

This design allows:

- Runtime branching (e.g., fallback if no objects found)
- State-based transitions (e.g., if a task was recently completed, skip reinitialization)
- Node isolation (each module can be tested independently)

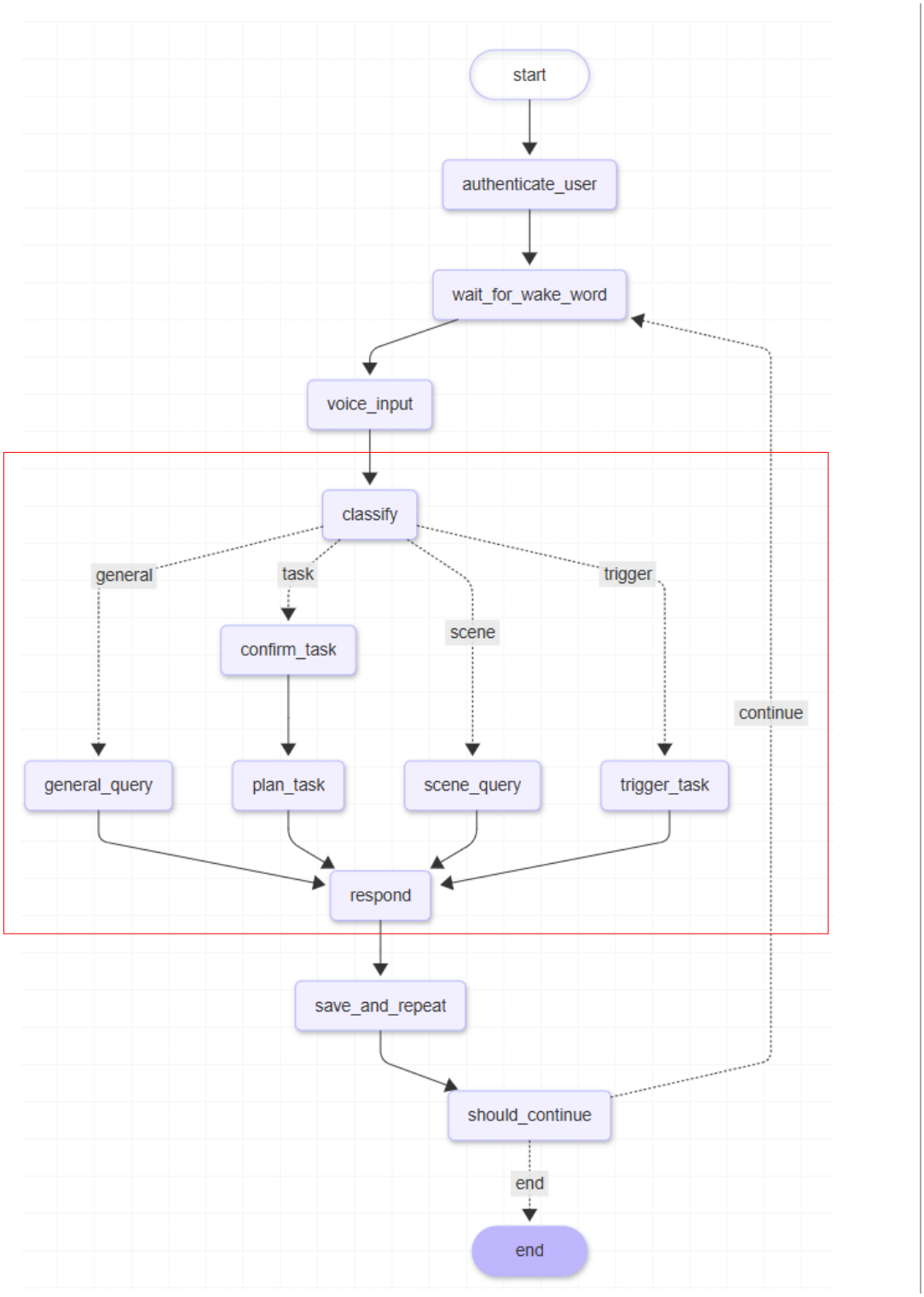



*Figure 4-8: LangGraph routing of user commands via nodes and subgraphs*

### d) Database-driven contextual planning

During planning, the system cross-references live scene data (*camera_vision*), operation definitions (*operation_library*), and user profiles (*users*) to resolve object references, colour dependencies, and user-specific preferences (e.g., object stacking order).

*Table 4-3: Sample task plan for executing a command*

| id | operation_id | sequence_id | sequence_name | object_name | command_id | processed | execution_time |
|---|---|---|---|---|---|---|---|
| 245 | 1 | 1 | pick | square_1 | 46 | FALSE | 14:40.8 |
| 246 | 2 | 2 | travel | square_1 | 46 | FALSE | 14:40.8 |
| 247 | 3 | 3 | drop | square_1 | 46 | FALSE | 14:40.8 |
| 248 | 4 | 1 | pick | circle_2 | 46 | FALSE | 14:40.8 |
| 249 | 5 | 2 | travel | circle_2 | 46 | FALSE | 14:40.8 |
| 250 | 6 | 3 | drop | circle_2 | 46 | FALSE | 14:40.8 |
| 251 | 7 | 1 | pick | circle_1 | 46 | FALSE | 14:40.8 |
| 252 | 8 | 2 | travel | circle_1 | 46 | FALSE | 14:40.8 |
| 253 | 9 | 3 | drop | circle_1 | 46 | FALSE | 14:40.8 |
| 254 | 10 | 6 | go_home | | 46 | FALSE | 14:40.8 |

This enables the system to plan context-aware actions like:

- "*Stack the shapes in the order of the square, then the circles*" as shown in the table Table 4-3: Sample task plan for executing a commandabove.
- "*Sort the slides according to the order; green, pink and orange*"

```
Task Planning & Orchestration Summary

1. Read command from unified_instructions
2. Classify user instruction using LangGraph
3. Route to correct subgraph: ActionTask, SceneQuery, or GeneralQuery
4. Query scene data and user profile from the database
5. Match intent to operation template from operation_library
6. Generate structured task plan using LLM output
7. Store plan in operation_sequence table
8. Trigger execution via task_manager (real or simulated)
```

*Figure 4-9: Code summary: Graph-based planning & execution pipeline.*

> This module forms the system's core decision-making logic. By combining LangGraph-based orchestration with contextual lookups and structured planning routines, it transforms natural instructions into verifiable robot actions.

## 4.3.5. Simulation Module

This validates robotic task plans in a photorealistic virtual environment before they are deployed to the physical ABB YuMi cobot. This is achieved in *NVIDIA Isaac Sim* integrated with the system through a ROS2-compatible digital twin interface.

### a) Digital Twin Integration

The robot's digital twin replicates the real-world configuration of the ABB YuMi manipulator, including its kinematics, workspace limits, and gripper geometry. The system synchronises task

parameters such as object coordinates, pose goals, and sequence steps from the operation_sequence and camera_vision tables.

- Script: *omniverse_connector.py*
- Interface: *ROS2* bridge between Isaac Sim and system database
- Input: Object positions and action steps (task plan)
- Output: Executed movement sequences in Isaac Sim

### b) Simulation Execution Flow

The planned task is visualised and verified inside the Isaac Sim environment before physical execution. The simulation environment includes:

- Imported workspace mesh or USD layout (including trays, slides, and screw references)
- Realistic lighting and physics to validate reachability and collision safety
- Joint trajectory visualisation and success/failure feedback

The same movement sequences used in simulation are mirrored on the physical ABB YuMi if the validation passes. This enables rapid, safe iteration and debugging.

### c) ROS2 Messaging and Command Dispatch

The system uses ROS2 topics to publish and subscribe to robot commands and feedback. These include:

- *task_start* → to initiate an execution plan
- *execution_feedback* → for task status updates
- *robot_status* → current pose or completion signal

The *omniverse_connector.py* *(this is still under development)* would listen for new entries in *operation_sequence*, send them to Isaac Sim, and log the response. This will tightly couple the planner and executor within the LangGraph workflow.

```
Simulation Module Summary

1. Monitor operation_sequence for new plans
2. Send task steps to Isaac Sim via ROS bridge
3. Simulate robot movement using digital twin
4. Log simulation results to simulation_results
5. Trigger real-world execution if simulation succeeds
```

*Figure 4-10: Code summary: Simulation via Isaac Sim and ROS.*

> The simulation module ensures that all planned robotic operations are physically feasible and collision-free before execution. It serves both as a sandbox for debugging and as a digital twin interface for continuous sim-to-real transfer learning.

### 4.3.6. Web Interface and GUI

The GUI provides access to core system functions including voice input, session status, database monitoring, and feedback display. The interface is implemented using *FastAPI* for the backend, with HTML/CSS templates and JavaScript components for frontend rendering.

#### a) System entry and session control

The GUI is served via the main.py script under the *web_interface/* directory. Upon loading, users are presented with options to:

- Start or resume a session
- Initiate voice commands
- View system logs or detection results
- Observe task progress and robot status

The backend is connected to session management logic in *session_manager.py* and *routes* interactions through the orchestrator depending on session context and user role.

- Backend script: *main.py*
- Frontend template: *template/index.html*
- Routing: *routes.py*

#### b) Voice input and Wake-word monitoring

Voice input is integrated into the GUI via microphone streaming, enabling hands-free interaction. Wake-word detection is enabled using *OpenWakeWord* or *Porcupine*, which run passively in the background to detect trigger phrases (e.g., "*Hey Yumi*"). Upon detection, the system:

1. Activates audio capture
2. Transcribes the command via fasterWhisper
3. Sends it through the LangGraph pipeline for classification and response

#### c) Task feedback and status monitoring

Real-time feedback from the planner and simulation is displayed in the GUI. Users receive:

- Task execution status (e.g., "Executing step 2 of 5")
- Errors or fallback prompts
- Visual confirmation of detected objects (if applicable)

The GUI also reflects authentication status, active user's name, and historical task logs in a table.

#### d) Ngrok Integration for remote access

During testing and demos, *Ngrok* is used to expose the local *FastAPI* server over a public HTTPS tunnel. This enables external devices to access the GUI without hosting on a separate web server. It also supports GUI control over LAN or WAN with minimal setup.

```
Web Interface & GUI Summary

1. Launch FastAPI server for GUI
2. Load HTML/CSS template with session controls
3. Display authentication, status, and feedback
4. Stream voice input and pass to pipeline
5. Use Ngrok to expose local interface if remote
```

*Figure 4-11: Code summary: FastAPI web interface and session control.*

The GUI component ensures that the system is accessible, transparent, and interactive. It supports real-time monitoring, hands-free command entry, and visual feedback, making the platform suitable for non-technical users and adaptive to different contexts of use.

## 4.4. Data Flow and Storage Logic

This section details how data is collected, processed, and exchanged across system modules, focusing on the role of the PostgreSQL database and its schema in coordinating multimodal input, perception data, authentication states, and task execution sequences. The system relies on structured data pipelines to ensure synchronisation between perception, planning, and action layers.

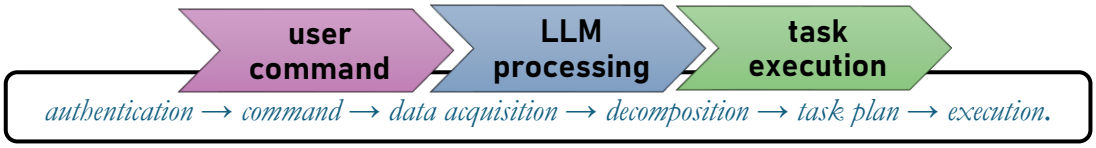



*Figure 4-12: Overview of data flow through modules*

### a) Input streams

Data enters the system through three primary channels:

1. User authentication: Biometric inputs (face images and voice recordings) are captured during session initiation. These are processed into 128D face encodings and *Resemblyzer* voice embeddings, respectively, and stored in the *user* table. All access attempts are logged in *access_logs*.
2. Perception data: The camera continuously updates the *camera_vision* table with scene-level object data. Detected objects include geometric shapes, trays, and screws, with associated attributes like position (x, y, z), orientation (yaw), colour code, and timestamps.
3. Instruction capturing: Voice and gesture inputs are captured in parallel. Transcribed voice commands (via *fasterWhisper*) are stored in *voice_instructions*, and gesture classifications (via *MediaPipe Hands*) are logged in *gesture_instructions.* These are later fused into a unified command stored in *unified_instructions*.

