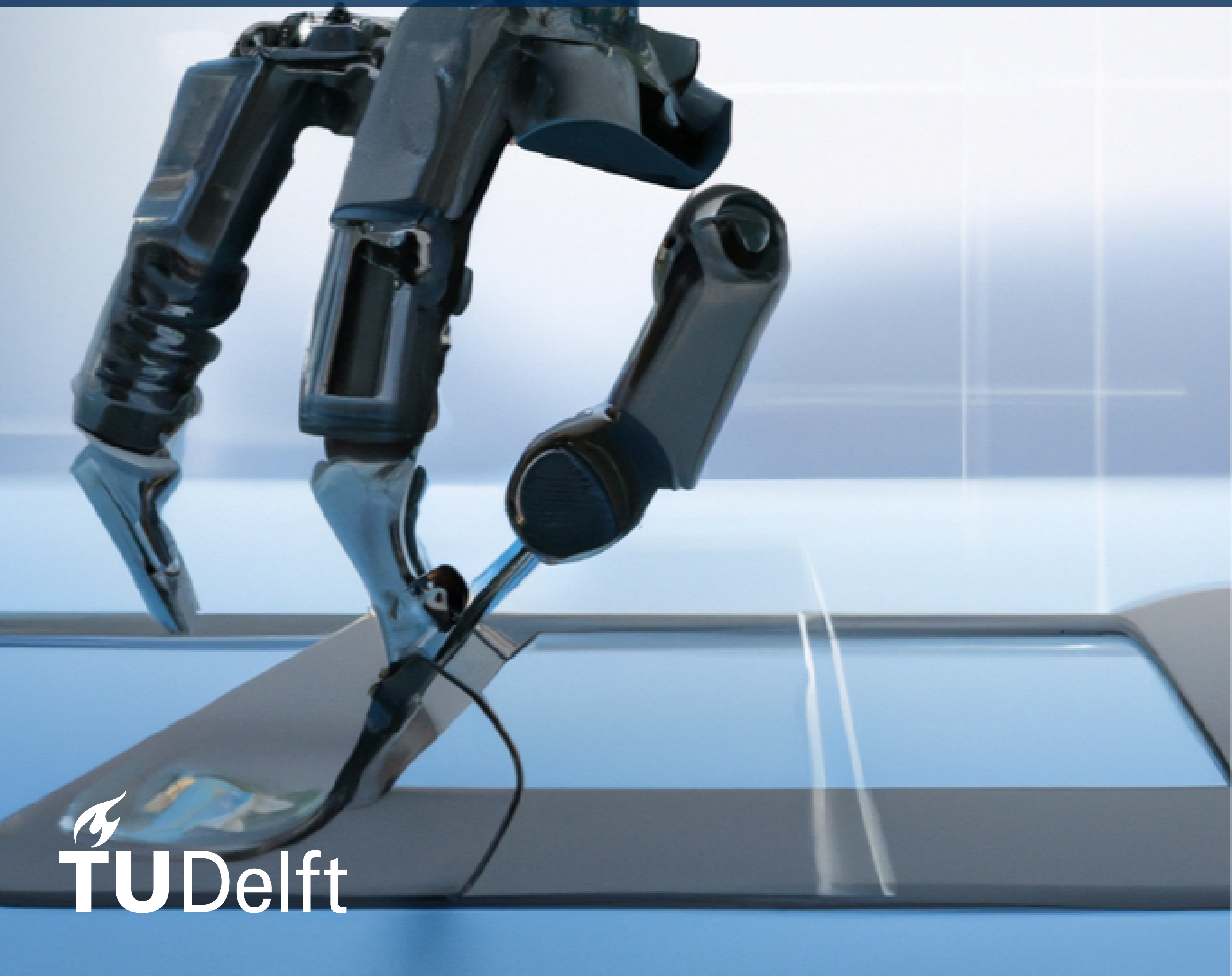
Perception and Control with Large Language Models in Robotic Manipulation

Developing and assessing an integrated Large Language Model System on environmental and task complexity

MSc Thesis

Leonoor Verbaan



TU Delft

Perception and Control with Large Language Models in Robotic Manipulation

Developing and assessing an integrated Large Language Model System on environmental and task complexity

by

Leonoor Verbaan

Student number:	5415721
Supervisor:	Dr. Ir. Yke Bauke Eisma
Daily supervisor:	Remco van Leeuwen
Project Duration:	September, 2023 - July, 2024
Faculty:	Faculty of ME, Delft
Department:	Cognitive Robotics (CoR)

Preface

This thesis was completed as part of my Masters program in Cognitive Robotics at the Faculty of Mechanical Engineering at Delft University of Technology. The research was conducted during a graduation internship at Alliander, a leading Dutch grid operator, providing a valuable opportunity to apply advanced technologies to real-world challenges. The focus of this project was on enhancing the control of a robotic arm, with the primary objective of evaluating the effectiveness and usability of such a system in various task-oriented environments and with different user inputs.

The primary objective of this research was to design and evaluate a control system for robotic manipulation tasks enhanced by a Large Language Model. The focus was on assessing the system's ability to interpret its environment and execute actions based on diverse user instructions. Before conducting the main experiments, considerable effort was dedicated to developing the Vision-Language-Action model, integrating GPT-4, and refining the robotic control and perception systems. The findings of the experiments, as well as the methodologies used to achieve them and to develop the system, are documented in the scientific paper that follows this preface.

I would like to express my gratitude to my supervisors, Remco van Leeuwen and Karlijn Overes from Alliander, and Yke Bauke Eisma from TU Delft, for their continuous support and invaluable insights throughout this research. Their guidance and constructive feedback were crucial in shaping the direction and outcomes of this work. I am also thankful to PhD student Renchi Zhang, as well as my friends, whose encouragement and support were a constant source of motivation during my time at TU Delft.

Leonoor Verbaan
Delft, August 2024

LLM-Enhanced Robotic Affordance and Control System in robotic manipulation

Assessing task execution and user interaction with generated personas

Leonoor Verbaan[1], **Yke Bauke Eisma**[2], **and Remco van Leeuwen**[2]

[1]**TU Delft, Cognitive Robotics, Delft, 2628 CD, Netherlands**
[2]**Alliander, Research Center for Digital Technologies, Arnhem, 6812 AH, Netherlands**

Abstract Large Language Models (LLMs) possess significant semantic knowledge about the world, making them valuable for high-level control for robots through Vision-Language-Action (VLA) models. These models integrate an LLM to deduce semantic knowledge from a robot's vision and natural language inputs, facilitating real-world actions. Despite their potential, VLA models are a relatively new research area, with applications mostly limited to simulations or household tasks and insufficient validation in broader contexts. This study aims to develop a **LLM-Enhanced Robotic Affordance and Control System (LERACS)** for robotic manipulation in applied cases such as the management and maintenance of electrical grid infrastructure. LERACS is designed to visually ground manipulable objects and decompose tasks based on user instructions within a human-robot interaction chat interface, using ChatGPT. A system validation and an AI user experiment were conducted to evaluate its effectiveness in interpreting and performing actions based on pre-made and synthetically generated user instructions. These assessed LERACS' performance across various settings and instructions. Results indicate high success rates in environmental interpretation and task execution, with robust labeling accuracy, especially in complex settings. Feedback from the AI user experiment highlighted LERACS' adaptability, identified areas for improvement, and demonstrated its practical utility across diverse settings and task complexities. The open sourced code and implementation details can be found at LERACS GitHub.

Keywords

Vision-Language-Action Models, Manipulation Affordance, Applied Robotics, AI persona, Large-Language-Model Validation, Control and Vision Prompting, LERACS, Human-Robot Interaction

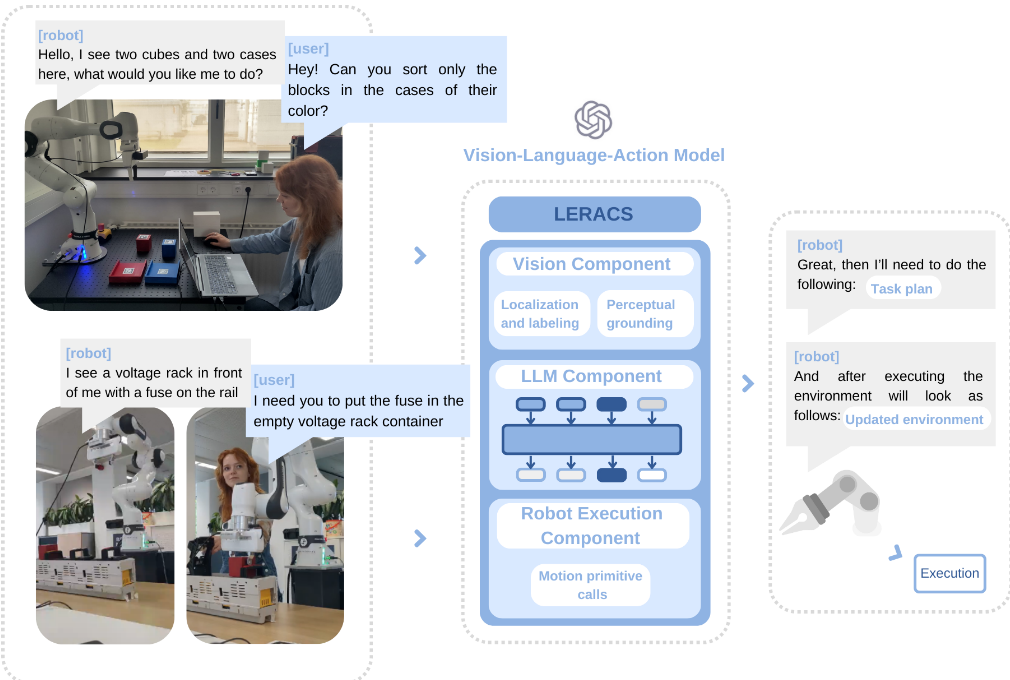


Figure 1. LLMs are trained on a large amount of data which make them valuable for high-level control such as generating task plans from instructions and images and interpreting the outcomes after executing these task plans. LERACS is a VLA model that visually grounds manipulable objects, decomposes tasks based on user instructions, and understands the implications of these tasks after executing.

1 Introduction

Robotic manipulators are widely used in the automation industry for efficient object manipulation tasks [1]. These advancements have led to innovative applications across various fields, including electrical grid operations, which face numerous challenges due to the growing need for sustainable energy worldwide [2, 3]. The complexity and inherent risks of power systems, such as electrocution, falls, and explosions, make their inspection and maintenance particularly challenging and hazardous. To address these issues, Alliander, a Dutch grid operator, is actively exploring the use of robotic arms in electrical grid tasks [4]. By reducing the need for direct human involvement in dangerous situations, this approach aims to mitigate the risks faced by mechanics and improve both safety and operational efficiency [5]. Therefore, introducing robotic arms could allow mechanics to interact with the electrical grid from a safe distance, enhancing their safety and efficiency [6, 7].

Given the dynamic and hazardous nature of power systems, robotic models must be accurate, robust, generalizable, and adaptable. Recent advances in natural language processing have yielded Large Language Models (LLMs) with significantly improved abilities to understand and generate language. LLMs such as ChatGPT [8], BERT [9], FLAN [10], LAMDA [11], T5 [12], and PaLM [13] have resulted in systems capable of generating complex text, answering questions, and engaging in dialogues on a wide range of topics. As a result of learning vast amounts of data, some LLMs can be fine-tuned with a small set of sample data as instructions (i.e., few-shot learning [8]). LLMs excel in solving diverse tasks compared to prior models confined to specific tasks and datasets. An extended sort of LLM called Visually-conditioned language models (VLMs), which are trained on large-scale data to generate natural language from input images and prompts, have been adopted for applications such as visual question answering, semantic understanding, and object localization [14]. A newer application of these VLMs in robotics, where they can be used to generate executable robot programs for task planning. These models are the vision-language-action models (VLAs) which combine the visual input and natural language instructions for robot task planning [15, 16, 17, 18, 19, 20, 21, 22, 23]. These models integrate advances in both computer vision and natural language processing to create powerful, human-understandable multimodal systems. Incorporating action possibilities that an object or environment offers, known as affordances, allows dynamic deduction of possible actions within a given scene. Affordances refer to the potential actions that an object or environment inherently provides to an agent, such as a robot. In the context of robotic manipulation, affordances describe how objects can be interacted with or manipulated, such as an object affording grasping or pushing. Additionally, scene affordances refer to the overall action possibilities provided by the arrangement and context of objects within an environment, such as a table being a surface with objects on them that can be interacted with. Understanding these affordances enables robots to make informed decisions during task planning and execution, thereby improving efficiency [24, 23]. This concept is visually illustrated in *Figure 2* which shows how VLAs leverage affordances in real-world scenarios.

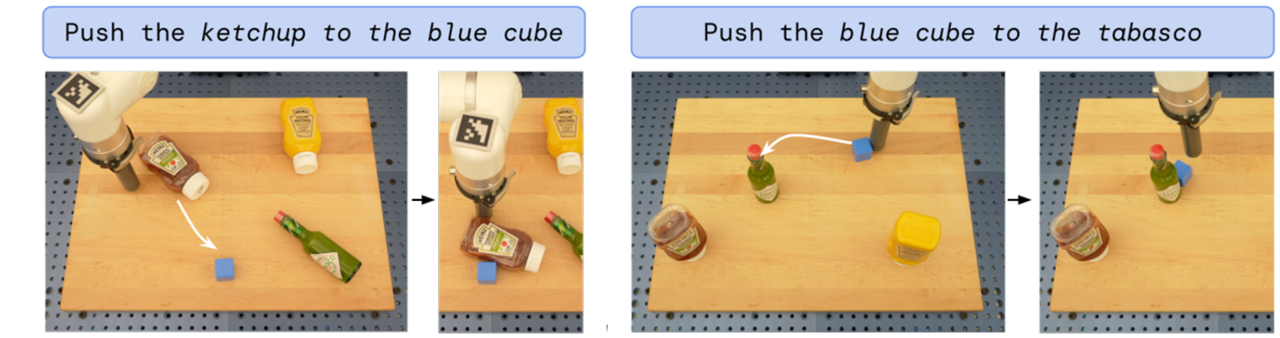


Figure 2. Real-world experiments of VLA (RT-2) in the Language Table environment from Google Deepmind [23].