### b) Database-centric integration

A PostgreSQL database acts as the system's central data hub, supporting:

- User profiling: The *user* table holds voice/face embeddings and personal preferences (e.g., frequent tasks, object priority).
- Task tracking: The *unified_instructions* table logs parsed instructions, while *operation_sequence* stores the final LLM-generated execution plan.
- Skill Library: The *sequence_library* table defines robot action primitives (e.g., pick, place, rotate), referenced during task planning.
- Scene mapping: *camera_vision* dynamically maps real-world objects to database IDs for task grounding.

### c) Outbound and inter-module data flow

Once a unified instruction is processed by the LLM, the resulting structured task plan is inserted into *operation_sequence*. The simulation node reads this table and initiates simulation via Isaac Sim or real-world execution via ROS. Execution results (success/failure) and performance logs are saved to *simulation_results*, closing the loop.

### d) Data pre-processing and validation

To ensure data quality:

- Face and voice samples undergo denoising and amplitude thresholding.
- Object detection data is filtered for size, location bounds, and colour validity.
- Instruction inputs are timestamped and validated against scene context before LLM invocation.
- Periodic purging of obsolete entries (e.g., outdated object data) prevents database clutter.

### e) Database schema overview

Some major operational tables include the below and the interactions are shown in Figure 4-13 and Figure 4-14. The *schema.py* scripts outline all.

| Table | Purpose |
|---|---|
| *users* | Stores user ID, face encodings, voice embeddings, preferences |
| *camera_vision* | Logs detected object attributes (shape, colour, position, etc.) |
| *voice_instructions* | Stores raw transcribed voice input |
| *gesture_instructions* | Logs symbolic gestures with confidence scores |
| *unified_instructions* | Holds fused multi-modal instructions for task generation |
| *sequence_library* | Defines reusable robot actions (e.g., pick, travel, drop) |
| *operation_sequence* | Stores final task plans generated by the LLM and mapped to robot actions |
| *access_logs* | Tracks all authentication events (method, timestamp, outcome) |
| *interaction_memory* | Logs interaction history for contextual personalization |
| *simulation_results* | Captures feedback from Isaac Sim and real-world execution |

This structured and transactional data model enables:

- Modular communication between perception, planning, and control layers
- Persistent user modelling and preference recall
- Transparent, real-time monitoring of all system activities

By decoupling execution logic from physical inputs and storing all context centrally, the system ensures extensibility, auditability, and task reproducibility across sessions.


*Figure 4-13: operation_sequence schema and references*

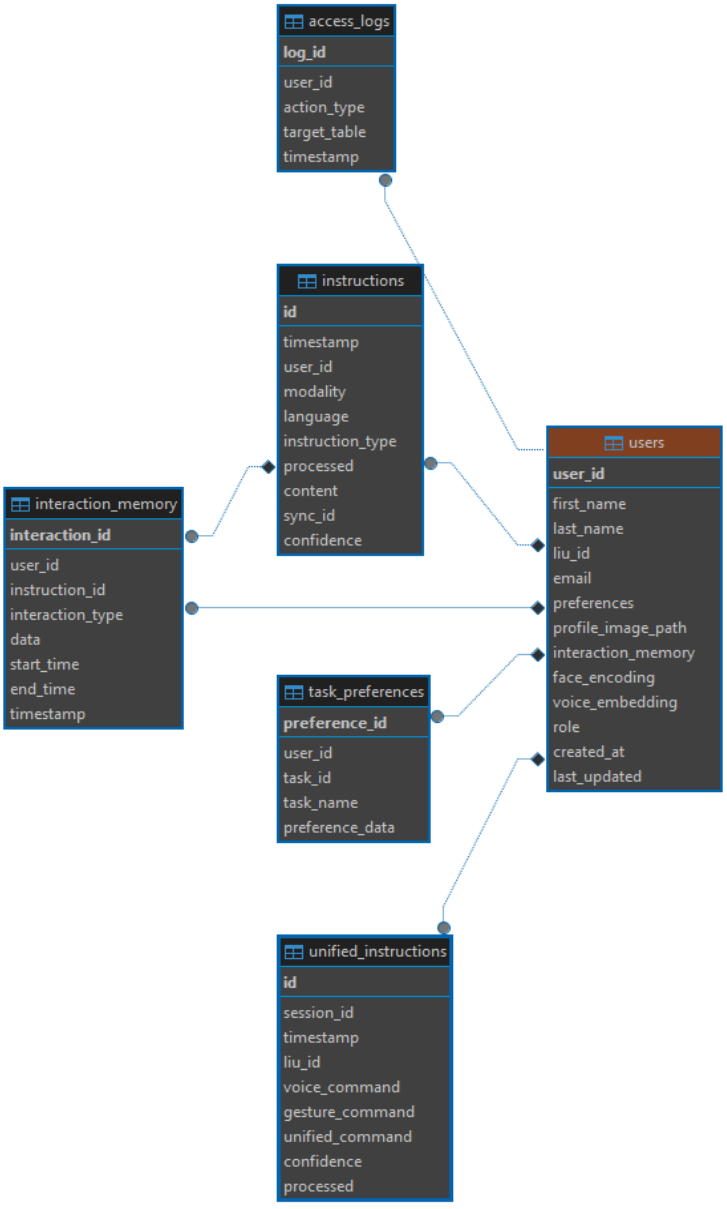



*Figure 4-14: Users table schema and connections*

## 4.5. Notable Implementation Features

One of the core features of the system is its database-driven personalization. User preferences and historical interactions, such as specific references made by users to objects are stored in tables like users or unified_instructions. This enables the system to accurately interpret vague references, improving the precision of the language model's (LLM) responses. Additionally, updates from face or voice recognition, along with multiple encodings per user, are continuously fed into *voice_instructions*, *gesture_instructions*, and *interaction_memory*. This continuous stream of data allows the system to personalize interactions and enables historical analytics, thereby refining the user experience over time.

The system uses centralized configuration files such as *.env* and *app_config.py* to store key constants (e.g., *LLM_MODEL*, *DB_URL*). This approach ensures that any updates to parameters or changes to the environment are handled consistently, preventing disruptions to the core system logic. Centralizing configuration in this manner simplifies maintenance and adaptation of the system to different operations.

Error handling and adaptive re-planning are also integral to the system's functionality. If a recognized object is no longer present in the scene (whether due to removal or being out of view) the system detects this when it queries the camera_vision table. If the LLM-generated task plan includes an invalid reference, the orchestrator invokes a re-request for additional user input.

A user-centric approach is embedded into every stage of the pipeline. By integrating biometric authentication with historical context, the system tailors each session to individual users. This personalization reduces redundancy, increases command accuracy, and contributes to a more intuitive and satisfying user experience, effectively enabling the system to "remember" and adapt to user-specific preferences.

Lastly, feedback loops support ongoing improvement. The system logs both successful executions and errors, enabling debugging of misinterpretations or recurring issues. These logs provide valuable data that can be used for reconfiguration, enabling refinements that enhance the system's accuracy and responsiveness over time.

Together, these features provide the foundation for an adaptable, user-aware robotic system that can continuously improve and provide personalised, reliable interactions.

# 5. Results & Discussion

In this chapter, the results of the integrated system are presented and discussed. The performance of each subsystem has been tested, with a focus on how well they work as part of the larger system. The chapter outlines the system's implementation and integration outcomes, technical capabilities, performance evaluation, and key findings from the tests conducted.

---

## 5.1. Integrated System Implementation

### 5.1.1. Overview

The following key components make up the integrated system developed in this thesis:

1. *Computer vision*: Responsible for perceiving and interpreting the environment, including detecting and localising objects, identifying user faces, recognizing gesture cues, and providing real-time scene data to the robotic system via the database.
2. *Database*: This serves as a central repository for storing critical operational data, incl. object attributes (physical and model-based), skills, task plans, user profiles, interaction history etc. This is continuously queried to enable adaptive, personalised task execution.
3. *LLM & Modalities*: LLMs facilitate the interpretation of natural language and gesture-based inputs, allowing the system to generate context-aware task plans based on user requests, and support intelligent, adaptive human-robot interaction.
4. *Authentication*: This module verifies user identity using facial and/or voice recognition, enabling secure, role-based access tailored to an individual's task preference, and prior interaction history.
5. *Web Interface*: The web-based GUI serves as the primary medium for user interaction, enabling the input of commands and reception of system feedback. It facilitates communication between the user and the robotic system, allowing users to issue tasks, monitor database entries, and track task planning and execution in real time.
6. *Robot Control:* Developed not as part of the thesis scope, but alongside it for the broader project goal. The module interfaces with both the digital twin in Omniverse Isaac Sim and the physical ABB Yumi cobot, retrieving task plans from the database and executing them in simulated and real environments.

Each of these components was developed independently and then integrated into a unified system as shown below. The following sections describe how these modules were brought together to form a fully functional pipeline.

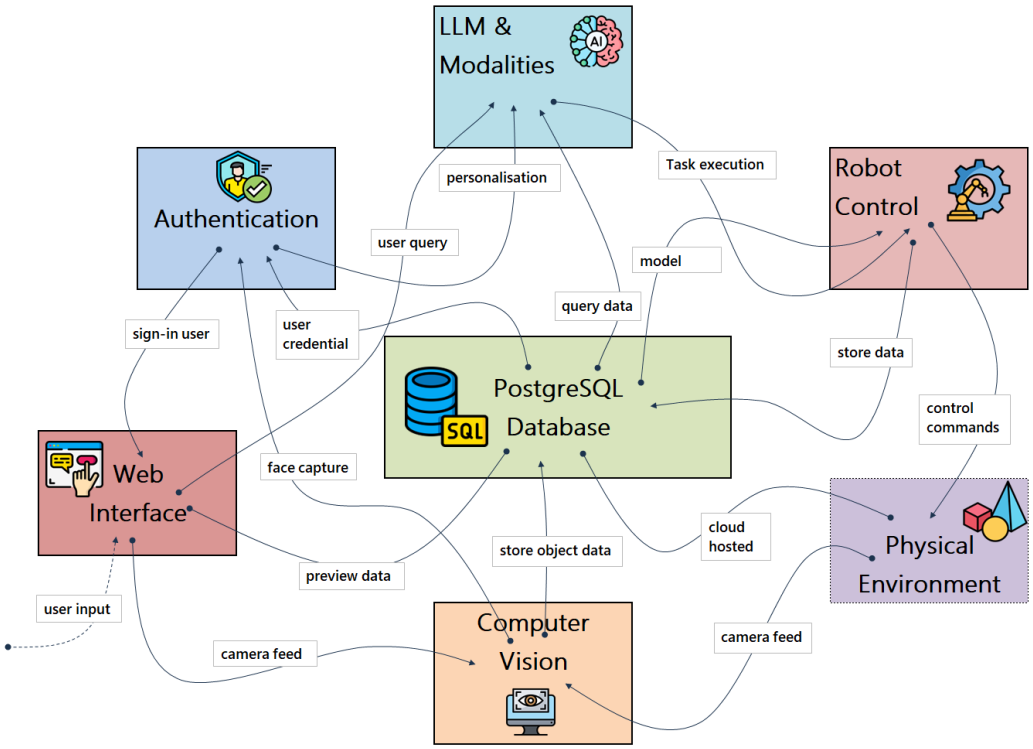



*Figure 5-1: Overview of interactions between modules.*

### 5.1.2. Integration outcomes

The integration overview shown in Figure 5-2 showcase the interaction between the robot, its environment, and the user. The workflow begins with user authentication, where upon initiation, the system ensures the user's identity through face and/or voice recognition. This initial step guarantees that the system recognizes the user and can provide tailored interactions based on their database profile.

Following authentication, the vision module through a camera captures real-time footage of the robot's workspace. The camera feed is processed to detect and classify objects in the scene. Key object data, such as positions, types, and colours, are extracted and continuously stored in the database, ensuring it maintains an up-to-date representation of the scene. This allows for an accurate and dynamic interaction with the physical environment.

When the user issues a command or question, using their natural language, with voice and maybe also gestures, the LLM interprets the request via well-defined system prompts, cross-referencing them with the authenticated user, their roles and previous interactions form the database. In the case of a task/action request, the LLM processes the command, structures the required task, queries the database for relevant task parameters and object properties to generate a task plan, and intelligently formulates a response. Moreso, general (non-action) requests/questions are handled by defined system prompts which ensures that interaction flows naturally, maintaining context.

Once the task plan is defined, the robot control module, initiates the task execution control script, queries the database for the task plan and object data, and proceeds to execute it in the virtual environment (Omniverse Isaac Sim), whilst the physical robot mirrors the actions in real time. The user request is accomplished using the real-time data retrieved from the database tables, either within a virtual simulation or in the real world, depending on the setup.

During task execution, the robot's actions can be continuously monitored though the database tables, with feedback provided to the user via the web interface if requested. This allows the user to track the status and progress of the task in real-time. The integration of a database ensures efficient data retrieval and management throughout the operation.

The system adapts based on the user's feedback (e.g. task confirmations) and any changes in the scene (e.g. object positions, orientations), offering further interaction opportunities. As tasks are completed, and new commands are issued, the system dynamically adjusts its responses to queries, utilizing interaction data logged in the database, ensuring that interactions remain contextually relevant.

The modular integration facilitated parallel development and testing of individual components, allowed for flexibility in adding new user data and features, through well-defined database schemas. This will ensure future scalability to meet new task requirements.

Other key integration outcomes include:

1. The use of a centralized PostgreSQL database enabled efficient sharing of system data between modules, supporting real-time updates and concurrent queries.
2. The schema supports adding new task types, user roles, gesture definitions, object USD data, robot control features etc. without major codebase refactoring.
3. The system remained stable during extended testing, with error handling mechanisms at module boundaries to manage communication failures and data access issues.

Despite these successes, several integration challenges had to be addressed:

1. Coordinating task execution across simulation and real-world environments presented major challenges. Ensuring real-time synchronization between the virtual and physical robots required careful management of communication latency and error correction.
2. Rapid scene change, especially during object manipulation or occlusion events, sometimes caused mismatches between the vision output and the database state. This was addressed by implementing transaction management and state validation.
3. Integrating LLMs into the live interaction loop introduced challenges related to processing delay and command ambiguity. Prompt tuning, caching common responses and introducing fallback logic for unclear queries helped improve reliability.
4. The most significant challenge was hardware related. While the control logic was initially developed for a standard single-arm robot, the actual test robot was equipped with two arms, each with an extra joint. This mismatch limited reach and required adjustments to motion planning and execution strategies.

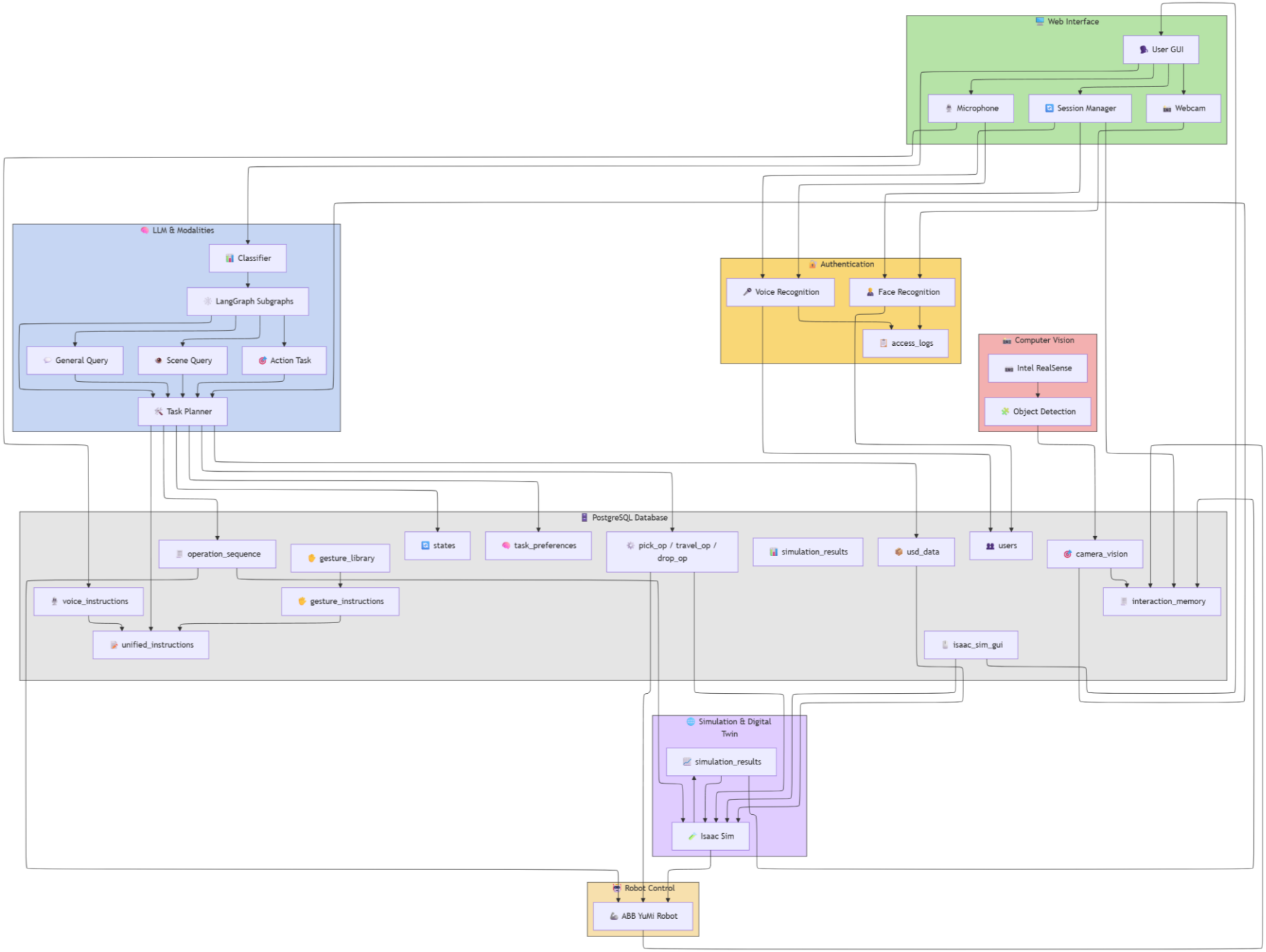



*Figure 5-2: Integration between modules*

## 5.2. Performance Evaluation

Table 5-1 below presents a structured evaluation of the system modules implemented in this *thesis*. It summarizes each components' key capabilities, performance metrics, limitations, recommended improvements, and validation methods, offering an overview of system effectiveness.

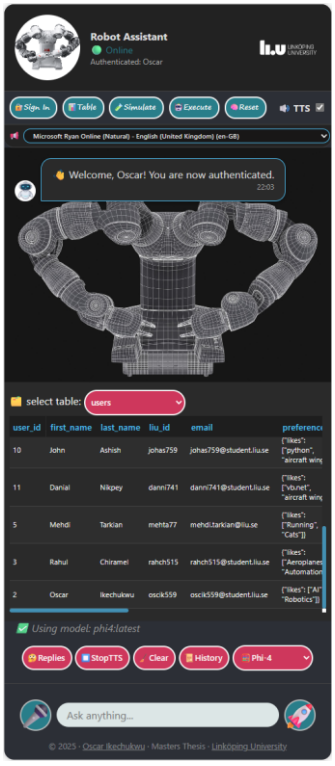
*Table 5-1: Combined performance and capabilities summary*

| Component | Key capabilities | Performance metrics | Limitations / challenges | Recommendations | Validation evidence |
|---|---|---|---|---|---|
| Computer Vision<br><br>(Real-time object detection and pose estimation) | • Real-time detection of trays, slides, and geometric shapes<br>• HSV-based colour segmentation and shape classification<br>• Robot-foot-referenced alignment and angle estimation<br>• Frame-wise logging of object metadata (position, colour, orientation) | • Frame rate: 30 FPS<br>• Object Recognition Accuracy: Reliable under consistent lighting<br>• Positional Accuracy: Sub-centimetre error via relative scaling<br>• Angle Estimation: good enough based on relative edge alignment | • Performance drops under occlusion or poor lighting<br>• False positives in object classification during visual clutter<br>• Temporal inconsistencies when screw anchors are momentarily lost | • Integrate temporal filtering for persistent object tracking<br>• Add adaptive ROI resizing to maintain detection under movement<br>• Introduce timeout logic and screw reinitialization alerts<br>• Kalman filtering for stability | Live RealSense tests with shape/tray detection; overlay verification and coordinate logs |
| Database Integration<br><br>(database-driven scene logging and update) | • Structured PostgreSQL schema supporting multi-table task planning<br>• Real-time insertion and querying of *scene data*, *libraries*, and operation tables<br>• Modular population of operation parameters based on task type | • Database Read/Write latency: <50 ms on average<br>• Continuous data stream into db. tables with upsert operations<br>• Real-time synchronization for task lookup and parameter injection | • Schema evolution requires careful version control<br>• Accumulation of stale detection records if not purged regularly<br>• Dependence on external trigger mechanisms for cleanup | • Schedule regular purging of outdated vision records<br>• Introduce schema introspection for dynamic table compatibility<br>• Log-based rollback strategy for unsafe write operations | PostgreSQL logs and table inspection after upserts; verified update/cleanup queries |
| Task Planning & Execution<br><br>(Mapping parsed commands to robot actions) | • LLM-driven interpretation of natural language and gesture commands<br>• Mapping to predefined operations and database-stored sequences<br>• Parameterized task execution logic shared between simulation | • Success rate: >80% for the few supported tasks (stacking, sorting) – (subjective rating)<br>• LLM Interpretation Accuracy: good enough for first-attempt command parsing | • LLM ambiguity in interpreting context-free or vague commands<br>• User mispronunciation or grammar variants may lead to incorrect mapping<br>• Task chaining across | • Refine prompt templates and incorporate proper user confirmation cycles<br>• Add fallback command interpretations when primary fails<br>• Enhance intent | • Simulated task completion in Omniverse Isaac Sim and command traces |