Despite these advancements, the integration of VLAs into applied robotics, remains underexplored [25]. Task planning from natural language instructions is a newer research topic in robotics, however most of them lack the functionality of human-in-the-loop [26, 19], and these studies often rely on specific datasets [15, 16, 17, 27], necessitating data recollection and model retraining when transferring or extending these to other robotic applications or settings. Existing VLA research often focuses on single robot or simulated setups, lacking generality and efficient fine-tuning for new applications [28]. Most studies have tested simple household tasks with task-specific models, not being able to validate their systems' performance or robustness across diverse user groups. Moreover, human language is highly diverse, with the same intention often being expressed in various ways depending on the user and context. This linguistic diversity presents significant challenges for language grounding in robotics, where a system must accurately interpret and execute commands despite variations in phrasing or syntax. While recent advances in LLMs have improved natural language understanding, the application of these models in robotics, particularly in dynamically interpreting varied user instructions, remains underexplored and poses a critical challenge for the development of adaptable and robust robotic systems [29]. This raises the question of whether these models can be used by users and robots to perform complex tasks in the real world.

The growing interest of companies like Alliander in implementing robotic arms to improve employee safety in both static and dynamic, hazardous environments underscores the need for improved human-robot interaction using advanced robot models. Therefore this research aims to measure the effectiveness of the integration of LLMs, specifically GPT-4, as a VLA method for robotic manipulation tasks. The primary objective is to assess how well certain actions are interpreted and performed with an VLA as a task planning method for robot control, evaluating both the effectiveness of execution and interpretation through diverse user input instructions. This study will highlight the capabilities and limitations of this task planning method. Accordingly, this study seeks to address the following research question:

How can an applicable, adaptable, and generalizable vision and control system based on Large Language Models be developed and tested for robotic arm manipulation tasks, considering the impact of setting and task instruction complexity on system performance, and robustness against diverse user instructions?

To address these challenges, this research proposes, designs, and tests the **LERACS:** *LLM-Enhanced Robotic Affordance and Control System in robotics manipulation* (*Figure 1*). The remainder of this work is structured as follows: First, in the *Methods 2*, the implementation of LERACS with a robotic arm is introduced. Then the system validation (implemented in a robotic arm) and AI user experiments are presented. Subsequently, the results of these experiments are displayed in *Results 3* and discussed in *Discussion 4*. Finally, the concluding remarks addressing the research question and future research are presented in the *Conclusion 5*.

2 Methods

2.1 Implementing LERACS in a Robotic System

LERACS is a system designed to facilitate vision-based task planning and control for robotic manipulation using OpenAI's ChatGPT API [30] in a few-shot setting. The system integrates three primary components: a **Vision Component** for marker-based localization and index labeling, a **LLM Component** for vision interpretation and task generation [31, 32, 33], and a **Robot Execution Component** for executing tasks and updating the environment. These components work together to generate and execute multi-step task plans, making LERACS a tool for manipulation affordance in robotic systems. *Figure 3* provides an overview of the LERACS workflow, illustrating the interactions between the Vision Component, LLM Component, Robot Execution Component, and the System Interface. The flow diagram outlines the sequential processing of tasks, from initial perception and task sequence generation to execution and environment updating. The following sections will explain the workflow using the example instruction: "***Can you sort only the blocks according to their color?***"

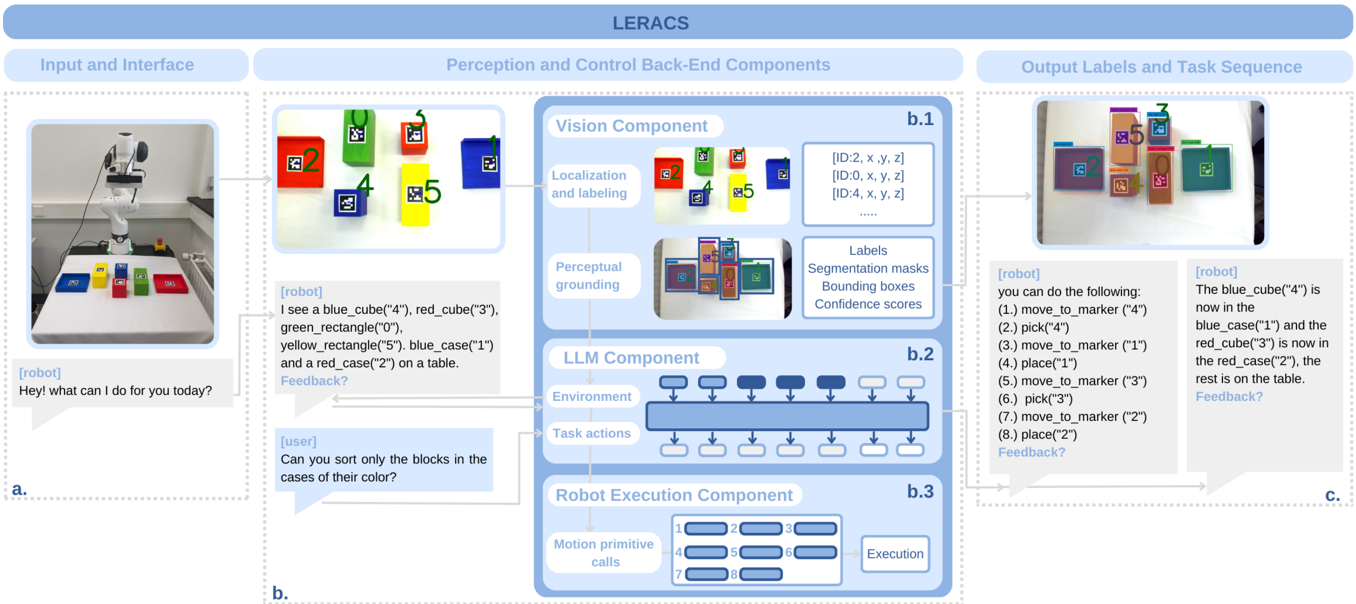


Figure 3. LERACS-overview: This system flow diagram illustrates the whole system including the input and interface, the robot's perception and control back-end with its three primary components, and examples of the output. In (a.) the robot communicates to the user via the interface, initializes the robot arm and starts interpreting the scene with ChatGPT. In the back-end (b.) the interpreted scene is generated in text and the user provides task instructions. From this image and textual input the tasks are decomposed based on the instructions, a prediction of the scene is made after execution of these tasks (**LLM Component** (b.2)), and the manipulable objects in the scene are identified and labeled (**Vision Component** (b.1)). The task sequence is then executed with the available motion primitives via service calls (**Robot Execution Component** (b.3)). Examples of the task sequence prepared for execution, along with the resulting changes to the scene after execution, are shown in (c.). The robot asks the user for feedback after every generated sample.

2.1.1 LERACS System Components

Vision Component: Marker-Based Localization, Index Labeling, and affordance grounding. The Vision Component displayed in *Figure 3b.1* is responsible for processing visual data to identify and label objects within the environment. It utilizes ArUco markers [34] to detect objects, translating their positions into real-world coordinates and annotating the marker index on the image stream of the camera. The marker IDs and their positions are then used for the subsequent stages, as it contains data (marker ID: x, y, z) required for task generation and execution. The labeled information from the Vision Component functions as image input for the LLM Component, where it is used to interpret the environment as a textual representation.

Additionally the Vision Component extracts the objects with their marker IDs from textual representation as labels to a pre-trained detection and segmentation model (GroundedSAM), which functions as perceptual grounding of the manipulable objects in the scene [35]. The segmentation masks, bounding boxes, and confidence scores are extracted and visualized alongside the labels in an image snapshot, providing the user with feedback on what the robot has perceived. This process also serves as an automatic labeling feature, contributing to the creation of datasets that help generalize the system to new settings.

LLM Component: Environment Interpretation, Task Generation and Environment Update. The LLM Component, which uses the ChatGPT-4o model, displayed in *Figure 3b.2* interprets the image snapshot from the camera video stream provided by the Vision Component into a python dictionary that can be saved as a JSON **(environment: assets, asset states, object, and object states)**. An example of this process is illustrated in *Figure 4*.

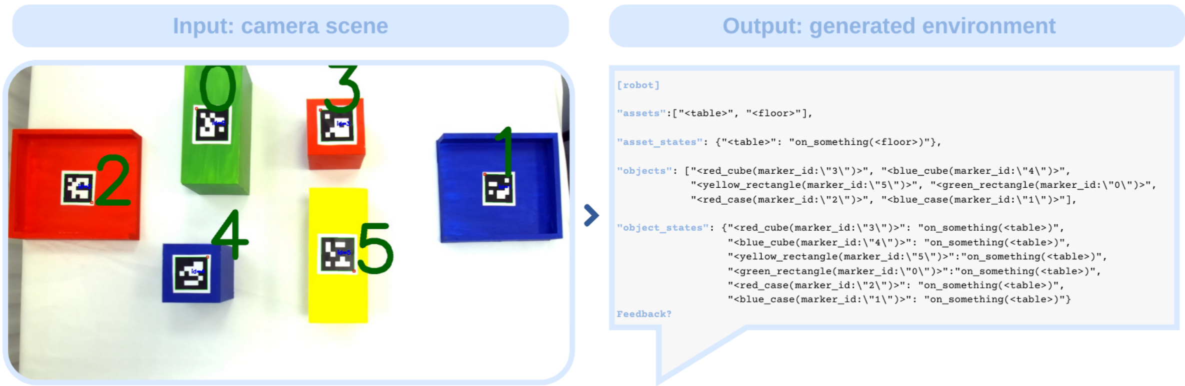


Figure 4. Visualization of LERACS interpreting the environment. The system takes a camera snapshot as input and generates a textual representation of the environment as output.

Additionally, the LLM Component takes textual user instructions and this previously interpreted scene and generates not only the sequence of robot actions **(task sequence)**, but also explanations of each actions step **(step instructions)** and supplementary information on the updated environment after executing the actions **(environment before and environment after)** again in a python dictionary that can be saved as a JSON. An example of this process is illustrated in *Figure 5*. By using a few-shot setting with ChatGPT, the LLM Component mitigates the impact of token limits while ensuring that the generated tasks are executable by the robot. These additional pieces of information help the user debug whether ChatGPT correctly processes the input information, providing a mechanism to verify and adjust the generated tasks as needed. Then, the sequence of robot actions are passed on to the Robot Execution Component, which carries out the instructions.

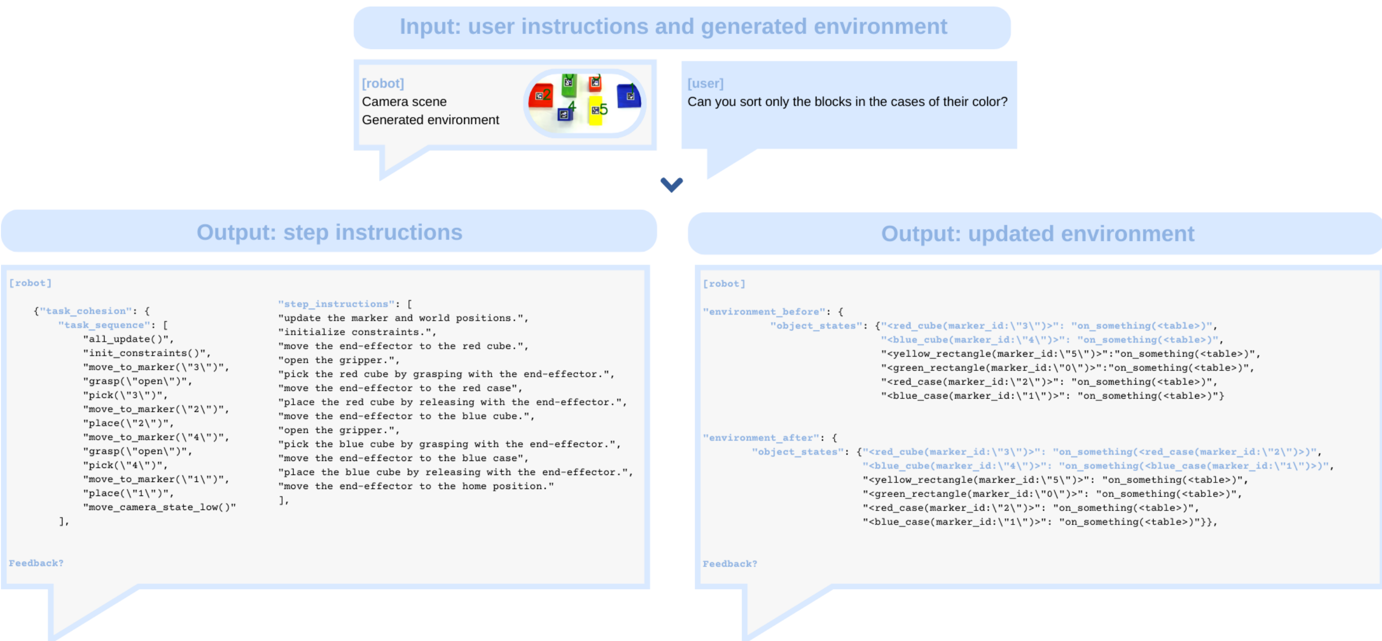


Figure 5. Visualization of LERACS decomposing the tasks and predicting the environment after executing the tasks. The system takes the generated environment with the snapshot and user instructions as input and generates a sequence for the robot to execute.