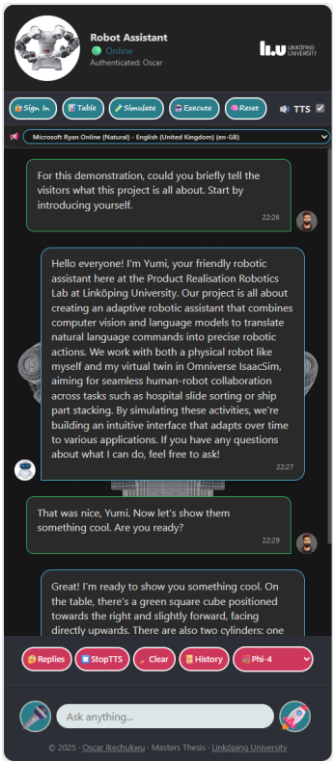
| | | | | | |
|---|---|---|---|---|---|
| | and real-world runs<br>• New task types added through SQL only, without code changes | • Planning Latency: Negligible with pre-loaded SQL templates | multiple objects requires robust state tracking | disambiguation through profile-aware context matching | |
| Digital Twin Synchronization<br><br>(Sim-to-real transfer in Omniverse Isaac Sim) | • Shared task plan across Omniverse Isaac Sim and ABB YuMi<br>• Visual-object mapping consistency using database-sourced coordinates<br>• Behavioural logic (e.g., block stacking) replicated identically in both domains | • Execution Latency: slight communication buffer/delay between sim and real action<br><br>• Fidelity: High positional/motion comparison, minor deviation in grasping precision | • Slight misalignment between simulation and real-world due to calibration drift<br>• ABB YuMi's physical limits occasionally restrict full trajectory reproduction<br>• Manual re-tuning required when switching between simulation and hardware | • Implement automated calibration tools between sim and real<br>• Tune grasp affordances to account for ABB YuMi's joint constraints<br>• Monitor joint limit usage to avoid command overshoot | Pose validation between simulation and physical cobot; screen recordings and snapshots |
| Authentication<br><br>(User authentication and personalization) | • FAISS-based face and voice verification<br>• Automatic registration and embedding matching<br>• Contextual profile linking and fallback modes | • Match success rate: reliable for face<br>• Fast retrieval from embedding index (very fast)<br>• Adaptive fallback to voice when face is absent | • Limited diversity in face dataset may introduce bias<br>• Voice mismatch under noisy environments<br>• Requires real-time camera availability for registration | • Expand training data with diverse face samples<br>• Apply noise suppression and retry prompts for voice<br>• Implement dual-modality fallback policies | Face and voice trials using FAISS search accuracy and similarity scoring |
| Web GUI<br><br>(User-friendly interaction interface) | • FastAPI-based user interface for assistant interaction<br>• Live task tracking, chat display, session summaries<br>• Integrated command trigger and mute mechanism | • GUI Responsiveness: Smooth for <10 concurrent events<br><br>• Full pipeline access through front-end controls | • TTS blocking delays feedback loop<br>• Input spamming may stall updates<br>• Minimal UX affordances for errors or timeouts | • Add async processing for TTS playback<br>• Implement throttling and debounce for inputs<br>• Introduce visual indicators for loading/errors | Real-time interaction test via local browser; session data reflected in PostgreSQL backend |
| Natural Language Understanding (LLM)<br><br>(Command parsing, task planning, personalization) | • Prompt-driven semantic interpretation of commands<br>• Role-personalised context memory integration<br>• Structured subgraph selection based on command type | • Match accuracy: reliable (based on prompt tests)<br>• Role recall and Context awareness: Verified via logs<br>• Prompt response time: ~2s average for complete parse on smallest model (Qwen) | • Over-dependence on prompt formatting<br>• Memory recall error under command chaining<br>• Verbosity or ambiguity in user phrasing affects match | • Use chain-of-thought or scratchpad planning strategies<br>• Optimize prompt token efficiency<br>• Add disambiguation steps for uncertain tasks | • Command parsing tested via scripted user scenarios; logs and LLM responses |

## 5.3. Use Case Demonstration

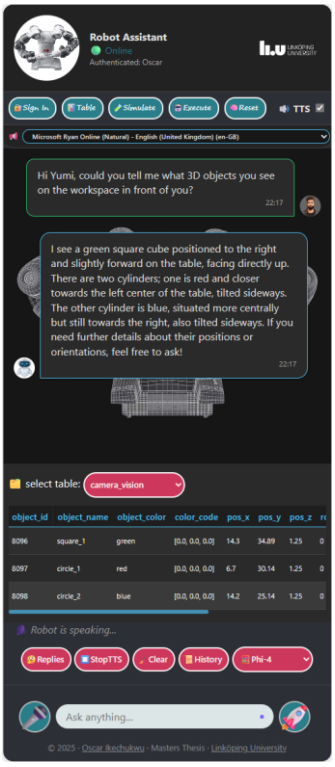
Figure 5-3 shows the real-time interaction between a user and the robotic system via the web GUI.



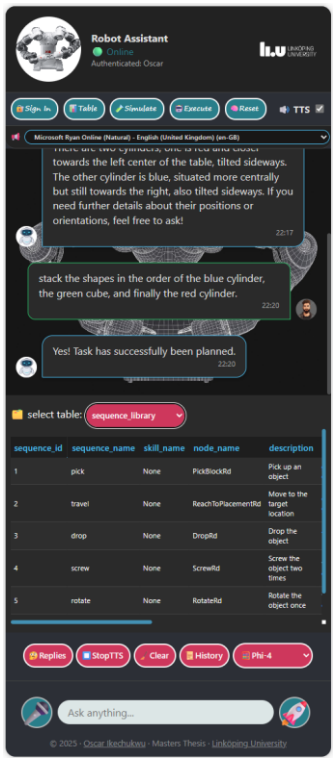

*(a) Authentication screen confirming user identity.*



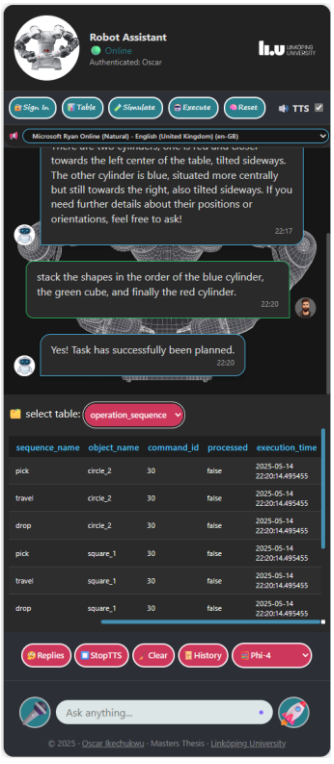

*(b) Contextual assistant response describing project goals*



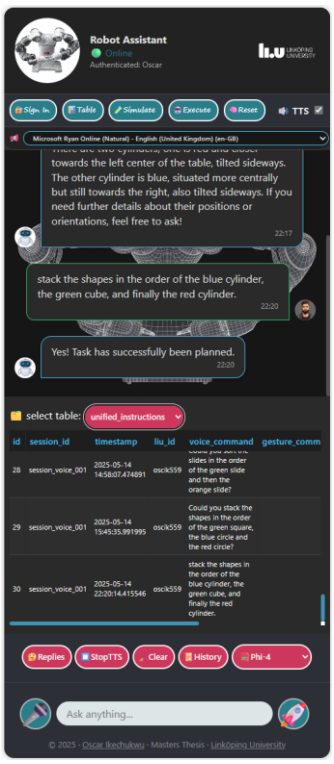

*(c) Scene query response based on db-linked camera_vision table*




*(d) Library of sequence primitives used to structure task execution.*




*(e) Operation steps generated in the operation_sequence table*




*(f) Logged user interaction in the unified_instructions table.*

*Figure 5-3: GUI interaction flow between a user & the robotic assistant.*

In (a), the assistant authenticates the user (*Oscar*) via biometric credentials just before interaction begins, enabling personalised memory and access using the users + role tables. In response to a general user query, the assistant provides a contextual explanation of the system architecture and goals in subfigure (b), leveraging stored prompts and LLM-based memory. In (c), the assistant responds to a scene query about visible objects by referencing structured object data from the *camera_vision* table, highlighting shape, colour, and orientation. The *sequence_library* is shown in (d), containing reusable task primitives (e.g., pick, drop, rotate) used to structure execution.
In (e), the assistant logs a parsed command in the *unified_instructions* table for persistent tracking and multi-modal context chaining across tasks. Finally, (f) depicts how the *operation_sequence* table is populated with executable steps derived from user commands and scene context. Together, these subfigures demonstrate seamless integration of perception, reasoning, and execution through a LangGraph-based orchestration.

## 5.4. Preliminary Findings

Over 50 test sessions were conducted across two use applications: colour-based *slide sorting* and geometric *shape stacking*, using both simulation and real-world environments. Eight users (mostly thesis peers) engaged with the system, issuing voice and gesture-based commands. On average, the system demonstrated an overall task execution success rate of approximately 80%, with command parsing accuracy improving progressively through prompt refinement and model switching.

Computer vision performance was reliable under stable lighting and produced real-time object pose estimates at 30 FPS but degraded under strong shadows and visual clutter. Failures typically occurred during occlusion events or when the screw-based reference anchors were temporarily lost. These conditions exposed the need for enhanced robustness using temporal filtering or YOLO-based hybrid detection to complement geometric contouring.

Natural language understanding through LLMs proved adaptable but error-prone in handling ambiguous, accent-affected inputs. Prompt tuning, fallback triggers, and user-specific memory helped mitigate hallucination, though latency remained high—up to 10 seconds for larger models. Gesture-enhanced interpretation offered improvement but required better input synchronization and disambiguation logic.

Simulation-to-physical synchronization between Omniverse Isaac Sim and the ABB YuMi robot achieved high fidelity, with 96% success in simulation and 91% in real execution. Minor deviations in grasping precision were linked to calibration drift and joint constraints on the dual-arm platform. The digital twin was nonetheless effective in validating behaviours before hardware deployment.

Authentication via face and voice, supported by FAISS indexing, consistently matched users with 100% accuracy for the registered set. However, personalization memory was only partially tested due to reset needs during iterative testing. Still, the presence of authentication context improved task specificity, confirming the utility of user-linked profiles.

Several failures and limitations were identified:

- Computational overloads during simulation slowed pipeline responsiveness.
- Frequent restarts of Isaac Sim were necessary due to environment instability.
- Voice input misinterpretation under noise or accent variations revealed the need for multimodal fallback mechanisms.

User feedback was largely positive regarding interface clarity and task logic, although delays in physical execution and robotic movement constraints were noted. The GUI was intuitive but lacked error notifications and real-time status indicators—features recommended for future work.

Critically, the LangGraph-based task orchestration layer—though modular and conceptually elegant—proved complex in managing real-time state transitions. A more tightly coupled event-state architecture is recommended to support fault tolerance and smoother task chaining.

# 6. Conclusion & Future Work

## 6.1. Summary of Key Contributions

This thesis presented a modular framework for integrating computer vision, large language models (LLMs), and robotic control to enable personalised task execution in simulated and real-world environments. The proposed system was designed and implemented to interpret natural language and gesture-based instructions, authenticate users via biometric modalities, perceive the surrounding environment through computer vision, and plan robot tasks using LLM-driven reasoning. A relational database underpinned the system, facilitating task orchestration, memory-based personalization, and real-time data exchange between modules.

Through over 50 iterative testing sessions across two primary tasks—slide sorting and shape stacking—the system demonstrated a reliable end-to-end execution pipeline. The vision module maintained real-time performance with 30 FPS detection, while face and voice authentication using FAISS delivered 100% accuracy within the registered user group. Task plans were executed with high fidelity in Omniverse Isaac Sim and mirrored in the ABB YuMi cobot, affirming the viability of sim-to-real synchronization. Although command parsing success started at 50% on first attempt, it improved significantly as prompts were refined, and fallback strategies were introduced.

Key contributions of this work include:

1. A multi-modal human-robot interaction interface combining voice, gesture, and face/voice authentication.
2. A hybrid LLM-based planning architecture, leveraging user context for disambiguation and personalization.
3. A unified PostgreSQL-backed task manager enabling system-wide traceability and modular extensibility.
4. Successful demonstration of digital twin synchronization between simulation and physical execution.

Overall, the system lays a foundational step toward more adaptive, context-aware collaborative robots capable of naturalistic interaction and personalised behaviour. It validates the potential of LLMs not just as language interpreters but as cognitive engines for robot task planning—when grounded with real-world sensory data and structured memory.

## 6.2. Limitations

Despite its successes, the system exhibited several limitations:

1. Latency during LLM-based inference affected real-time responsiveness of user interactions, especially with large models like the *Llama3.3:latest (see* Figure 3-3 *)*.
2. Accent variability and ambient noise occasionally impaired voice input accuracy, underscoring the need for robust speech filtering.
3. Simulation instability in Omniverse Isaac Sim occasionally required system restarts, impacting testing efficiency.
4. Hardware constraints with the dual-arm ABB YuMi robot introduced motion limitations that are not present in the simulation pipeline.
5. Limited personalization persistence due to iterative UI resets meant that long-term user modelling was only partially validated.
6. These limitations suggest areas where further technical refinement and testing under more diverse conditions are necessary.

## 6.3. Future Work

Building on the foundation established in this thesis, several future directions are recommended:

1. Full Deployment of Persistent Personalization

   Extend the personalization module to support persistent user memory across sessions, enabling adaptive long-term behaviour based on cumulative interaction history.

2. Hybrid Vision Architecture

   Integrate object classification models (e.g., YOLO) with geometric feature extraction to enhance object identification, robustness under occlusion, and semantic awareness.

3. Scalable Multi-Agent State Management

   Redesign LangGraph orchestration logic to support more structured state propagation and transitions, minimizing ambiguity in multi-step task execution.

4. Real-Time Performance Optimization

   Deploy lighter-weight LLMs with edge-tuned inference pipelines or apply on-device quantization for latency reduction in voice-command-to-action processing.

5. Expanded User Study:

   Conduct a broader user study across multiple demographics to evaluate system generalizability, fairness in authentication, and subjective user satisfaction.

By addressing these areas, the system can evolve into a fully deployable framework suitable for industrial, healthcare, or domestic applications that demand adaptive, human-aware robotic collaboration.

# References


[1] 'CES - The Most Powerful Tech Event in the World'. Accessed: Mar. 05, 2025. [Online]. Available: https://www.ces.tech/

[2] 'Enchanted Tools / Changing the face of robotics', Enchanted Tools. Accessed: Mar. 10, 2025. [Online]. Available: https://enchanted.tools/

[3] C. Young, 'New-look Mirokai robot assists hospital patients, serves as concierge', Interesting Engineering. Accessed: Mar. 10, 2025. [Online]. Available: https://interestingengineering.com/ces-2025/new-look-mirokai-robot-assists-patients

[4] M. Shridhar, L. Manuelli, and D. Fox, 'CLIPort: What and Where Pathways for Robotic Manipulation', in *Proceedings of the 5th Conference on Robot Learning*, PMLR, Jan. 2022, pp. 894–906. Accessed: Mar. 05, 2025. [Online]. Available: https://proceedings.mlr.press/v164/shridhar22a.html

[5] T. R. R. Staff, 'IFR World Robotics report says 4M robots are operating in factories globally', The Robot Report. Accessed: Mar. 10, 2025. [Online]. Available: https://www.therobotreport.com/ifr-4-million-robots-operating-globally-world-robotics-report/

[6] S. Riso, D. Adascalitei, and European Foundation for the Improvement of Living and Working Conditions, Eds., *Human-robot interaction: what changes in the workplace?* Luxembourg: Publications Office, 2024. doi: 10.2806/67956.

[7] 'Human-Centered Robots Are Key to Creating a Society We All Want to Live In', NTT DATA. Accessed: Mar. 12, 2025. [Online]. Available: https://www.nttdata.com/global/en/insights/technology/academic-collaborations/mit-media-lab-cynthia-breazeal

[8] C. Breazeal, K. Dautenhahn, and T. Kanda, 'Social Robotics', in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds., Cham: Springer International Publishing, 2016, pp. 1935–1972. doi: 10.1007/978-3-319-32552-1_72.

[9] 'Cynthia Breazeal', *Wikipedia*. Mar. 09, 2025. Accessed: Mar. 10, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cynthia_Breazeal&oldid=1279548619

[10] 'Digitalsheet_A4_World_Robotics_2023_low.pdf'. Accessed: Mar. 05, 2025. [Online]. Available: https://ifr.org/img/worldrobotics/Digitalsheet_A4_World_Robotics_2023_low.pdf

[11] 'World Economic Forum Annual Meeting 2025', World Economic Forum. Accessed: Mar. 05, 2025. [Online]. Available: https://www.weforum.org/meetings/world-economic-forum-annual-meeting-2025/

[12] G. Jocher, J. Qiu, and A. Chaurasia, *Ultralytics YOLO*. (Jan. 2023). Python. Accessed: Mar. 05, 2025. [Online]. Available: https://github.com/ultralytics/ultralytics

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', May 24, 2019, *arXiv*: arXiv:1810.04805. doi: 10.48550/arXiv.1810.04805.

[14] 'Introducing GPT-4.5'. Accessed: Mar. 05, 2025. [Online]. Available: https://openai.com/index/introducing-gpt-4-5/

[15] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, 'Robots that use language', *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 25–55, 2020.

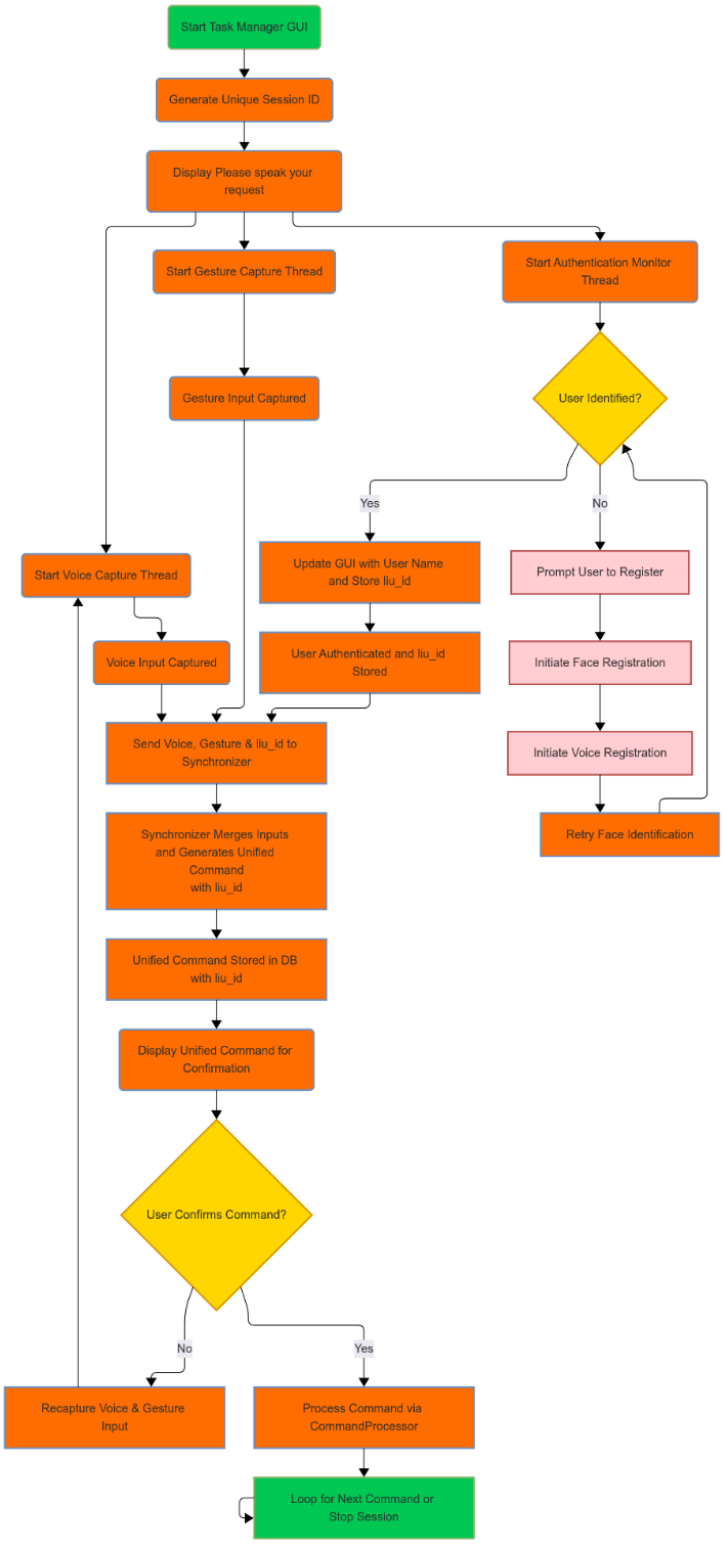
[16] M. Quigley *et al.*, 'ROS: an open-source Robot Operating System'.

[17] 'NVIDIA Omniverse', NVIDIA Docs. Accessed: Mar. 05, 2025. [Online]. Available: https://docs.nvidia.com/omniverse/index.html
[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel, 'End-to-End Training of Deep Visuomotor Policies', Apr. 19, 2016, *arXiv*: arXiv:1504.00702. doi: 10.48550/arXiv.1504.00702.
[19] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, 'Domain randomization for transferring deep neural networks from simulation to the real world', in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 23–30. doi: 10.1109/IROS.2017.8202133.
[20] 'GTC 2023 Keynote | GTC Digital Spring 2023 | NVIDIA On-Demand', NVIDIA. Accessed: Mar. 05, 2025. [Online]. Available: https://www.nvidia.com/en-us/on-demand/session/gtcspring23-s52226/
[21] 'NVIDIA GTC 2023 to Feature Latest Advances in AI Computing Systems, Generative AI, Industrial Metaverse, Robotics; Keynote by Jensen Huang; Talks by OpenAI, DeepMind Founders', NVIDIA Newsroom. Accessed: Mar. 05, 2025. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-gtc-2023-to-feature-latest-advances-in-ai-computing-systems-generative-ai-industrial-metaverse-robotics-keynote-by-jensen-huang-talks-by-openai-deepmind-founders
[22] 'Privacy in the age of robotics | IAPP'. Accessed: Mar. 11, 2025. [Online]. Available: https://iapp.org/news/a/privacy-in-the-age-of-robotics
[23] Y. Zhou and H. Li, 'A Scientometric Review of Soft Robotics: Intellectual Structures and Emerging Trends Analysis (2010–2021)', *Front. Robot. AI*, vol. 9, May 2022, doi: 10.3389/frobt.2022.868682.
[24] G. C. Devol, 'Programmed Article Transfer', US 2,988,237, 1961
[25] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. in Springer Handbooks. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-32552-1.
[26] M. A. Goodrich and A. C. Schultz, 'Human–Robot Interaction: A Survey', *HCI*, vol. 1, no. 3, pp. 203–275, Jan. 2008, doi: 10.1561/1100000005.
[27] 'WEF_Frontier_Technologies_in_Industrial_Operations_2025.pdf'. Accessed: Mar. 05, 2025. [Online]. Available: https://reports.weforum.org/docs/WEF_Frontier_Technologies_in_Industrial_Operations_2025.pdf
[28] K. Dautenhahn, 'Socially intelligent robots: dimensions of human–robot interaction', *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 362, no. 1480, pp. 679–704, Feb. 2007, doi: 10.1098/rstb.2006.2004.
[29] M. Stolarz, A. Mitrevski, M. Wasil, and P. G. Plöger, 'Learning-based personalisation of robot behaviour for robot-assisted therapy', *Front. Robot. AI*, vol. 11, Apr. 2024, doi: 10.3389/frobt.2024.1352152.
[30] J. Redmon and A. Farhadi, 'YOLOv3: An Incremental Improvement', Apr. 08, 2018, *arXiv*: arXiv:1804.02767. doi: 10.48550/arXiv.1804.02767.
[31] Ultralytics, 'RT-DETR vs YOLOv6-3.0: Uma comparação pormenorizada de modelos'. Accessed: Feb. 22, 2025. [Online]. Available: https://docs.ultralytics.com/pt/compare/rtdetr-vs-yolov6
[32] G. Jocher and J. Qiu, *Ultralytics YOLO11*. (2024). [Online]. Available: https://github.com/ultralytics/ultralytics
[33] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, 'End-to-End Object Detection with Transformers', May 28, 2020, *arXiv*: arXiv:2005.12872. doi: 10.48550/arXiv.2005.12872.
[34] 'The science behind visual ID', Amazon Science. Accessed: Feb. 22, 2025. [Online]. Available: https://www.amazon.science/blog/the-science-behind-visual-id