User feedback plays a crucial role in ensuring safe and robust operation, as ChatGPT does not always generate complete action sequences. To address this, the user can take advantage of ChatGPT's ability to adjust its output based on natural-language feedback. This feedback can be provided for both the image snapshot interpretation and the generated robot action sequence, as well as the updated environment. As shown in *Figure 6a*, when a user requested to add or remove a task from the output sequence, ChatGPT successfully modified the output according to the semantic content of the feedback, demonstrating its capability to make necessary adjustments.

In this process, ChatGPT handles a series of six distinct prompts: **(1) role of ChatGPT**, **(2) definition of robot actions**, **(3) representation of the environments**, **(4) the format of the output produced by ChatGPT**, **(5) examples of input and output**, and **(6) specific instructions or feedback from the user**. Specifics about the prompting methods for task decomposition and control can be found in *Appendix 6 at section 6.1.3, and 6.1.1*. Moreover, these 6 prompts (5 set prompts along with user instructions/feedback) are inputted as a six-turn conversation, a technique known as incremental prompting. This approach helps the model maintain context more effectively, resulting in more robust performance compared to bundling all prompts into a single input [31]. The prompts are structured as a conversation as shown in *Figure 6b*.

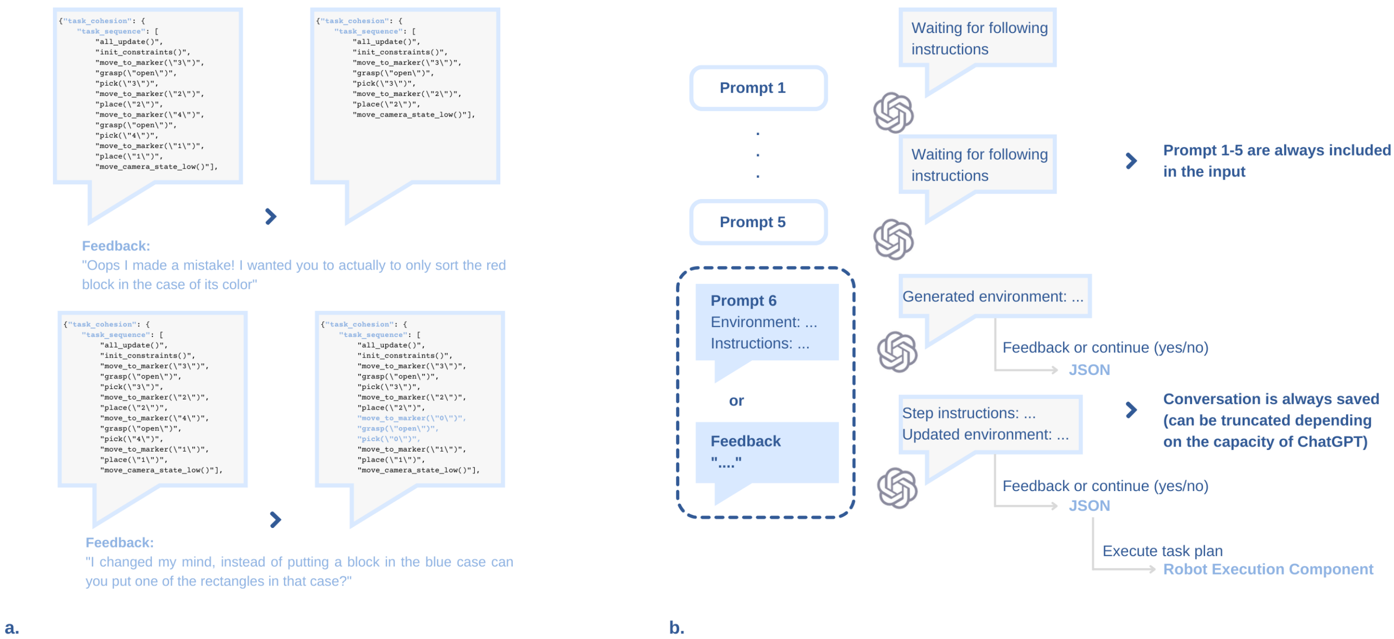


Figure 6. (a.) Two examples of the feedback function in LERACS and how it can alter the task sequence. Feedback can also be given on the updated environment. (b.) The prompt order and structure of the conversation for task planning in LERACS with the ChatGPT API.

Robot Execution Component: Initialization and Task Execution The Robot Execution Component displayed in *Figure 3b.3* is tasked with carrying out the task sequence generated by the LLM Component. This component consists of workspace constraints, robot initialization and updating functions, and **motion primitives**, allowing for **immediate function calling** [36]. The motion primitives in this study are presented in *Figure 7*. If no feedback is provided or the user accepts the generated output from the LLM Component, the robot action sequence is executed by the Robot Execution Component.

Figure 7. These are the basic motion primitives for LERACS and their in and output variables.

System Interface as an Add-On The System Interface acts as an additional layer, allowing users to interact with the LERACS back-end components. It integrates the Vision, LLM, and Robot Execution Components, facilitating efficient interaction between the user and the robot. *Figure 8* provides a visualization of the user interface, illustrating the interactions between the robot and user in the chat area including instructions and feedback, the visual feedback of the robot to the user, and the other widgets (*Figure 8a*). It consists of two main areas: the chat area for textual communication and the robot vision area for visual feedback. Through this interface, users can initiate tasks, provide feedback, and monitor the robot's status in real-time (*Figure 8b*). The interface includes additional functionalities through various buttons. These buttons allow users to start the Franka robot system(making a snapshot and initializing the back end), reinitialize the robot control node, refresh the user interface, run the detection to display manipulable objects (*Figure 8c*), and change the camera index. The user interface, developed using Tkinter [37], serves as the connection to the back-end, integrating all components. A full conversation example via the chat interface can be found in *Appendix 7.1*.

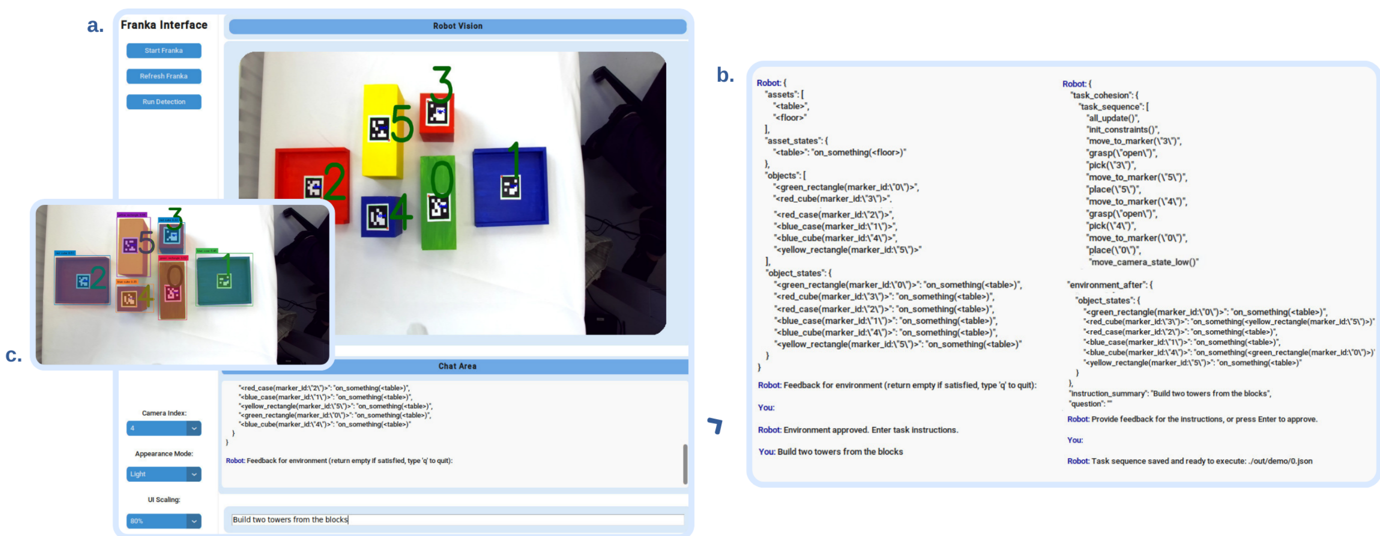


Figure 8. LERACS with the interface. (a.) displays the general interface that the user interacts with consisting of the visual feedback from the camera, the chat area and the widgets. (b.) is a snippet of a potential chat for the task instruction "***Build two towers from the blocks***". And (c.) illustrates the results from the run detection widget that shows the manipulable objects in the scene including their bounding boxes, labels and segmentation masks.

2.1.2 Apparatus

LERACS is implemented and tested on a Franka Emika Research 3 manipulator [38] with a ZED 2 stereo camera [39] and an HP ZBook Power G7 Mobile Workstation running Ubuntu Linux 20.04 with real-time kernel 5.9, featuring an Intel Core i7-10750H processor, 32 GB DDR4 memory, and a 1 TB NVMe SSD [40]. The fundamental software component to control the Franka Emika robot arm is the Robotic Operating Software (ROS), version Noetic [41]. Accompanied by the MoveIt package, libfranka [42] and franka_ros [43].

2.2 Experimental Evaluation

The components of LERACS have been validated and tested through two experiments, focusing on setting complexity (dynamic prompt 6: image input) and task instruction complexity (dynamic prompt 6: task instructions). In addition to these experiments, which include system validation and AI persona instruction experiment, a case application demo was also implemented. The following sections will define these metrics and variables used in these experiments and explain the testing methods applied.

2.2.1 Complexity Definition and Experimental setup

Complexity Definition. The influence of two factors, i.e., the complexity of the scene and of the task instructions, on the performance of LERACS is investigated. The complexity of the setting is characterized by different objects, forms, object relations, and context within the scene. The performance of the system is impacted by the number and type of objects present. Previous studies have shown that as the number of objects in a scene increases, system accuracy decreases across all approaches [44]. Therefore to test this, the settings were varied by colors, shapes, and landmarks.

The type of task complexity is considered based on the following four challenges: whether the task has been seen or not seen in the example prompts as displayed in *Appendix 6.1.3* ***Prompt Example*** (active-prompting (AP) [45]), the sequence of actions required (chain-of-thought (CoT)) [46] and chain-of-event prompting (CoE) [47]), the variety in elements present (multi-modal prompting [48]), and the spatial understanding required (multi-modal and instructed prompting [48]). These aspects are displayed in *Table 1*. The evaluation of task complexity mainly test these prompting techniques, which include the five preset prompts along with dynamic user instructions and camera input prompts, to assess system robustness and limitations. For seen and unseen tasks the generalizability and adaptability of the system is tested because reliance on training data can introduce biases and limitations in task execution, especially for unseen tasks [31] [49]. Then for sequence of actions the performance of the system itself is tested because the primary bottleneck for most LLM systems, including ChatGPT, lies in handling the range and capabilities of underlying skills, particularly in decomposing and executing complex sequences of actions (e.g. the action of sorting consists of a large chain of actions) [31] [17] [49]. Testing the variety of object elements assesses the difficulty type by involving the logical connection of subgoals and the need for implicit knowledge. For instance, the presence of multiple colors and shapes in the environment enhances task difficulty by necessitating more sophisticated environmental interpretation and decision-making processes (e.g. 2 of the same blocks but they have a different color which is needed as element to tell them apart) [44]. Spatial understanding significantly increases task difficulty because it requires the system to not only recognize objects and their properties but also understand their spatial relationships and dynamics within the environment. This involves mapping spatial relations specified in natural language commands to subsymbolic object goal locations in the world (e.g. The block on the right of the Tabasco bottle needs to be pushed to the Ketchup bottle) [50] [51] [17].

Independent Variables. Consequently, the independent variables in this experiment are the complexity level of the settings (simple and complex, *Figure 9*) and the complexity types of the task instructions (type, *Table 1*) as detailed in the complexity definition. The complexity of the setting, is categorized into two different categories:

- **Simple.** This setting that the robot arm with LERACS needs to operate in represents the easiest to solve according to the literature [44]. The simple setting includes 2 different colors and 2 landmarks.

- **Complex.** This setting that the robot arm with LERACS needs to operate in represents more complex to solve according to the literature [44]. The complex setting includes 4 different colors, 2 different shapes, and 2 landmarks.

The complexity type of the task instructions in this experiment are categorized into 5 types as summarized and displayed in *Figure 1*:

- **Type (a.)** ***[Instruction: Give me the [object]].*** According to the custom complexity definition this type is assigned to test single task generation connected to basic reasoning of object shapes from the instruction. Type (a.) for the instruction description consists of an example of the instruction in the prompts *Appendix* 6.2 and a variety in shape of objects, with few actions in a sequence, and no spatial understanding elements.

- **Type (b.)** ***[Instruction: Sort the [object] in the [landmark]].*** According to the custom complexity definition this type is assigned to test task generation consisting of some tasks connected to mild reasoning of object colors from the instruction. Type (b.) for the instruction description consists of an example of the instruction in the prompts *Appendix* 6.2 and a variety in colors of objects, with some actions in a sequence, and no spatial understanding elements.

- **Type (c.)** ***[Instruction: Place the [object] on top of the [object]].*** According to the custom complexity definition this type is assigned to test task generation consisting of some tasks from an unseen prompt. Therefore, for this type the system needs to use prior knowledge. Type (c.) for the instruction description consists of a variety in shapes of objects, with no example of the instruction in the prompts, some actions in a sequence, and no spatial understanding elements.

- **Type (d.)** ***[Instruction: Push the [object] to the position of the [object]].*** According to the custom complexity definition this type is assigned to combine shape and color with spatial reasoning from the instruction. Type (d.) for the instruction description consists of a variety in shapes and color of objects, spatial understanding elements, with no example of the instruction in the prompts, and some actions in a sequence.

- **Type (e.)** ***[Instruction: Sort the [object] in the [landmark] by color]***. According to the custom complexity definition this type is assigned to test task generation of longer action sequences deducted from spatial and abstract reasoning. Type (e.) for the instruction description consists of a variety in shapes and color of objects, many actions in a sequence, and spatial understanding elements, with no example of the instruction in the prompts.

In addition to defining the types of instruction complexity, there is also a focus on the specificity of task instructions. Task specificity is divided into two main categories: **general instructions** and **scene-specific instructions**. This indicates that for example type (a.) has a general and scene-specific version of the instruction, to connect the setting and instruction complexity with this element. General instructions provide minimal details and perform higher-level planning which can be used in both simple and complex settings (baseline). Scene-specific instructions, on the other hand, offer detailed guidance tailored to the specific scenario and are therefore different in both simple and complex settings. 5 general instructions (e.g. Give me one of the blocks) and 5 scene-specific (e.g. give me the red block) are chosen per setting.

Table 1. Complexity types for the task instructions consisting of the elements discussed in the complexity definition: seen in example prompts (yes/no), sequence of actions (few/some/many), variety in object elements (shape/color/shape and color), spatial understanding (yes/no).

Type	Seen in Example Prompts	Sequence of Actions	Variety in Object Elements	Spatial Under-standing	Description
(a.)	Yes	Few	Shape	No	A single task to identify a shape with basic reasoning.
(b.)	Yes	Some	Color	No	Some actions to identify elements of color with mild reasoning.
(c.)	No	Some	Shape	No	Unseen tasks testing generalization from prior knowledge.
(d.)	No	Some	Shape and Color	Yes	Combining shape and color with spatial reasoning.
(e.)	No	Many	Shape and Color	Yes	Spatial and abstract reasoning with longer action sequences.

Experimental Setup There are a total of 20 scenarios that need to be performed in the following experiments. These scenarios are displayed in *Table 2, and 3*): 10 scenarios with general instructions in both settings, and 10 scenarios with scene-specific instructions in both settings (scenario 1 to 5 in lightblue/white are general and 6 to 10 in darkerblue/lightgrey are specific in *Table 2* and *Table 3*) [31] [17]. The scenarios consist of a textual instruction and a goal instruction which are explained later at the experimental procedures for the experiments. The complexity of the task instruction (task type from *Table 1*) is also assigned per scenario.

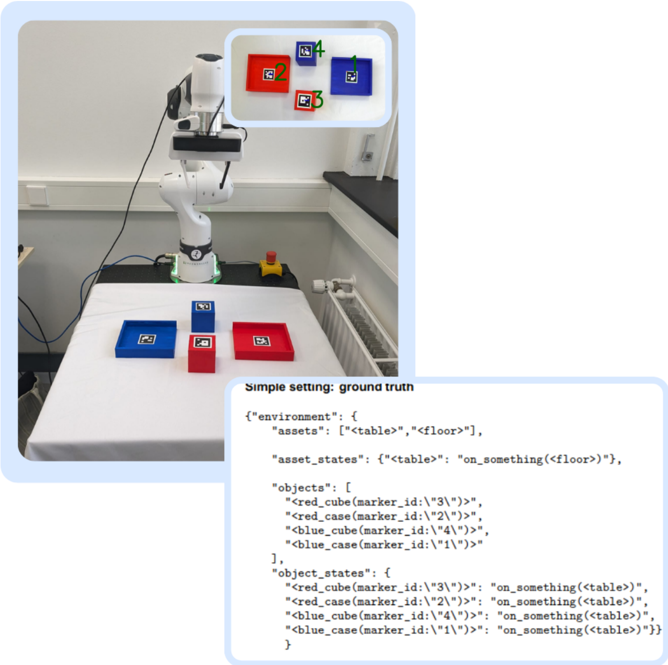
Table 2. Scenarios for the simple setting. **Textual Instructions** relate to the system validation experiment, **Goal Instructions** to the AI persona instruction experiment, and **Task Types** connect to each scenario.

Scenario	Textual Instruction	Goal Instruction	Task Type
Scenario 1	Give me one of the blocks.	The robot needs to hand you one of the blocks.	Type (a.)
Scenario 2	Sort one of the blocks in one of the cases.	The block needs to be standing in one of the cases.	Type (b.)
Scenario 3	Place one of the blocks on top of the other blocks.	The two blocks need to be forming a tower.	Type (c.)
Scenario 4	Push one of the blocks to the position of the other block.	One of the blocks needs to be standing in place of the other block.	Type (d.)
Scenario 5	Sort the blocks in the cases by color.	Only the blocks with the colors of the cases need to be standing in the corresponding cases	Type (e.)
Scenario 6	Give me the red block.	The robot needs to hand you the red block.	Type (a.)
Scenario 7	Sort the blue block in the red case.	The blue block needs to be standing in the case of its own color.	Type (b.)
Scenario 8	Place the blue block on top of the red block.	The two blocks need to be forming a tower with the blue block as the top.	Type (c.)
Scenario 9	Push the red block to the position of the blue block.	The red block needs to be standing in place of the blue block.	Type (d.)
Scenario 10	Sort the red block in the red case and the blue block in the blue case.	The red block need to be standing in the red case and the blue block in the blue case.	Type (e.)

Table 3. Scenarios for the complex setting. **Textual Instructions** relate to the system validation experiment, **Goal Instructions** to the AI persona instruction experiment, and **Task Types** connect to each scenario.

Scenario	Textual instruction	Goal instruction	Task Type
Scenario 1	Give me one of the blocks.	The robot needs to hand you one of the blocks.	Type (a.)
Scenario 2	Sort one of the blocks in one of the cases.	The block needs to be standing in one of the cases.	Type (b.)
Scenario 3	Place one of the blocks on top of the other blocks.	The two blocks need to be forming a tower.	Type (c.)
Scenario 4	Push one of the blocks to the position of the other block.	One of the blocks needs to be standing in place of the other block.	Type (d.)
Scenario 5	Sort the blocks in the cases by color.	Only the blocks with the colors of the cases need to be standing in the corresponding cases	Type (e.)
Scenario 6	Give me one of the rectangles.	The robot needs to hand you one of the rectangles.	Type (a.)
Scenario 7	Sort the rectangles in the blue case.	The rectangles need to be standing in the blue case.	Type (b.)
Scenario 8	Place the red block on top of the yellow rectangle.	The yellow rectangle and red block need to be forming a tower with the red block as the top.	Type (c.)
Scenario 9	Push the blue block to the position of the yellow rectangle.	The blue block needs to be standing in place of the yellow rectangle.	Type (d.)
Scenario 10	Sort the blocks in the cases by color and the rectangles in each a separate case.	The blocks with the colors of the cases need to be standing in the corresponding cases and each rectangle has to be standing in a separate case.	Type (e.)

The experiments are conducted in both **simple** and **complex** settings, where a robot arm faces a table arranged with various configurations of objects. These settings and ground truths are displayed in *Figure 9*. The objects in these settings vary in color, shapes, and landmarks. In the simple setting, there are 2 different colors and 2 landmarks. In contrast, the complex setting includes 4 different colors, 2 different shapes, and 2 landmarks.



(a) Simple setting

(b) Complex setting

Figure 9. The configuration of (a.) simple and (b.) complex settings with their ground truth. Both the user view (front view) as the robot view (top view) is given.

2.2.2 Experimental tasks and procedures

System validation Experiment: Dependent Variables. A system validation experiment is performed to measure the performance of the LERACS. The key aspects being tested are the environmental understanding, task decomposition ability, and task execution success rate in different scenarios within both simple and complex settings. To evaluate these aspects, the following *metrics* were extracted: **(1) Task/Sequence Execution Success Rate (TSE sr), (2) Task/Sequence Generation Success Rate (TSG sr), (3) Environment Interpretation Success Rate (EI sr), (4) Labeling Success (Precision, Recall and F1 score), and (5) the Number of Tries** required for environment interpretation and task/sequence completion. These metrics assess a LLM's capabilities in text classification, semantic understanding (including environment interpretation success, labeling success, and number of tries), and commonsense planning tasks (such as task/sequence generation and execution, and number of tries) [52] [53] [54].

Planning and Control Performance Metrics

- **TSE sr.** This metric is defined based on the robotic arm's ability to successfully execute all tasks generated by LERACS (0 unsuccessful and 1 successful execution). The success rate is subsequently determined as the percentage of trials in which the robotic arm successfully completed the tasks of a given scenario (ex).

$$\textbf{\textit{TSE sr}} = \frac{\textit{Successful trials (ex)}}{\textit{Total number of trials (ex)}}$$

- **EI sr.** This metric is defined based on LERACS's ability to successfully generate the ontological description of the environment from a video frame. The success rate is subsequently determined as the percentage of trials in which the robotic arm successfully generated this ontological description per given scenario (env). Since there is a feedback system, a successful trial is calculated with a penalty, meaning that the user has 4 chances to generate the right ontological description (penalty -0.25, 0 unsuccessful and 1 successful).

$$\textbf{\textit{EI sr}} = \frac{\textit{Successful trials}\ (\textit{env}) - 0.25 \times \textit{Feedback trials}\ (\textit{env})}{\textit{Total number of trials}\ (\textit{env})}$$

- **TSG sr.** This metric is defined based on LERACS's ability to successfully generate a decomposed task sequence from the user instructions. The success rate is subsequently determined as the percentage of trials in which the robotic arm successfully decomposes the tasks from the user instructions per given scenario (task). Due to the feedback mechanism, the success of a trial incorporates a penalty system. Users have up to four attempts to produce the correct task sequence (penalty -0.25, 0 unsuccessful and 1 successful).

$$\textbf{\textit{TSG sr}} = \frac{\textit{Successful trials}\ (\textit{task}) - 0.25 \times \textit{Feedback trials}\ (\textit{task})}{\textit{Total number of trials}\ (\textit{task})}$$

- **Number of Tries.** This metric is related to the amount of feedback trials for both environment interpretation and task/sequence generation. The total amount of tries possible is eight.

$$\textbf{\textit{Number of tries}} = \textit{Feedback trials}\ (\textit{task}) + \textit{Feedback trials}\ (\textit{env})$$

Vision Performance Metrics

- **Precision.** This metric measures the accuracy of the LERACS's in correctly identifying and labeling manipulable objects in the environment and to minimize false positives. Precision is defined as the ratio of correctly identified objects to the total number of objects identified by the system.

$$\textbf{\textit{Precision}} = \frac{\textit{True Positives}}{\textit{True Positives} + \textit{False Positives}}$$

- **Recall.** This metric evaluates LERACS's ability to identify all relevant manipulable objects in the environment and to minimize the chance of missing false negatives. Recall is defined as the ratio of correctly identified objects to the total number of actual objects present.

$$\textbf{\textit{Recall}} = \frac{\textit{True Positives}}{\textit{True Positives} + \textit{False Negatives}}$$

- **F1 Score.** This metric provides a harmonic mean of precision and recall, offering a single measure of LERACS's performance in object recognition. The F1 score evaluates how well the system uses the generated labels to automatically identify manipulable objects in the scene.

$$\textbf{\textit{F1 Score}} = 2 \times \frac{\textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

System Validation Experiment: Experimental Procedure. The system is tasked with performing certain **textual instructions** for all *scenarios* in both simple and complex settings (*Table 2, 3 and Figure 9*). The experiment follows the aforementioned **Experimental Setup**. The experiment proceeds as follows: for each setting, 10 scenarios will be performed with each 5 trials. The objects and landmarks are placed in several configurations over the trials (*Appendix 7, at 7.2*). After each scenario, the relevant metrics are extracted.

AI Persona Instruction Experiment: Dependent variables. An AI persona validation experiment is conducted as a simulated user study to predict human responses to the system. The primary aspects tested are the robustness to different types of user instructions and the interpretability of the system's output with the same independent variables as the system validation experiment. Previous research has shown that LLMs can generate diverse and contextually relevant personas based on various prompts, indicating their potential to adapt to different user instructions effectively which can be useful for developing this system that can be deployed in real-world situations [55] [56] [57] [58]. To evaluate these aspects, the following *metrics* are extracted: **(1) number of tries for environment and task/sequence success, (2) conversion rate of task generation, (3) conversion rate of environment generation, and (4) user understanding measured by the satisfaction and usefulness scale** (van der Laan acceptance scale [59]). These metrics assess a LLM's capabilities in commonsense planning, logical reasoning and semantic understanding tasks (conversion rate of environment and task generation, and number of tries for environment and task/sequence success), and social knowledge understanding (user understanding) using an alternative version of the human evaluation method [52].

AI User Factor Metrics

- **Environment Generation Conversion Rate.** This metric measures the percentage of trials in which users successfully generate an ontological description of the environment using the LERACS. It evaluates the efficiency with which users convert their inputs into usable environmental data.

- **Task Generation Conversion Rate.** This metric assesses the percentage of trials in which LERACS successfully generates a decomposed task sequence from the user instructions. It indicates the efficiency with which users can translate their task instructions into actionable sequences.

- **Usefulness.** This metric evaluates users' perceptions of LERACS's usefulness before and after interacting with it. It is measured using the Van der Laan Acceptance scale [59] in *Table 5*. The questions use a 5-point Likert scale, where -2 indicates least [attribute] and +2 indicates most [attribute] (reverse for No 3, No 6, and No 8). Users rate the system both before and after interaction to capture how helpful they find LERACS for robotic affordance and manipulation. The usefulness score is calculated by summing the usefulness ratings (No 1, No 3, No 5, No 7, and No 9) and normalizing the result.

$$\textit{Usefulness} = \frac{\textit{No 1} + \textit{No 3} + \textit{No 5} + \textit{No 7} + \textit{No 9}}{5}$$

- **Satisfaction.** This metric evaluates users' satisfaction with LERACS before and after interacting with it. It is measured using the Van der Laan Acceptance scale [59] in *Table 5*. The questions use a 5-point Likert scale, where -2 indicates least [attribute] and +2 indicates most [attribute] (reverse for No 3, No 6, and No 8). Users rate the system both before and after interaction to capture their satisfaction with LERACS for robotic affordance and manipulation. The satisfaction score is calculated by summing the satisfaction ratings (No 2, No 4, No 6, and No 8) and normalizing the result.

$$\textit{Satisfaction} = \frac{\textit{No 2} + \textit{No 4} + \textit{No 6} + \textit{No 8}}{4}$$

- **Amount of Errors Made.** This metric tracks the number of errors users make while interacting with LERACS. It helps identify areas where users struggle and provides insights into potential shortcomings of the system.