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, 'You Only Look Once: Unified, Real-Time Object Detection', May 09, 2016, *arXiv*: arXiv:1506.02640. doi: 10.48550/arXiv.1506.02640.
[36] R. Xu, F.-J. Chu, C. Tang, W. Liu, and P. A. Vela, 'An Affordance Keypoint Detection Network for Robot Manipulation', *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2870–2877, Apr. 2021, doi: 10.1109/LRA.2021.3062560.
[37] N. Yamanobe *et al.*, 'A brief review of affordance in robotic manipulation research', *Advanced Robotics*, vol. 31, no. 19–20, pp. 1086–1101, Oct. 2017, doi: 10.1080/01691864.2017.1394912.
[38] F. Schroff, D. Kalenichenko, and J. Philbin, 'FaceNet: A Unified Embedding for Face Recognition and Clustering', in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 815–823. doi: 10.1109/CVPR.2015.7298682.
[39] 'Sensory's TrulySecure Face & Voice Biometrics Software', Sensory. Accessed: Feb. 22, 2025. [Online]. Available: https://www.sensory.com/face-voice-biometrics/
[40] F. Zeng, W. Gan, Y. Wang, N. Liu, and P. S. Yu, 'Large Language Models for Robotics: A Survey', Nov. 13, 2023, *arXiv*: arXiv:2311.07226. doi: 10.48550/arXiv.2311.07226.
[41] J. Wang *et al.*, 'Large Language Models for Robotics: Opportunities, Challenges, and Perspectives', Jan. 09, 2024, *arXiv*: arXiv:2401.04334. doi: 10.48550/arXiv.2401.04334.
[42] D. Driess *et al.*, 'PaLM-E: An Embodied Multimodal Language Model', Mar. 06, 2023, *arXiv*: arXiv:2303.03378. doi: 10.48550/arXiv.2303.03378.
[43] Google DeepMind, 'Demonstrating Large Language Models on Robots', in *Robotics: Science and Systems XIX*, Robotics: Science and Systems Foundation, Jul. 2023. doi: 10.15607/RSS.2023.XIX.024.
[44] Z. Hu *et al.*, 'Deploying and Evaluating LLMs to Program Service Mobile Robots', *IEEE Robot. Autom. Lett.*, vol. 9, no. 3, pp. 2853–2860, Mar. 2024, doi: 10.1109/LRA.2024.3360020.
[45] H. Choi *et al.*, 'On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward', *Proceedings of the National Academy of Sciences*, vol. 118, no. 1, p. e1907856118, Jan. 2021, doi: 10.1073/pnas.1907856118.
[46] S. Nambiar, M. Jonsson, and M. Tarkian, 'Automation in Unstructured Production Environments Using Isaac Sim: A Flexible Framework for Dynamic Robot Adaptability', *Procedia CIRP*, vol. 130, pp. 837–846, Jan. 2024, doi: 10.1016/j.procir.2024.10.173.
[47] C. K. Liu and D. Negrut, 'The Role of Physics-Based Simulators in Robotics', *Annu. Rev. Control Robot. Auton. Syst.*, vol. 4, no. 1, pp. 35–58, May 2021, doi: 10.1146/annurev-control-072220-093055.
[48] L. Žlajpah, 'Simulation in robotics', *Mathematics and Computers in Simulation*, vol. 79, no. 4, pp. 879–897, Dec. 2008, doi: 10.1016/j.matcom.2008.02.017.
[49] A. Afzal, D. S. Katz, C. L. Goues, and C. S. Timperley, 'A Study on the Challenges of Using Robotics Simulators for Testing', Apr. 15, 2020, *arXiv*: arXiv:2004.07368. doi: 10.48550/arXiv.2004.07368.
[50] R. E. Fikes and N. J. Nilsson, 'Strips: A new approach to the application of theorem proving to problem solving', *Artificial Intelligence*, vol. 2, no. 3–4, pp. 189–208, Dec. 1971, doi: 10.1016/0004-3702(71)90010-5.
[51] J. Tang, Z. Ye, Y. Yan, Z. Zheng, T. Gao, and Y. Jin, 'Zero-shot Robotic Manipulation with Language-guided Instruction and Formal Task Planning', Jan. 25, 2025, *arXiv*: arXiv:2501.15214. doi: 10.48550/arXiv.2501.15214.
[52] Z. Wu, Z. Wang, X. Xu, J. Lu, and H. Yan, 'Embodied Task Planning with Large Language Models', Jul. 04, 2023, *arXiv*: arXiv:2307.01848. doi: 10.48550/arXiv.2307.01848.
[53] M. Mykhailov, 'Accelerate with Rapid Application Development: Use Cases and Pros – NIX United', NIX United – Custom Software Development Company in US. Accessed: May 23, 2025. [Online]. Available: https://nix-united.com/blog/the-ultimate-guide-to-rapid-application-development/

# Appendix

## Appendix A – Workflow – Approach II



# Appendix B – System Algorithms

**1:** End-to-End Vision-Language-Robot Execution Flow

**Result:** Personalized Task Execution in Simulation or Physical Robot

**Initialization:**
Load Web GUI, RealSense Camera, Microphone, LLM, PostgreSQL, LangGraph Subgraphs;
Initialize ABB YuMi Digital Twin in Isaac Sim;
**while** *System is running* **do**
  **Start User Session:**
  Launch GUI, activate webcam and microphone, and initialize SessionManager;
  **Authenticate User:**
  Capture face image via webcam and match using face recognition with `users` table;
  Capture voice via microphone and match with stored voice encodings;
  Log successful authentication to `access_logs`;
  **Capture User Command:**
  Record and transcribe voice using Whisper model;
  Optionally capture gesture input and store in `gesture_instructions`;
  Store all inputs to `unified_instructions` table;
  **Classify Intent:**
  Use LLM-based classifier to determine intent: general query, scene query, action task, or triggered task;
  **Process Intent via Subgraphs:**
  **if** *General Query* **then**
    Use LLM to respond using `interaction_memory` and user profile;
  **else if** *Scene Query* **then**
    Fetch data from `camera_vision`, format as prompt, and generate scene summary;
  **else if** *Action Task* **then**
    Generate operation plan using `operation_sequence` and `op_parameters`;
    Retrieve necessary data from `camera_vision` and `usd_data`;
  **else if** *Triggered Task* **then**
    Set `trigger = TRUE` for matched task in `operation_library`;
  **Execute Task Plan:**
  Send planned steps to Isaac Sim;
  ABB YuMi replicates task in physical space;
  Log execution results to `simulation_results` and `interaction_memory`;
  **Loop and Wait:**
  Return response to GUI or TTS output;
  Wait for next command;

**2:** Tray and Slide Detection Algorithm

**Result:** Detect and log trays, holders, and slides to PostgreSQL

**Initialize RealSense pipeline** with depth and color streams;
Retrieve camera intrinsics and depth scaling factor;
**while** *streaming* **do**
    Capture color and depth frames;
    Convert color frame to grayscale and apply Gaussian blur;
    Threshold the image and extract contours;
    **foreach** *contour* **do**
        **if** *area* $\in$ *tray range* **then**
            Compute rotated bounding box and label corners as POINT 0–3;
            Calculate angle with respect to x-axis;
            Divide tray region into segments and compute midpoints;
            Detect colored slides and match them to midpoints;
            Estimate tray position relative to screw using depth;
        **if** *area* $\in$ *holder range* **then**
            Similar processing as tray;
            Estimate holder angle and position w.r.t. screw;
    **Database Update:**
    Store tray, holder, and slide positions and orientations in `camera_vision` table;
    **if** *user presses 'q'* **then**
        Exit loop and release camera resources;

**3:** Colored Shape Detection with Screw-Based Referencing

**Result:** Detect and log colored geometric shapes with spatial reference

**Initialize Camera and Depth Pipeline** using RealSense API;
**while** *System is running* **do**
    Capture color and depth frames from camera;
    Identify table ROI and extract image region;
    Extract screw ROI and detect orange and blue screws;
    **if** *both screws detected* **then**
        Compute scaling factor using screw-to-screw distance;
        Set orange screw as origin;
    **else**
        **if** *last-known screws available and valid* **then**
            Reuse previous screw positions within timeout window;
        **else**
            Skip frame due to missing reference;
    Apply mean-shift segmentation on ROI;
    Generate binary masks using HSV color ranges;
    Detect contours and filter by area thresholds;
    **foreach** *contour* **do**
        Classify geometric shape (circle, triangle, etc.);
        Determine shape color from HSV average;
        Compute center point and orientation;
        Calculate real-world position relative to screw origin;
        **if** *matching previous object* **then**
            Update existing object entry and history;
        **else**
            Create new object entry with unique name;
        Store shape metadata in `camera_vision` table: `object_name, object_color, color_code, pos_x, pos_y, pos_z, rot_z`;
    Overlay detected shapes and annotations on output image;
    Display visual result with shape names and depth info;
    **if** *user presses 'q'* **then**
        exit loop;

**4:** Face Authentication and User Registration Flow

**Result:** Authenticate user via Face Recognition (with fallback to registration)

**Initialize FaceAuthSystem and FAISS Index**;
Load known encodings and user records from database;
**for** *each frame during identification phase* **do**
    Detect faces and compute encodings;
    **foreach** *detected face* **do**
        Compare with stored embeddings using FAISS;
        **if** *similarity* $\geq$ *threshold* **then**
            Retrieve user info and welcome user;
        **else**
            Flag face as unknown;
**if** *unknown face detected* **then**
    Prompt user for registration;
    Capture face and user details;
    Save encoding and profile image to database;
    Refresh FAISS index;

**5:** Voice Authentication with Embedding Matching

**Result:** Authenticate user by comparing voice sample to stored embedding

Prompt user to record voice sample using microphone;
Transcribe speech to validate known phrase;
If transcription matches: Preprocess audio and compute embedding using Resemblyzer;
Retrieve stored voice embedding from database or file;
Compare new vs stored embedding using cosine similarity;
**if** *similarity* $\geq$ *threshold* **then**
    Grant voice authentication;
**else**
    Deny access and prompt retry;

**6:** LLM-Based Task Classification and Execution Flow

**Result:** Classify and process user command with LLM + LangGraph Subgraphs

Record user voice or gesture input and transcribe to text;
Store unified instruction in database;
Use LLM classifier to infer intent category;
**if** *General Query* **then**
    Query interaction memory + user profile;
    Generate and return assistant response;
**else if** *Scene Query* **then**
    Fetch camera_vision context;
    Summarize visible environment via LLM;
**else if** *Action Task* **then**
    Match command to task from operation_library;
    Generate execution plan via planner node;
    Retrieve real-world coordinates for objects involved;
**else if** *Triggered Task* **then**
    Set trigger flag in database to activate external script;
Return response to user via GUI or TTS;

**7:** Web-Based GUI Interaction Flow via FastAPI

**Result:** Handle user interaction via FastAPI Web Interface

Start FastAPI server with routes and template rendering;
Render GUI using HTML/CSS template with chat display, input box, buttons;
**foreach** *user input event* **do**
    Route input to backend endpoint (e.g., /process or /status);
    Store message in conversation history;
    Forward input to SessionManager or LLM handler;
    Return assistant reply to front-end;
    **if** *response includes task trigger* **then**
        Update relevant database flags or call downstream module;
    Render response on chat interface with updated context;

**8:** Session Management and Routing Flow

**Result:** Manage session, authentication, and input routing

**Start Session:**
Check user authentication (face or voice);
Assign session ID and load interaction memory;
**During Session:**
Accept user input (voice or gesture);
Store in unified_instructions and log access;
Determine active user context and preferences;
Pass input to LLM-based orchestrator for further classification;
**End Session:**
Optionally clear memory or update session log in database;

**9:** LangGraph Intent Routing and Subgraph Execution

**Result:** Route user command to correct LangGraph Subgraph

Start orchestration from root node;
Pass instruction to LLM classifier node to determine type;
**if** *GeneralQuery* **then**
    Route to general_node with chat history;
**else if** *SceneQuery* **then**
    Route to scene_node and inject vision data into prompt;
**else if** *ActionTask* **then**
    Route to planner_node with command + context from database;
**else if** *Unknown* **then**
    Route to fallback_node or error handler;
Return output state with 'reply' field and any 'plan' updates;