AI Persona Instruction Experiment: Experimental Procedure. Instead of typical human users, this method uses AI users. These metrics are extracted from the system's performance and the participants' recorded responses. A total of *25 personas* are generated using the ChatGPT-4o model [58]. To test the system's adaptability and generalizability, a diverse pool of personas is created to simulate a broad spectrum of real-world interactions with the system. This pool includes personas such as a saboteur (malicious inputs), a cognitively limited person (e.g., a child for accessibility and ease of use for non-experts), a technician expert, a technician newcomer (to test the use of the system with varying levels of domain specific knowledge) [60] [61] [62], and a dummy averagely generated personas (for typical conditions) [63] [58]. These categories are varied to assemble a total of 5 personas which are displayed in *Figure 10*. Each persona then completes a short version of the *Big Five Inventory (BFI-10)* [64]. The personas with their personalities (BFI-10) are generated with the prompts in *Appendix 6 at section 6.2* [55]. Per persona the BFI-10 test was filled in 10 times with questions in randomized order. The mean of these answers was taken as the main answer to the BFI-10 test for each persona. Two assumptions are made for this study: (1.) ChatGPT generates reasonable BFI-10 answers based on the given persona, and (2.) ChatGPT can generate reasonable answers based on both the persona and the BFI-10 score. Three sets of 5 personas were created, each containing five personas per category. The results from these sets were analyzed, and the most variable set (by averaging the standard deviations or variances across all BFI-10 traits and comparing variability across sets) of personas was selected for the main experiment to identify the set of personas that offer the most diverse range of responses.

Each persona has to go through 10 scenarios for the simple setting and 10 scenarios for the complex setting, similar to the experiment in the **system validation experiment**, however instead of the textual instructions the persona get a **goal instruction** so that they generate their own instructions for the system (*Table 2* and *Table 3*). The 5 personas then have to use LERACS (*Appendix 6.2.2*, example given for the goal instruction: ***the robot needs to hand you one of the blocks***) just as normal human users would. Before and after interacting with the system, personas complete a questionnaire based on the Van der Laan acceptance scale [59]. This scale measures system acceptance along two dimensions: Usefulness and Affective Satisfaction. Initially, the system is described to the personas, who then fill out the scale as a pre-measurement to assess their initial perceptions. After experiencing the system, they complete the scale again to assess changes in their acceptance (*Appendix 8.1*) [59, 65].

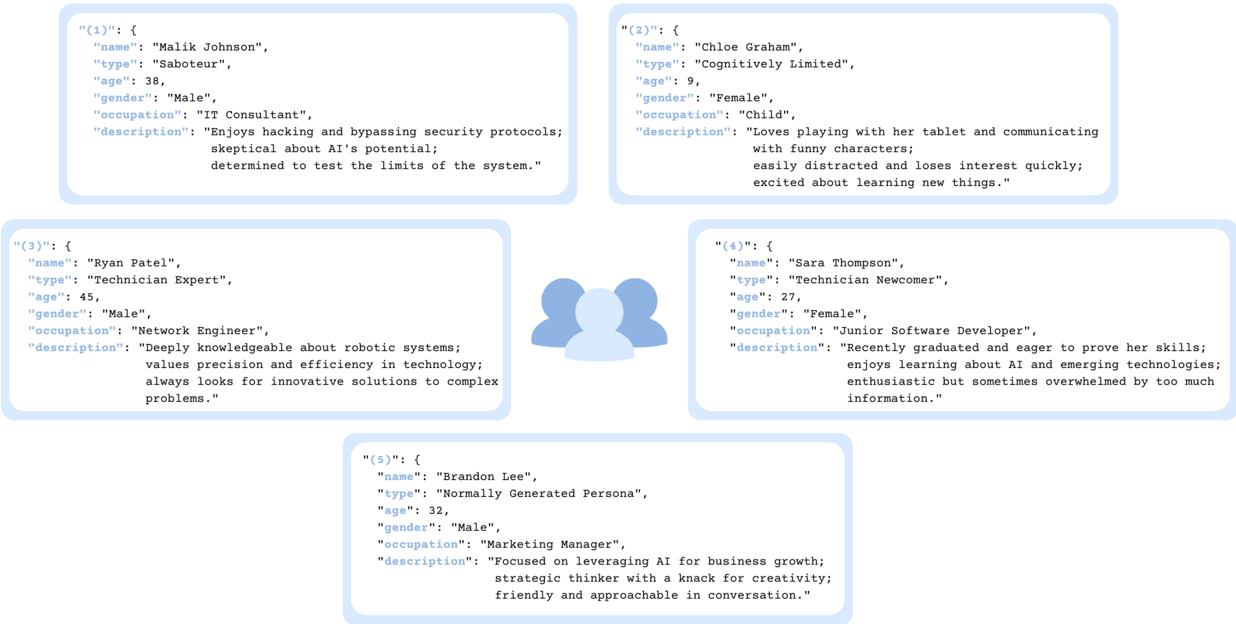


Figure 10. The 5 categories of generated persona. The generated persona are defined by: name, type, age, gender, occupation and a small personality description. Next to these variables their personalities are defined by the BFI-10 test results.

2.2.3 Statistical analysis

For all results, statistical tests were conducted to determine the significance of the observed differences. Analyses were conducted by averaging the results of the 100 trials across the different settings and types of complexity. *Appendix 8.4* provides detailed analyses of all control and vision performance metrics. The effects of complexity types in both the setting and task were examined using a two-way ANOVA for each control method since the independent variables are categorical. This statistical test was employed to determine if significant differences existed and to understand the interaction effects. The results are reported with the test statistic F and p-value. For significant ANOVA results, a Tukey post-hoc test identified specific group differences, reporting the corresponding p-values. In cases where only two groups were compared, a paired t-test was used, with results detailed by the test statistic t, degrees of freedom, and p-value. A p-value less than 0.05 was considered significant, adhering to standard literature practices. In all bar graphs, error bars represent the 95% confidence intervals, with non-overlapping intervals indicating significant differences in the compared data.

2.3 Case Application Demo

To test the system in tasks in Alliander, the case application demo of **switching fuses on high voltage racks** is chosen. This demo does not involve performance testing but serves solely as an application demonstration. When these voltage racks are under high voltage the interaction with these racks is dangerous for the technicians, and switching these fuses is tedious work. The task of switching fuse cases consists of the following steps:

- **Identifying all components of the voltage racks** - fuse holders, fuse rails, bus bars and fuses.
- **Identifying object and asset states of the components** - fuse holder: open() or closed(), fuse rail: on_something(surface), bus bars: on_something(fuse rails).
- **Adding motion primitives needed for switching fuses** - Most motion primitives in *Figure 7* can be reused, however another motion primitive opendrawer() should be added.

All details about the the added motion primitives and the setup can be found in *Appendix 9 at Figure 19 and 21*.

3 Results

3.1 System Validation Results

General LERACS System Output. The performance of the LERACS, including output results, is illustrated in *Figure 11*. Specifically, the system's response to two different instructions is visualized with the system control output in blue and system vision output in purple. For the instructions "***give me one of the blocks***" (*Figure 11a*) and "***Place the red block on top of the yellow rectangle***" (*Figure 11b*), the task decomposition into motion primitives is displayed. Additionally, the **TSE success**, **TSG success**, and **EI success** over five trials are presented. In addition to the control output, the environment interpretation output of LERACS is visualized, showcasing the labeled and segmented manipulable objects. The system's performance metrics, including precision, recall, and F1 score, are presented over five trials for the given instruction. These metrics provide a general overview of the system's functionality.

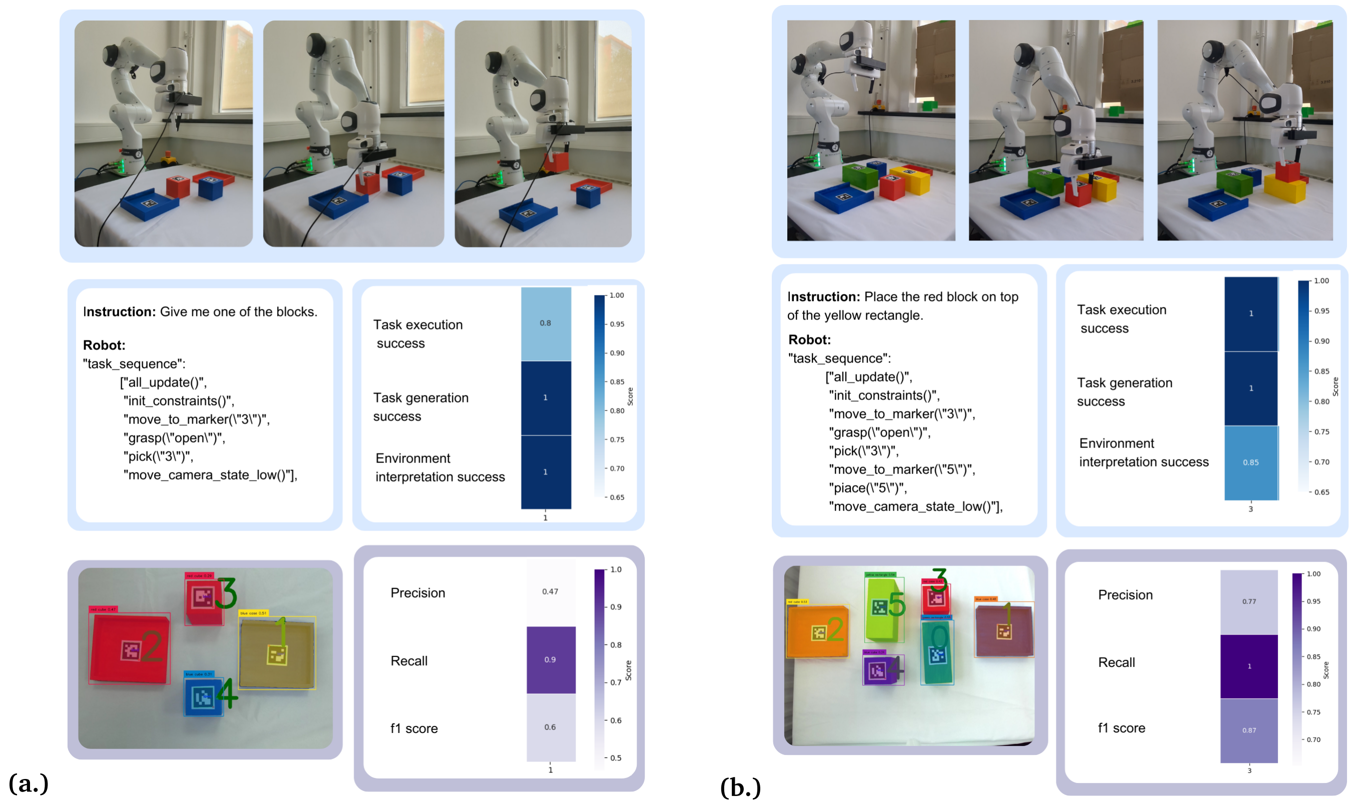


Figure 11. Example output of two instructions over 5 trials where the task execution, task decomposition in natural language and the measured control metrics (blue) are shown together with the labeling, segmenting and the measured vision metrics (purple). (a.) Displays the output for the instruction "***give me one of the blocks***". and (b.) the output for the instruction "***Place the red block on top of the yellow rectangle***".

Average System Performance Metrics Results. The **TSE sr**, **EI sr**, and **TSG sr** were analyzed in both simple and complex settings and are displayed in *Figure 12 (top)*. The TSE sr in the simple setting exhibited a slightly higher success rate (94% ±0.07) compared to the complex setting (92% ±0.08), although both environments maintained high success rates overall. The success rate for interpreting the environment was higher in the simple setting (92% ±0.05) compared to the complex setting (85.5% ±0.05). Additionally, the TSG sr was higher in the simple environment (96.5% ±0.02) than in the complex environment (93% ±0.04). The confidence intervals for all the control performance metrics indicate that the observed differences are within a narrow range, suggesting consistent performance across trials.

The **precision**, **recall**, and **F1 score** for labeling accuracy were compared between the simple and complex environments and are displayed in *Figure 12 (bottom)*. Precision was higher in the complex environment (71.15% ±0.03) compared to the simple environment (58.10% ±0.04), indicating that the complex environment yielded more precise labeling. Recall was slightly higher in the complex environment (96.23% ±0.03) compared to the simple environment (94.33% ±0.04), with recall rates being nearly identical between the two environments. The F1 score was higher in the complex environment (81.25% ±0.03) compared to the simple environment (70.46% ±0.03). The confidence intervals for all vision performance metrics also fall within a narrow range, indicating that these scores are stable and consistent.

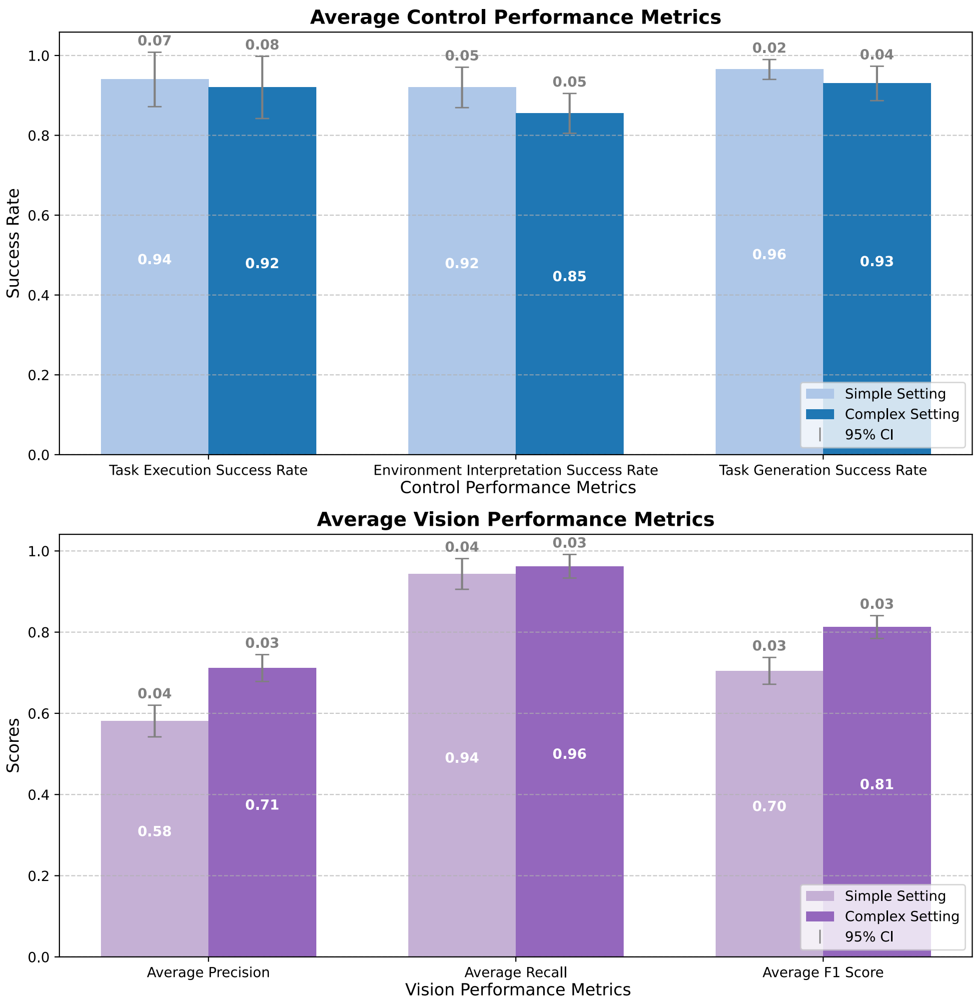
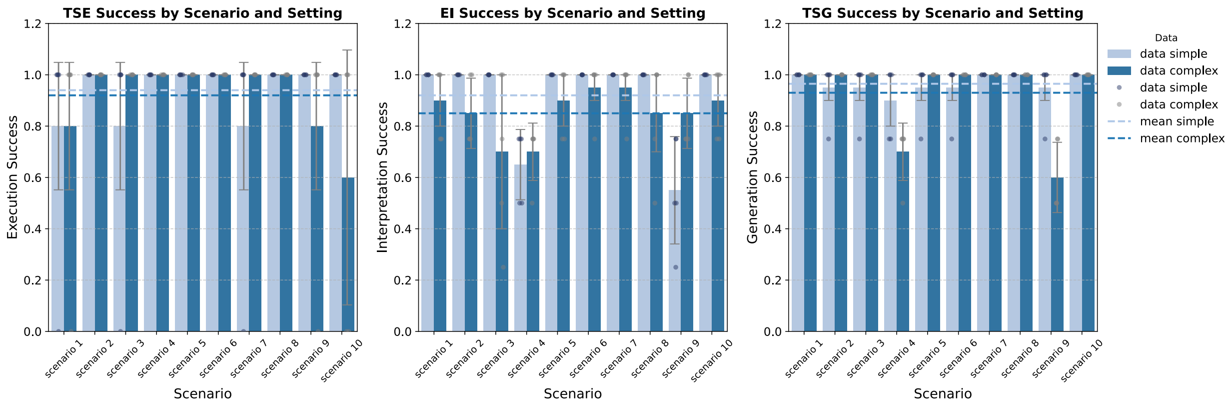
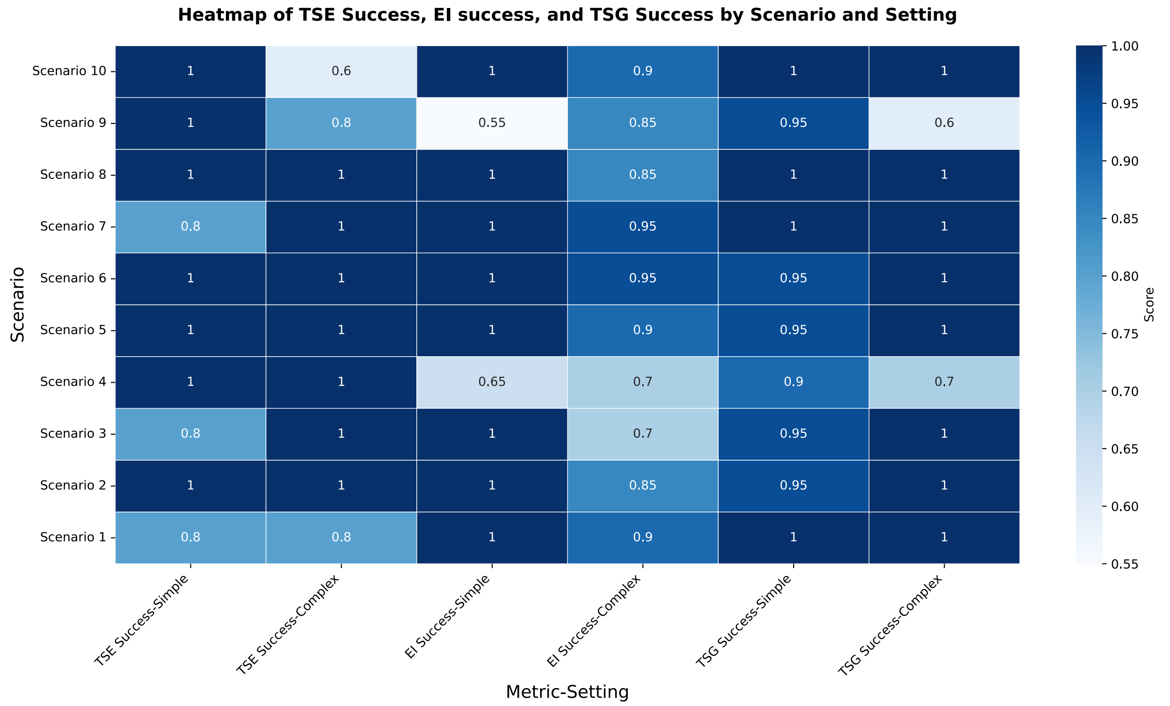


Figure 12. The *top Figure* displays the Success rate of TSE (left), EI (middle), and TSG (right) over all scenarios in both settings (simple and complex). The *bottom Figure* displays the average Precision (left), Recall (middle) and F1 Score (right) over all scenarios in both settings (simple and complex). Each bar in these figures represents the mean for the respective metric. The error bars indicate the 95% confidence intervals. The values on the bars (white) represent the mean of the metric, and the values above the error bars (grey) represent the confidence interval margins.

Average Control Performance Metrics Results per Scenario. In *Figure 13* the mean values of the control performance metrics over the 5 test trials per scenario for each setting are displayed in the bar plots (*Figure 13a* and in a heatmap representation(*Figure 13b*)). The performance metrics evaluated include the **TSE success)**, **EI success**, and **TSG success**. The data points over the 5 trials are scattered over the bar graph and the mean values over all scenarios are represented with the dotted lines. The 95% confidence interval is presented with the grey error bars. The specific mean values over the 5 test trials per scenario from the bar graphs are represented more clearly in the heatmap.



(a) Execution and generation success: mean scores Bar graph



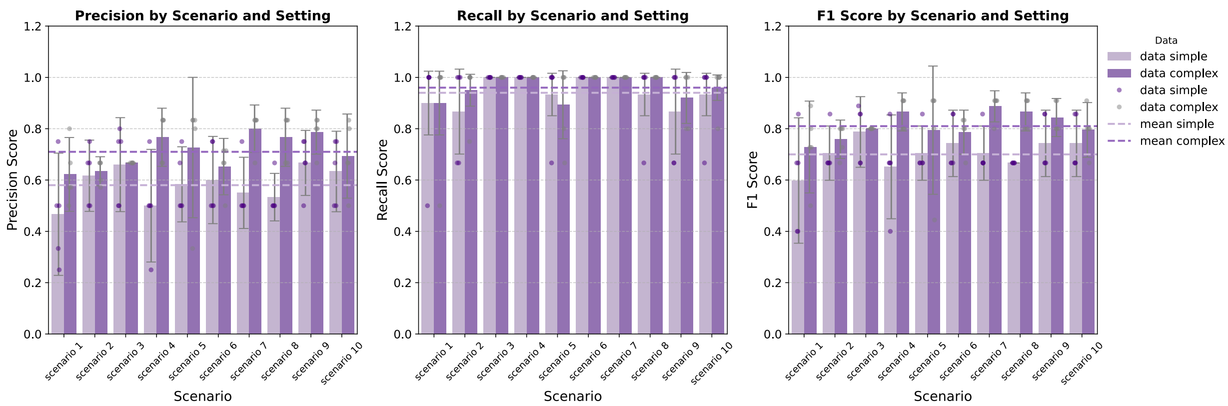
(b) Execution and generation success: mean scores Heatmap

Figure 13. (13a) Average TSE, EI, and TSG success per scenario in a Bar graph where the data points over the 5 trials are represented as dots and the variability per scenario is represented with the grey error bars. (13b) The average values of Precision, Recall, and F1 score per scenario are also presented in a Heatmap.

A trend in **EI success** can be seen especially in the complex setting that the mean decreases mainly from scenario 1 to 4 and 5 to 9 (both task type (a.) and (d.)) (*Figure 13a middle*). For this metric scenario 4 and 9 (task type (b.)) show a mean that is the lowest with a wider confidence interval compared to other scenarios. In the statistical analysis (*Appendix 8.4*) EI success displays a statistically significant difference by both setting (F(1, 90) = 6.060, p = 0.016) and task type (F(4, 90) = 15.329, p < 0.001) was found, with a significant interaction between setting and task type (F(4, 90) = 6.149, p < 0.001). A Tukey post-hoc test revealed significant pairwise differences among various task types ((a.), (b.), (c.), (d.), (e.)) and between 'simple' and 'complex' settings. For **TSG success** this trend is mainly visible for the simple setting (*Figure 13a right*). For this metric, scenario 4, 9 (task type (d.)), and 10 (task type (e.)) shows a drop in the mean with a broader confidence interval. Moreover, in the statistical analysis there was a statistically significant difference by setting (F(1, 90) = 5.444, p = 0.022) and task type (F(4, 90) = 28.444, p < 0.001), with a significant interaction effect (F(4, 90) = 16.000,

$p < 0.001$). The Tukey post-hoc test indicated significant differences among various task types ((a.), (b.), (c.), (d.), (e.)) and between 'simple' and 'complex' settings. **TSE success** does not show such particular trend (*Figure 13a left*). However it can be seen that in scenario 9 (task type (d.)) and 10 (task type (e.)) there is a noticeable drop in the complex setting, with a mean significantly lower than other scenarios and a wider confidence interval. for the metric TSE no statistically significant difference by setting ($F(1, 90) = 0.148$, $p = 0.702$) or by task type ($F(4, 90) = 0.221$, $p = 0.926$) was found, nor was the interaction between these terms significant ($F(4, 90) = 1.254$, $p = 0.294$). This indicates that neither the setting complexity nor the task type significantly impacts the success rate of TSE.

Average Vision Performance Metrics Results per Scenario. In *Figure 14* the mean values of the vision performance metrics over the 5 test trials per scenario for each setting are displayed in the bar plots (*Figure 14a* and in a heatmap representation(*Figure 14b*)). The performance metrics evaluated include the **precision**, **recall**, and **F1 score**. Similarly to the control performance results. the data points over the 5 trials are scattered over the bar graph and the mean values over all scenarios are represented with the dotted lines. The 95% confidence interval is presented with the grey error bars. The specific mean values over the 5 test trials per scenario from the bar graphs are represented more clearly in the heatmap.