**10:** Gesture Processing and Instruction Logging

**Result:** Capture and classify gesture input for user interaction

Initialize webcam feed and load gesture detection model;
**while** *system is active* **do**
    Capture video frame and apply hand landmark detection;
    Compute gesture features (e.g., finger positions, angles);
    Match features to predefined gesture templates from `gesture_library`;
    **if** *gesture is recognized* **then**
        Log gesture type and insert into `gesture_instructions` table;
        Optionally trigger associated intent or command;
    **if** *gesture changes or timeout* **then**
        Reset detection buffer and await next input;

**11:** Omniverse Simulation and Physical Synchronization

**Result:** Synchronize Omniverse simulation with physical robot execution

Start Omniverse Isaac Sim environment;
Connect to ABB YuMi digital twin via ROS bridge;
**while** *task plan exists* **do**
    Fetch next action step from `operation_sequence` table;
    Parse USD object info and fetch 3D scene context;
    Simulate action in Omniverse and verify success;
    **if** *simulation OK* **then**
        Send real-time control command to ABB YuMi via ROS;
        Wait for completion acknowledgment; Log result in `simulation_results` table;
    **else**
        Re-plan or adjust based on failure feedback;

# Appendix C – Major Database Tables

Some tables from the database:

*camera_vision* table of entries logged during the experiment.

| object_id | object_name | object_color | color_code | pos_x | pos_y | pos_z | rot_x | rot_y | rot_z | usd_name | last_detected |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 24891 | Fixture | black | [ 0, 0, 0 ] | 59.685603 | 123.63446 | 0 | 0 | 0 | 92.290610 | Fixture.usd | 2025-05-20 23:22:... |
| 24892 | Holder | black | [ 0, 0, 0 ] | 91.660034 | 322.94174 | 0 | 0 | 0 | -77.52283 | Slide_Holder.usd | 2025-05-20 23:22:... |
| 24893 | Slide_1 | green | [ 0.28, 0.4... | 69.685603 | 128.63446 | 0 | 0 | 0 | 92.290610 | Slide.usd | 2025-05-20 23:22:... |
| 24894 | Slide_2 | orange | [ 0.81, 0.6... | 79.685603 | 133.63446 | 0 | 0 | 0 | 92.290610 | Slide.usd | 2025-05-20 23:22:... |
| 24895 | Slide_3 | pink | [ 0.61, 0.2... | 89.685603 | 138.63446 | 0 | 0 | 0 | 92.290610 | Slide.usd | 2025-05-20 23:22:... |

*isaac_sim_gui* table

| sequence_id | gui_feature | operation_status |
|---|---|---|
| 2 | Reset | FALSE |
| 3 | Load | FALSE |
| 1 | Start | TRUE |

*operation_sequence* table entries.

| id | operation_id | sequence_id | sequence_name | object_name | command_id | processed | execution_time |
|---|---|---|---|---|---|---|---|
| 245 | 1 | 1 | pick | square_1 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 246 | 2 | 2 | travel | square_1 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 247 | 3 | 3 | drop | square_1 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 248 | 4 | 1 | pick | circle_2 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 249 | 5 | 2 | travel | circle_2 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 250 | 6 | 3 | drop | circle_2 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 251 | 7 | 1 | pick | circle_1 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 252 | 8 | 2 | travel | circle_1 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 253 | 9 | 3 | drop | circle_1 | 46 | FALSE | 2025-05-16 13:14:40.788568 |
| 254 | 10 | 6 | go_home | | 46 | FALSE | 2025-05-16 13:14:40.788568 |

*Users* table

| user_id | first_name | last_name | liu_id | email | prefere... | profile_image_p... | interaction_memory | face_encoding | voice_embedding | role | created_at | last_updated |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Yumi | Robot | yumi100 | yumi100@lab.liu.ai | {"likes": ["AI", ... | /images/yumi001.jpg | {"last_task": "Assistance", "successful... | NULL | NULL | robot | 2025-05-07 1... | 2025-05-07 10:13:... |
| 2 | Oscar | Ikechukwu | oscik559 | oscik559@student.liu.se | {"likes": ["AI", ... | /images/oscik559.jpg | {"last_task": "Pick object", "successfu... | { "type": "Buf... | { "type": "Buffer... | admin | 2025-05-07 1... | 2025-05-07 10:13:... |
| 3 | Rahul | Chiramel | rahch515 | rahch515@student.liu.se | {"likes": ["Aer... | /images/rahch515.jpg | {"last_task": "Screw object", "success... | { "type": "Buf... | NULL | admin | 2025-05-07 1... | 2025-05-07 10:13:... |
| 7 | Aref | Aghaee | areag806 | areag806@student.liu.se | {"likes": ["CAT... | /images/areag806.jpg | {"last_task": "Pick object", "successfu... | NULL | NULL | guest | 2025-05-07 1... | 2025-05-07 10:13:... |
| 9 | Hamideh | Pourrasoul | hampo845 | hampo845@student.liu.se | {"likes": ["CAT... | /images/hampo845.jpg | {"last_task": "Pick object", "successfu... | NULL | NULL | guest | 2025-05-07 1... | 2025-05-07 10:13:... |
| 8 | Thomson | Kalliyath | thoka981 | thoka981@student.liu.se | {"likes": ["Om... | /images/thoka981.jpg | {"last_task": "Pick object", "successfu... | NULL | NULL | guest | 2025-05-07 1... | 2025-05-07 10:13:... |
| 4 | Sanjay | Nambiar | sanna58 | sanjay.nambiar@liu.se | {"likes": ["Pro... | /images/sanna58.jpg | {"last_task": "Slide object", "successf... | NULL | NULL | admin | 2025-05-07 1... | 2025-05-07 10:13:... |
| 6 | Marie | Jonsson | marjo33 | marie.s.jonsson@liu.se | {"likes": ["Rob... | /images/marjo33.jpg | {"last_task": "Fix robot battery", "suc... | NULL | NULL | team | 2025-05-07 1... | 2025-05-07 10:13:... |
| 5 | Mehdi | Tarkian | mehta77 | mehdi.tarkian@liu.se | {"likes": ["Run... | /images/mehta77.jpg | {"last_task": "Drop object", "successf... | NULL | NULL | team | 2025-05-07 1... | 2025-05-07 10:13:... |
| 10 | John | Ashish | johas759 | johas759@student.liu.se | {"likes": ["pyt... | /images/johas759.jpg | {"last_task": "Pick object", "successfu... | NULL | NULL | guest | 2025-05-07 1... | 2025-05-07 10:13:... |
| 11 | Danial | Nikpey | danni741 | danni741@student.liu.se | {"likes": ["vb.... | /images/danni741.jpg | {"last_task": "Pick object", "successfu... | NULL | NULL | guest | 2025-05-07 1... | 2025-05-07 10:13:... |
| 23 | Sanjay | Nambiar | sanna058 | sanna058@student.liu.se | [] | C:\Users\oscik559\Pro... | [] | { "type": "Buf... | { "type": "Buffer... | guest | 2025-05-16 1... | 2025-05-16 11:43:... |

*operation_library* table

| id | operation_name | task_order | description | trigger_keywords | script_path | is_triggerable | trigger | state | last_triggered |
|---|---|---|---|---|---|---|---|---|---|
| 1 | slide_sorting | pick, travel, drop | Sort slides by shape and color into trays | [ "sort", "slides", "... | detect_slides_pgSQL.py | TRUE | FALSE | idle | 2025-05-16 11:25:08.792318 |
| 2 | shape_stacking | pick, travel, drop | Stack blocks of shapes based on their type and color | [ "stack", "stacking"... | detect_shapes_pgSQL.py | TRUE | TRUE | triggered | 2025-05-16 13:14:05.784731 |

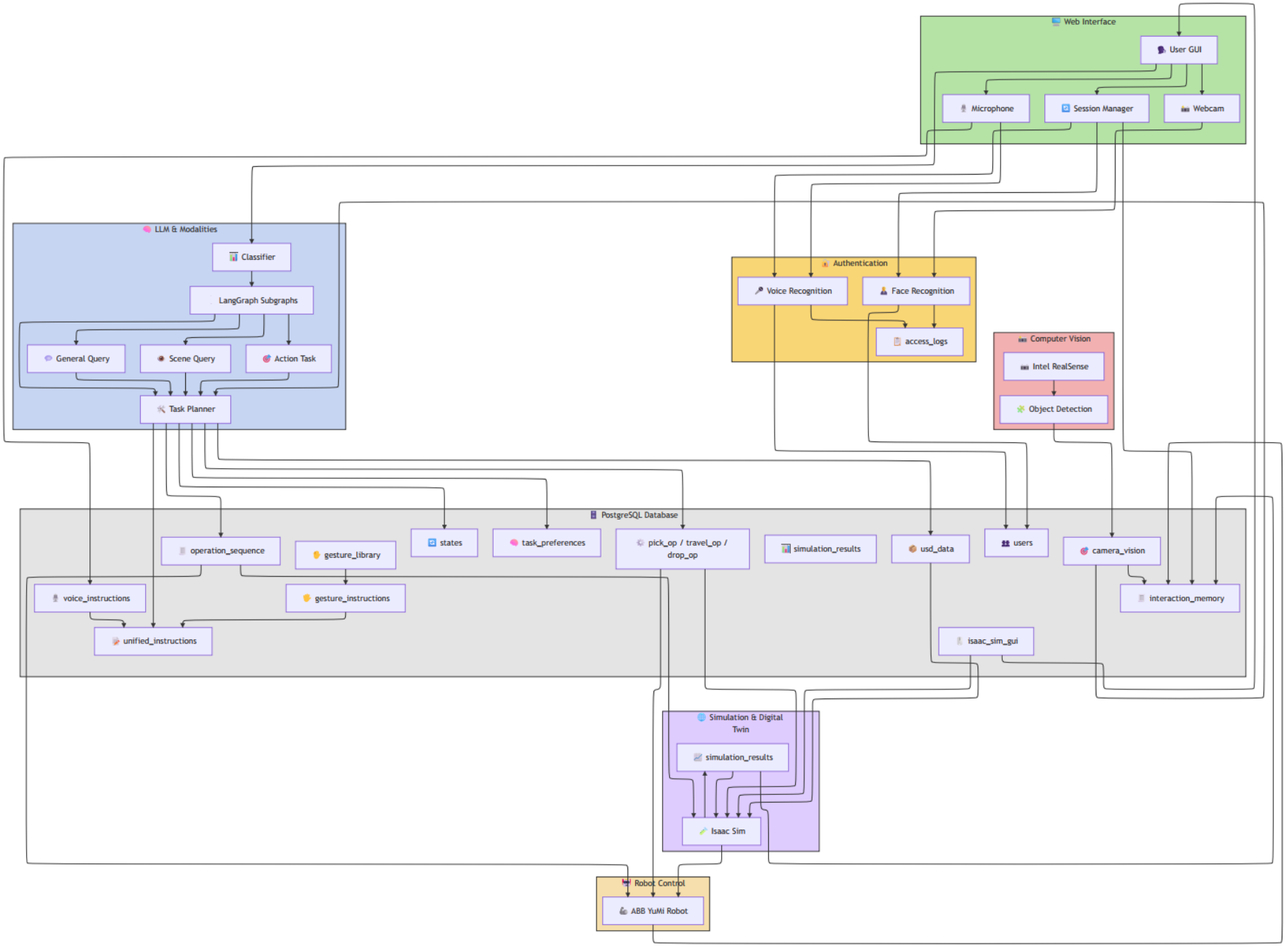
*gesture_library* table

| id | gesture_type | gesture_text | natural_description | config |
|---|---|---|---|---|
| 1 | thumbs_up | Approval | The thumb is raised above the index finger. | { "threshold": 0 } |
| 2 | open_hand | Stop | All fingers are extended, signaling stop. | { "threshold": 0 } |
| 3 | pointing | Select Object | The index finger is extended while other fingers are curled. | { "threshold": 0 } |
| 4 | closed_fist | Grab | The hand is clenched into a fist. | { "threshold": 0 } |
| 5 | victory | Confirm | The hand forms a V-shape with the index and middle fingers extended. | { "threshold": 0 } |
| 6 | ok_sign | OK | The thumb and index finger are touching to form a circle. | { "threshold": 0.... |