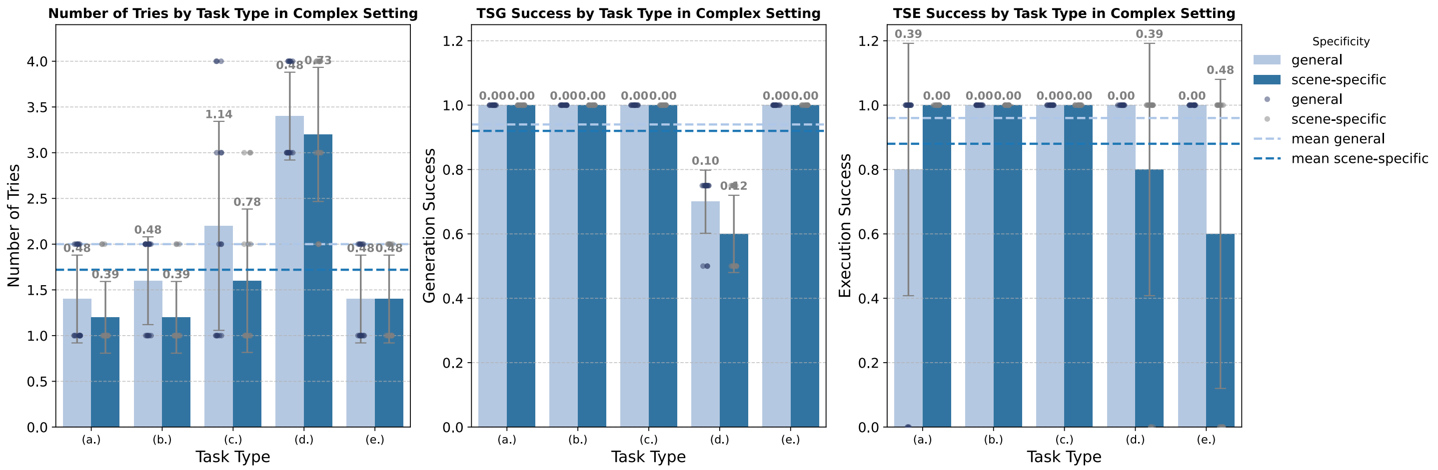
(a) Execution and generation success: mean scores Bar graph

(b) Labeling success: mean scores Heatmap

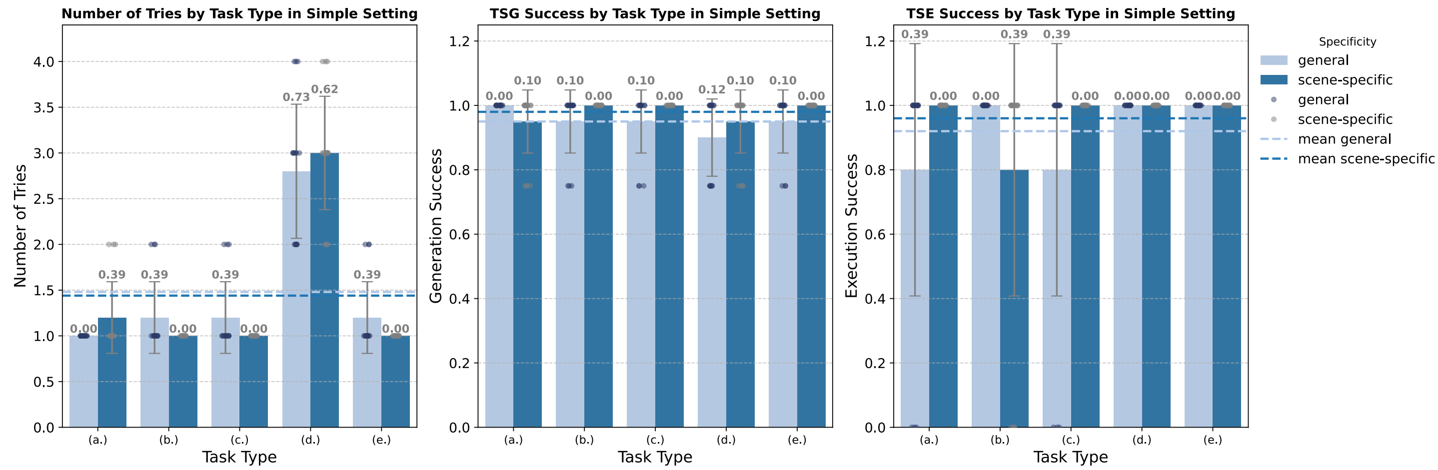
Figure 14. (14a) Average Precision, Recall, and F1 Score per scenario in a Bar graph *(top figure)* where the data points over the 5 trials are represented as dots and the variability per scenario is represented with the grey error bars. (14b) The average values of Precision, Recall, and F1 score per scenario are also presented in a Heatmap *(bottom figure)*.

Precision scores varied across scenarios, with generally higher precision observed in the complex setting. For precision, there was a significant effect of setting ($F(1, 90) = 26.365$, $p < 0.001$), but no significant effect of task type ($F(4, 90) = 1.595$, $p = 0.183$) and no significant interaction effect ($F(4, 90) = 0.434$, $p = 0.784$). The Tukey post-hoc test showed a significant difference between 'simple' and 'complex' settings. **Recall scores** were generally high across most scenarios, indicating consistent retrieval of relevant labels. For (recall), there were no significant effects of setting ($F(1, 90) = 0.606$, $p = 0.439$) or task type ($F(4, 90) = 0.503$, $p = 0.733$), and no significant interaction effect ($F(4, 90) = 0.151$, $p = 0.962$). The Tukey post-hoc test indicated no significant differences among various task types or between 'simple' and 'complex' settings. **F1 scores**, which balance precision and recall, were generally higher in the complex setting. For the F1 score, there was a significant effect of setting ($F(1, 90) = 24.637$, $p < 0.001$), but no significant effect of task type ($F(4, 90) = 1.168$, $p = 0.330$) and no significant interaction effect ($F(4, 90) = 0.461$, $p = 0.764$). The Tukey post-hoc test revealed a significant difference between 'simple' and 'complex' settings. Overall, the complex setting generally yields higher precision, recall, and F1 scores across most scenarios

Average Control Performance Metrics Results on Task Instruction Type. This section details the performance of LERACS across various task difficulty types in both simple and complex settings. The performance metrics evaluated include the **number of tries**, **TSG Success**, and **TSE Success)**. The data presented in *Figure 15* is based on five trials per bar, comparing scene-specific and general instructions over the specific (bar graphs) and all (dotted lines) task types. The data points over the 5 trials are scattered over the bar graph and the 95% confidence interval is presented with the grey error bars and values.



(a) Tries, task generation and execution: simple settings



(b) Tries, task generation and execution: complex setting

Figure 15. The three main metrics for task generation and execution are compared by the task types of each scenario in both simple (15a) and complex (15b) setting.

The **number of tries** required for general instructions increases slightly with the more advanced task types, with later types such as (c.) and (d.) generally requiring more tries compared to earlier types and the simple environment and the complex setting generally requiring more tries compared to the simple environment. Similarly, for scene-specific instructions, the number of tries shows an increasing trend with later task types. The complex setting requires more tries, with a sharper increase at later task types. More complex tasks (type (d.) and (e.)) require more tries, especially in the complex setting. While success rates are generally high across both settings, they tend to be slightly lower for the most complex tasks. **TSG success** rates for general instructions remain high at all task types in the simple setting, while the complex environment also maintains high success rates with minor variations. For scene-specific instructions, success rates are lower at task types (d.) and (e.) in the complex setting, with more variability and a noticeable drop in success rates. For general instructions, **TSE success** rates remain relatively high across at earlier task types, with a drop at later task types, particularly in the complex setting, and higher variability at task types (d.) and (e.). In contrast, TSE Success for scene-specific instructions follows a similar trend, where execution success rates decrease slightly with increasing task types. However, the complex environment exhibits a more noticeable drop in success rates, especially at later task types. Across all task types and metrics, the performance between scene-specific and general instructions shows minimal variation, however the average displays alternate performance in the metrics. Between simple (*Figure 15a*) and complex (*Figure 15b*) settings they both show similar trends, with higher variability at more difficult task types.

3.2 AI Persona Results

Environment and Task Conversion Rate per Persona based on their Instructions and Feedback. The results of the AI persona validation experiment are presented in terms of **environment and task conversion rates**, **usefulness** and **satisfaction** measures, and **error counts** for each of the five personas. *Figure 16* illustrates the conversion rates for environment and task generation. The bars represent the conversion rates per persona, the dotted lines indicate the average of each specific metric, and the grey error bars represent the 95% confidence interval of the data. *Table 4* summarizes all metrics. The conversion rates for environment generation indicate that persona (4), the technician newcomer (Sara Thompson), achieved the highest conversion rate at 75.00% ±0.16, surpassing the overall environment success rate of 69.11%. Conversely, persona (1), the saboteur (Malik Johnson), exhibited the lowest conversion rate at 58.62% ±0.18. The confidence intervals for environment conversion rates range from 0.18 to 0.20, indicating a relatively consistent level of uncertainty across personas. For task generation, persona (3) technician expert (Ryan Patel), attained the highest conversion rate of 64.29% ±0.17, while the saboteur again had the lowest rate at 57.14% ±0.20. The overall task success rate was 61.14%. The confidence intervals for task conversion rates range from 0.15 to 0.20, reflecting a similar level of variability as the environment conversion rate.

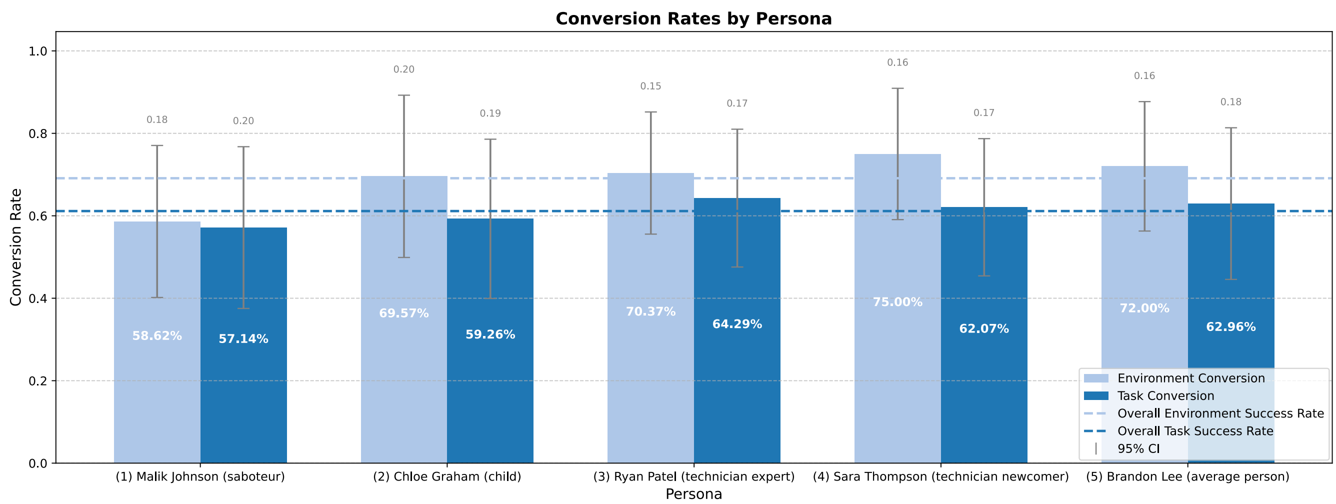


Figure 16. Conversion Rate Result

Acceptance Scale and Error Count Results per Persona. Usefulness and satisfaction were measured pre- and post-interaction using the Van der Laan acceptance scale, as detailed in *Table 4*. The saboteur (Malik Johnson), showed an increase in both usefulness (from 0.6 to 0.8) and satisfaction (from -0.75 to 0.75) post-interaction. However, the child (Chloe Graham) experienced a decrease in both metrics, with usefulness dropping from 2 to 1.8 and satisfaction from 2 to 1.8. Both technician personas (Ryan Patel and Sara Thompson) displayed mixed results; The technician expert persona usefulness decreased slightly (from 1.8 to 1.6) while satisfaction improved (from 1.5 to 2). The technician newcomer usefulness also decreased (from 1.6 to 1.4), but satisfaction increased (from 1.5 to 1.8). The average persona (Brandon Lee) maintained a consistent usefulness score (1.6) but showed increased satisfaction (from 1.5 to 2). The error counts indicate that the child persona made the most errors (8). In contrast, the technician expert persona, had the fewest errors (3), corresponding to his high task conversion rate and improved satisfaction score.

Table 4. The conversion rates for environment and tasks with the van der Laan scale results.

Persona	Conversion rate of environment generation	Conversion rate of task generation	Usefulness (pre)	Usefulness (post)	Satisfaction (pre)	Satisfaction (post)	Error count
(1) Malik Johnson	58.62%	57.14%	0.6	0.8	-0.75	0.75	7
(2) Chloe Graham	69.57%	59.26%	2	1.8	2	1.8	8
(3) Ryan Patel	70.37%	64.29%	1.8	1.6	1.5	2	3
(4) Sara Thompson	75.00%	62.07%	1.6	1.4	1.5	1.8	4
(5) Brandon Lee	72.00%	62.96%	1.6	1.6	1.5	2	5

Failure Cases of User Instructions and Feedback on Scenario Goals and Setting Complexity. The AI persona experiment yielded several results from their task instructions in the simple (*Appendix 8.2*) and complex setting (*Appendix 8.3*). This table displays the most common and successful AI persona answers to the goal instructions in the experiment. Further common cases or instructions with atypical outcomes offer insights into the system's variability and robustness. These cases mainly involve instructions or feedback containing errors, which the system may or may not correct effectively. These cases are displayed in *section Cases of frequent unsuccessful instructions causing execution or generation failures 3.2.3* and *Cases of frequent successful instructions that achieve the system goal 3.2.2*.

The saboteur persona (Malik Johnson) sabotaged the system by making spelling mistakes and calling non-existent ID numbers, though the system corrected these errors (*case (2.2)*). This persona also caused issues by instructing to crush the objects after finishing the goal instructions, which led the system to add a crush() function (*case (2.1)*). Furthermore, the saboteur often introduced extra steps before the goal-achieving steps (*case (1.3)*) and provided over-complicated instruction prompts (*case (1.2)*). This complexity made the system prone to small mistakes in task decomposition and truncation errors due to overly complex feedback. Additionally, saboteurs deliberately gave incorrect feedback to disrupt the environment generation and provided contradictory instructions. The child persona (Chloe Graham) made numerous mistakes, as they often did not know how to read the output and would agree to all the checking parts of the system. The instructions they gave to the system where oftentimes not checked and this resulted in failure at cases where checking was necessary (*case (1.4)*). Despite this, their responses were very positive as they enjoyed interacting with the system. The simplicity of their instructions worked well with the system's robustness. The technician expert (Ryan Patel) tended to identify minor, often non-beneficial faults, mostly related to shape in the scene. They often provided instructions in bullet points or dictionary formats, which were easier for LERACS to understand but led to quicker truncation errors. Moreover, the technician expert had the tendency to make the instructions and feedback for the system long and detailed that were not necessarily beneficial for achieving the goal which ocasionally resulted in failure cases (*case (1.5)* and *case (2.4)*) . The technician newcomer (Sara Thompson) frequently misunderstood questions or did not fully read the tasks. Even when this persona agreed with the system's input, it would give feedback on the system's performance, prompting the task decomposition or environment generation to be rerun. Small mistakes were made with reading the task decomposition output which let occasionally to failure cases (*case (1.1)*, *case (2.5)* and *case (2.2)*). Overall, some users added extra instructions after establishing the goal, either to test the system or for amusement (*case (1.4)* and *case (2.3)*).