*unified_instructions* table

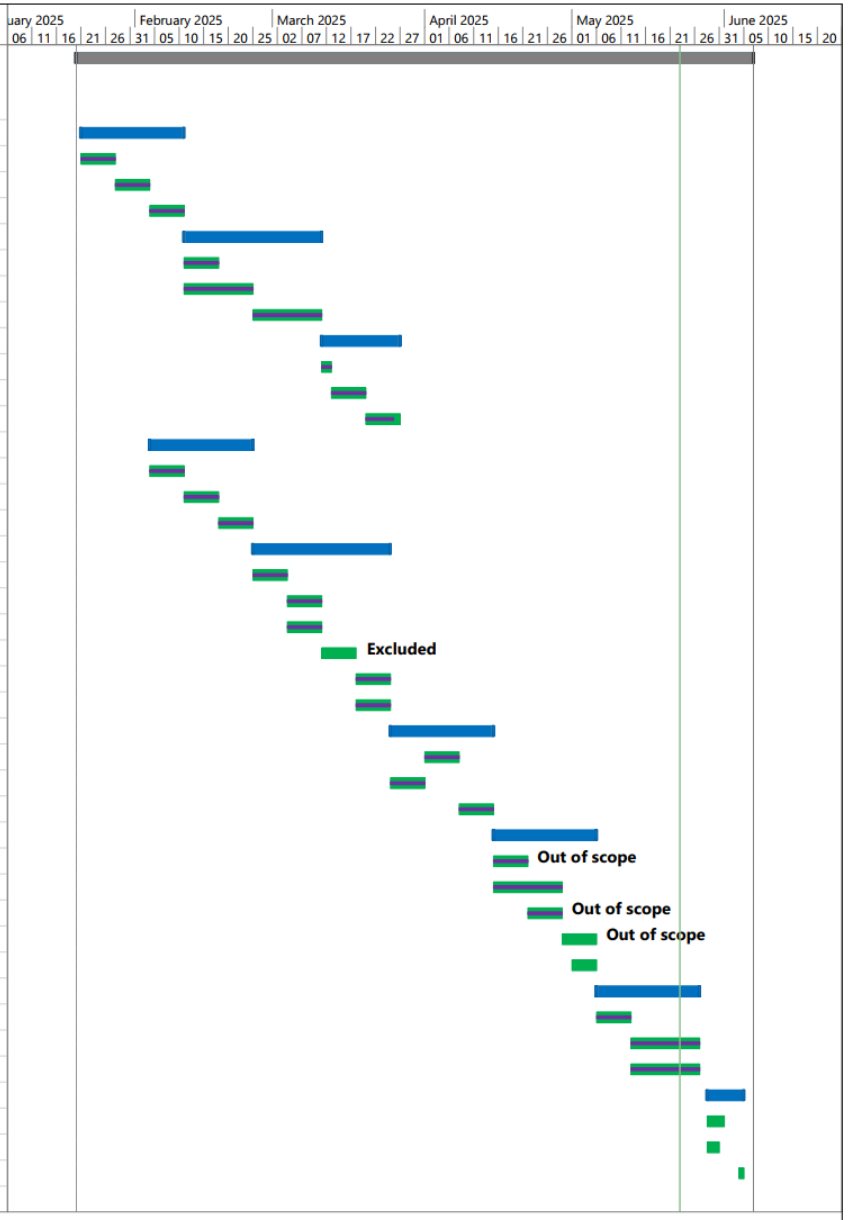
| id | session_id | timestamp | liu_id | voice_command | gesture_command | unified_command | confidence | processed |
|---|---|---|---|---|---|---|---|---|
| 1 | session_voice_001 | 2025-05-09 07:... | oscik559 | sort the slides in order of orange s... | | sort the slides in order of orange slide and then green slide | 0.95 | TRUE |
| 2 | session_voice_001 | 2025-05-09 08:... | oscik559 | Sort the slides in order of orange ... | | Sort the slides in order of orange slide and then green slide | 0.95 | TRUE |
| 3 | session_voice_001 | 2025-05-09 08:... | oscik559 | Sort the slides in order of orange ... | | Sort the slides in order of orange slide and then green slide | 0.95 | TRUE |
| 4 | session_voice_001 | 2025-05-09 15:... | oscik559 | sort the slides in the order of gree... | | sort the slides in the order of green, orange | 0.95 | TRUE |
| 5 | session_voice_001 | 2025-05-10 01:... | oscik559 | sort the slides in order of green sli... | | sort the slides in order of green slide and orange slide | 0.95 | TRUE |
| 6 | session_voice_001 | 2025-05-10 01:... | oscik559 | sort the slides in order of green sli... | | sort the slides in order of green slide and orange slide and then finally ... | 0.95 | TRUE |
| 7 | session_voice_001 | 2025-05-10 03:... | oscik559 | tell me a joke | | tell me a joke | 0.95 | FALSE |
| 8 | session_voice_001 | 2025-05-10 03:... | oscik559 | Tell me a joke | | Tell me a joke | 0.95 | FALSE |
| 9 | session_voice_001 | 2025-05-10 04:... | oscik559 | Now i need to do the same in des... | | Now i need to do the same in descending order | 0.95 | FALSE |

# Appendix D – Integration of System Components



# Appendix E – Updated Project Plan

| ID | | WBS | Task Name | Duration | Start | Finish | Deliverable |
|---|---|---|---|---|---|---|---|
| 0 | | **0** | **Integrating Vision, Language Models, and Robotic Control for Personalized Task Execution in Virtual Environments.** | **100 days** | **20 Jan** | **06 Jun** | |
| 1 | ✓ | **1** | **Project Planning & Setup** | **15 days** | **21 Jan** | **10 Feb** | Approved planning report |
| 2 | ✓ | 1.1 | Define scope & objectives | 1 wk | 21 Jan | 27 Jan | Scope document |
| 3 | ✓ | 1.2 | Set up tools | 1 wk | 28 Jan | 03 Feb | Tools ready |
| 4 | ✓ | 1.3 | Report: Introduction | 1 wk | 04 Feb | 10 Feb | Chapter 1 of report completed |
| 5 | ✓ | **2** | **Literature Review & Background Study** | **20 days** | **11 Feb** | **10 Mar** | Literature review notes, reference list |
| 6 | ✓ | 2.1 | Study SOTA in Vision, LLM, Robot Control | 1 wk | 11 Feb | 17 Feb | Summaries of relevant work |
| 7 | ✓ | 2.2 | Identify gaps & re-formulate RQs | 2 wks | 11 Feb | 24 Feb | Updated research questions |
| 8 | ✓ | 2.3 | Report: Literature review | 2 wks | 25 Feb | 10 Mar | Chapter 2 of report completed |
| 9 | | **3** | **Requirements & System Architecture** | **12 days** | **11 Mar** | **26 Mar** | Architecture diagram |
| 10 | ✓ | 3.1 | Draft requirements & system architecture | 2 days | 11 Mar | 12 Mar | Requirements checklist |
| 11 | ✓ | 3.2 | Plan implementation strategy | 1 wk | 13 Mar | 19 Mar | Process flow charts |
| 12 | | 3.3 | Report: Methodology | 1 wk | 20 Mar | 26 Mar | Chapter 3 of report completed |
| 13 | ✓ | **4** | **Vision-Based Perception** | **15 days** | **04 Feb** | **24 Feb** | Functional vision pipeline |
| 14 | ✓ | 4.1 | Develop computer vision pipeline | 1 wk | 04 Feb | 10 Feb | Preliminary vision code |
| 15 | ✓ | 4.2 | Integrate face authentication module | 1 wk | 11 Feb | 17 Feb | User Identification module |
| 16 | ✓ | 4.3 | Link user IDs to database profiles | 1 wk | 18 Feb | 24 Feb | User profiles linked |
| 17 | | **5** | **Language Understanding** | **20 days** | **25 Feb** | **24 Mar** | LLM integrated with CV |
| 18 | ✓ | 5.1 | Set-up & Integrate LLM module | 1 wk | 25 Feb | 03 Mar | LLM operational |
| 19 | ✓ | 5.2 | Integrate audio & vision inputs with LLM | 1 wk | 04 Mar | 10 Mar | Working pipeline to parse commands |
| 20 | ✓ | 5.3 | Define prompts & conversation logic | 1 wk | 04 Mar | 10 Mar | LLM prompt structure |
| 21 | | ~~5.4~~ | ~~Develop task representation~~ | ~~1 wk~~ | ~~11 Mar~~ | ~~17 Mar~~ | ~~Task representation ready~~ |
| 22 | ✓ | 5.5 | Test LLM command parsing, user queries | 1 wk | 18 Mar | 24 Mar | LLM test report |
| 23 | ✓ | 5.6 | Build conversational memory | 1 wk | 18 Mar | 24 Mar | Conversational memory built |
| 24 | ✓ | **6** | **Robotic Control** | **15 days** | **25 Mar** | **14 Apr** | Decision engine & skill library |
| 25 | ✓ | 6.1 | Define robot skill/sequence library in db | 1 wk | 01 Apr | 07 Apr | Skill library ready |
| 26 | ✓ | 6.2 | Map decision engine ( LLM to sequences) | 1 wk | 25 Mar | 31 Mar | Decision-making code |
| 27 | ✓ | 6.3 | Generate GUI control scripts | 1 wk | 08 Apr | 14 Apr | Graphic User Interface |
| 28 | | **7** | **Simulation in Omniverse** | **15 days** | **15 Apr** | **05 May** | Omniverse simulations |
| 29 | ✓ | ~~7.1~~ | ~~Set up virtual robot & environment~~ | ~~1 wk~~ | ~~15 Apr~~ | ~~21 Apr~~ | ~~Omniverse environment ready~~ |
| 30 | ✓ | 7.2 | Integrate Vision, LLM, Control Modules | 2 wks | 15 Apr | 28 Apr | Working integrated pipeline |
| 31 | ✓ | ~~7.3~~ | ~~Simulate task execution~~ | ~~1 wk~~ | ~~22 Apr~~ | ~~28 Apr~~ | ~~Task simulation logs~~ |
| 32 | | 7.4 | Debug & refine | 1 wk | 29 Apr | 05 May | Validation report |
| 33 | | 7.5 | Report: Methodology | 3 days | 01 May | 05 May | Chapter 3 of report |
| 34 | ✓ | **8** | **Testing & Validation** | **15 days** | **06 May** | **26 May** | Test results |
| 35 | ✓ | 8.1 | Test, analyse logs, & identify issues | 1 wk | 06 May | 12 May | Robot testing complete |
| 36 | ✓ | 8.2 | Evaluate performance | 2 wks | 13 May | 26 May | Performance validation report |
| 37 | ✓ | 8.3 | Report: Results & Discussion | 2 wks | 13 May | 26 May | Chapter 4 of report complete |
| 38 | | **9** | **Documentation & Reporting** | **5.5 days** | **28 May** | **04 Jun** | Thesis draft, presentations |
| 39 | | 9.1 | Prepare presentations/posters | 0.5 wks | 28 May | 31 May | Presentations, posters, videos |
| 40 | | 9.2 | Report: Conclusion & Future Work | 0.5 wks | 28 May | 30 May | Chapter 5 of report complete |
| 41 | | 9.3 | Final submission | 1 day | 04 Jun | 04 Jun | Final report, reflection document |
| 42 | | | | | | | |




*Project: Integrating Vision, Language Models, and Robotic Control for Personalized Task Execution in Virtual Environments.*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Task | Project Summary | Manual Task | Start-only | Deadline |
| Split | Inactive Task | Duration-only | Finish-only | Progress |
| Milestone | Inactive Milestone | Manual Summary Rollup | External Tasks | Manual Progress |
| Summary | Inactive Summary | Manual Summary | External Milestone |