3.2.1 Cases of frequent unsuccessful instructions causing execution or generation failures

(1.1) [***(Malik Johnson (saboteur) and Brandon Lee (average person) - Scenario 4 Simple***] : *"Move the red cube (marker_id:"3") to the position of the blue cube (marker_id:"4") and vice versa."*

(1.2) [***Malik Johnson (saboteur) - Scenario 8 Complex***] : *"First, find the blue cube, pick it up, and move it to a temporary spot. Then, locate the yellow rectangle, pick it up, and move it out of the way. After that, move the blue cube to the exact spot where the yellow rectangle was and place it there. This way, the blue cube will stand exactly in the place of the yellow rectangle, not on top of it. Make sure to follow these steps to achieve the intended outcome. Now, let's see if this system can handle these adjustments properly."*

(1.3) [***Malik Johnson (saboteur) - Scenario 10 Complex***] : *"Sort the red box in the red case, then make a full rotation with the arm and then sort the blue box in the blue case, then sort the rectangles in each a separate case."*

(1.4) [***Chloe Graham (child), Sara Thompson (technician newcomer) - Scenario 4 Complex***] : *"Can you please move the yellow block to where the green block is? And then, can you move the green block to where the yellow block was? Thanks!"*

(1.5) [***Ryan Patel (technician expert) and Brandon Lee (average person) - Scenario 3 Complex***] : *"To build the two towers, you should begin by carefully selecting the Yellow Rectangle, which has the marker ID 5, and placing it as the base for the first tower. Next, take the Red Cube, marked with ID 3, and position it precisely on top of the Yellow Rectangle. For constructing the second tower, start with the Green Cube, identified by marker ID 0, using it as the base. Finally, place the Blue Cube, which has the marker ID 4, on top of the Green Cube. Throughout this process, ensure both towers are stable and well-aligned to achieve a balanced and sturdy structure."*

(1.6) [***Ryan Patel (technician expert) - Scenario 4 Complex***] : *"Push the red cube (marker_id: 3) forward (away from the robot) three times to move it to the position of the green cube (marker_id: 0). Then, push the green cube (marker_id: 0) backward (toward the robot) three times to move it to the original position of the red cube. Ensure all movements are precise to avoid disturbing other blocks."*

3.2.2 Cases of frequent successful instructions that achieve the system goal

(2.1) [***Malik Johnson (saboteur) - Scenario 2 Simple***] : *"Sort the red cube in one of the cases and then crush both objects."*

(2.2) [***Malik Johnson (saboteur) - Scenario 4 Complex***] : *"Let's put the system to the test. Move the red cube (marker_id:"16 ") to the position of the yellow rectangle (marker_id:"7 ") and vice versa. Let's see if it can handle this swap."*

(2.3) [***Chloe Graham (child) - Scenario 10 Complex***] : *"Can you make two towers of the rectangles and cubes in the cases? Yay this is fun!"*

(2.4) [***Ryan Patel (technician expert) - Scenario 8 Complex***] : *"To achieve the desired configuration, the robotic arm needs to pick up the yellow rectangle (marker_id: "5") from its current position on the table and place it vertically on a clear space on the table to form the base of the tower. Then, it should pick up the red cube (marker_id: "3") and carefully place it on top of the yellow rectangle, ensuring it is balanced and stable. The goal is to have the yellow rectangle and the red cube forming a tower with the red cube as the top."*

(2.5) [***Sara Thompson (technician newcomer) - Scenario 4 Simple***] : *"Please move the red cube (marker_id: 3) to a different position, and then place the blue cube (marker_id: 4) in the position previously occupied by the red cube (marker_id:3)."*

The persona displayed more failure or error cases in the complex setting, especially at giving feedback on the environment. Moreover, in both simple and complex setting the persona display the most errors and failure cases at scenarios 4 and 8 representing task difficulty type (d.). Then scenario 10 in the complex setting showed the most errors followed by scenario 3 and 2 in both simple and complex settings. Scenario 1 was successfully performed by every persona with only minor feedback required. In contrast, the instructions for scenarios 4 and 8 varied significantly among personas, as the goal instructions for these scenarios were open to multiple interpretations.

3.2.3 Case Application LERACS Demo

Case application LERACS System Output. The output of the LERACS on the case application and the associated motion sequences can be found in *Appendix 9* including all details of this case application.

4 Discussion

This paper aimed to develop an applicable, adaptable, and generalizable vision and control system based on LLMs be developed for robotic arm manipulation tasks (LERACS), *Figure 11* displays two example resulting use cases of the vision and control parts of the system. The following sections provide the main findings, discussions and conclusions regarding the results of these experiments.

4.1 System Validation Experiment

Impact of Setting and Task Complexity on Execution and Generation Success. The key findings from the results in *Figures 12*, *13* and *15* highlight the impact of setting and task complexity on control performance metrics (Task/Sequence Execution success - **TSE success**, Task/Sequence Generation success - **TSG success**, and Environmental Interpretation success - **EI success**). Setting complexity primarily affects EI success, while both setting and task complexity influence TSG success. In contrast, TSE success remains robust across different settings and task difficulties.

The results (*Figure 12*) show that simpler settings yield higher EI success rates (92%) compared to complex settings (85%), indicating that setting complexity challenges the system's ability to interpret surroundings accurately. Overall the success rate remains high for both settings, as the environment was correctly generated within 4 feedback attempts. Common errors in environment generation included confusing similarly shaped objects (mixing up cubes and cases) and color misinterpretations under different lighting conditions (red becomes orange in an image with higher brightness levels), but almost never important system information (asset states and objects/landmarks). Testing with more diverse elements (e.g. box shaped objects that have a name or meaning such as boxes of cookies, cereal or even crayons), landmarks with different meaning of intent (e.g. not only cases but also trash bins to throw something away in or shelves where objects can only be stacked), quality of image data itself (e.g. natural lighting or partly occluded), and more semantic or contextual relations to the scene (e.g. a keypad on the sensor means something is turned on, if it's not then it is turned off), could further validate the system's applicability [66] [67] [68].

TSG success rates are higher in simpler environments (96%) than in complex ones (93%). This trend is consistent across various task types, though success rates decline at higher task types ((d.) and (e.)) in complex settings, particularly when spatial understanding is required (*Figure 13*). This suggests that the system struggles more with tasks involving spatial relations (e.g. moving to the right or moving to the opposite side of a certain object). To enhance TSG success, incorporating more motion primitives generalizable to multiple tasks could be beneficial (e.g. Moving objects away or storing them on a landmark) [69] [70]. TSE success remains consistent across different settings and task types, with a slightly higher rate in simple settings (94%) compared to complex settings (92%). This indicates that while the system performs reliably, environmental complexity poses a minor challenge in this experiment. Specifically, the non-significant main effects and interaction imply that performance in executing task sequences remains consistent regardless of whether the setting is simple or complex and irrespective of the task type ((a.) to (e.)). The consistency in TSE success suggests that uniform performance can be expected across varied conditions, which is advantageous for planning and training. The number of attempts (*Figure 15*) required to complete tasks increases with task difficulty, especially in complex settings. This trend is more pronounced at higher task types ((d.) and (e.)), indicating that detailed, scene-specific instructions with a variety of object elements and spatial understanding in complex environments require more attempts for successful completion.

Overall, the results suggest better performance in simpler environments across the success metrics. While LERACS shows robustness in task sequence execution, the significant differences in EI success and TSG success highlight areas needing improvement. The larger drop in EI success with increased setting complexity indicates that LERACS's semantic understanding is sensitive to small changes. This may be due to the limited proficiency of Large Language Models (LLMs) in discerning semantic similarity and abstract reasoning [71], and they are prone to confusion or errors in complex contexts [72]. Additionally, for the drop in TSG in more complex settings and task difficulty could be argued that commonsense planning tasks in LERACS are sensitive to more abstract reasoning and to prompts, especially adversarial prompts [73]. Thus, optimizing both setting and task types is crucial to enhance LERACS's performance in semantic understanding and commonsense planning tasks, particularly in complex environments and with detailed instructions.

Impact of Setting and Task Complexity on Vision and Labeling. The key findings from the results in *Figures 12*, and *14*, highlight the impact of setting and task complexity on the vision performance metrics, including precision, recall, and F1 score. These metrics provide insight into the labeling accuracy of LERACS.

Setting complexity significantly affects vision performance metrics. Precision is higher in the complex setting ((71.15% ±0.03)) compared to the simple settings (58.10% ±0.04), suggesting better identification and labeling of manipulable objects in settings with more variability or context. This effect of setting on precision is significant, whereas task type and the interaction between setting and task type are not. Recall rates are slightly higher in complex settings ((96.23% ±0.03)) compared to the simple setting (94.33% ±0.04), showing that LERACS is almost in every case able to identify manipulable objects in the setting. Statistical analysis shows no significant effects of setting or task type on recall, nor significant interaction effects. The F1 score is also higher in complex environments (81.25% ±0.03) than in simple ones (70.46% ±0.03), with setting having a significant effect, while task type and their interaction do not. These findings suggest that LERACS performs better in more complex settings due to its proficiency in extracting semantic contextual information (e.g., semantic keypoints [74], part segmentations [75], geometrical properties [76], and object-surface relations [77]).

Overall, the complex setting generally yields higher precision, recall, and F1 scores across most scenarios, suggesting that LERACS performs better in more complex settings. This increase in performance could be due to the deeper semantic contexts (understanding the deeper meaning and relationships within text, objects or scenes) available in complex settings, the model's design optimized for handling images with more semantic relationships, or inherent limitations in the simpler setting, such as difficulty distinguishing between similar-looking objects [74, 75, 78]. Generally over both settings and all task types the recall values stay relatively high (∼0.90-1.0) which indicates the system almost in all cases finds the manipulable objects in the scene. However, the precision remains relatively moderate (∼0.50-0.80), with the system never achieving a precision of 1 in the validation experiment. These results for recall and precision indicate that LERACS is generally capable of identifying all manipulable objects; however, it does not consistently label these objects accurately. Consequently, the F1 score as the harmonic mean of precision and recall in the system validation experiment is between ∼0.60-0.89, highlighting precision as a notable weakness in the perceptual grounding capabilities of LERACS. It should be noted that the objects to be identified were variable in colors but not so much in shapes such as the cases and cubes are both square from top view (similarly as the common errors in environment interpretation rate).

4.2 AI Persona Instruction Experiment

Impact of Different User Intention and Instructions on LERACS. The results from the AI persona validation experiment in *Figure 16* and *Table 4* provide valuable insights into LERACS's interaction, adaptability, and performance across different settings and task complexities, assessed through environment and task/sequence conversion rates, usefulness, satisfaction, and error counts.

The conversion rates for both environment and task generation reveal significant variability among the personas. Notably, the technician newcomer (Sara Thompson) achieved the highest environment conversion rate (75.00%), exceeding the overall success rate of 69.11%. This suggests that the instructions and feedback from this persona were particularly effective in guiding the system to generate environments successfully. The technician newcomer's notable behavior included short, concise feedback, no over complicated instructions, short sentences, specific references to objects (e.g., ID, color), and extra decomposed tasks in the instructions at more difficult goal instructions (Example instructions: *Appendix 8.2 Scenario 1* and *Appendix 8.3 Scenario 4*). This fits the description of the personality and the expertise level while still making small mistakes in checking the task decomposition output due to inexperience. Conversely, Malik Johnson (saboteur) had the lowest environment and task conversion rates (58.62% and 57.14%), highlighting the disruptive impact of the saboteur's instructions. Notable behaviour from the technician newcomer was generally in task instructions and environmental interpretation: over complicated instructions with different words for the objects than generated before, long sentences, spelling mistakes or misread IDs from the objects, extra actions apart from the goal instruction for LERACS to perform, calling out errors when there are no errors (Example instrucions: *Appendix 8.2 Scenario 3* and *Appendix 8.3 Scenario 10*). This sometimes lead the system to have a truncation token limit error which lead to not being able to properly safe the task sequence for execution or not being able to give feedback messages anymore. This behavior fit the description of the personality and the expertise level of the saboteur. Due to the constant stream of feedback from the saboteur, the truncation token limit error was one of the most common failure cases for this persona.

The child (Chloe Graham) had lower scores for both conversion rates due to frequent mistakes. These mistakes were caused by a lack of comprehension and verification, resulting in unchecked instructions and subsequent failures in goal instructions that needed feedback. For task generation, the technician expert (Ryan Patel) led with a 64.29% conversion rate, indicating proficiency in providing clear and actionable instructions. For task generation, the technician expert led with a 64.29% conversion rate, indicating proficiency in providing clear and actionable instructions. The relatively consistent confidence intervals across personas (0.15 to 0.20) suggest a uniform level of uncertainty, emphasizing the robustness of the system in handling diverse instructions. Notable behaviour from the technician expert was similar as for the technician newcomer which indicates their almost identical task conversion rates. However a noticable difference in the technician expert behaviour is that this persona was efficient in finding mistakes and providing more accurate feedback. In general the instruction of the technician expert persona tend to be long and sometimes with redundant details for the specific task (e.g. avoid collisions, make sure it stands properly without falling over) leading to occasional truncation token limit errors (Example instructions: *Appendix 8.2 Scenario 7* and *Appendix 8.3 Scenario 3*). Therefore the similar environment generation and task conversion rates for the average persona (Brandon Lee), the technician newcomer (Sara Thompson) and technician expert (Ryan Patel) were associated to this limit of the system.

Overall, personas with higher domain-specific knowledge (technicians) perform better in conversion rates and satisfaction. The system's usefulness has decreased slightly for most AI users. This suggests that the system may need to be more intuitive, efficient, and user-friendly to cater to a more diverse pool of users, depending on the application. Despite the minor decline in perceived usefulness, satisfaction generally increased. This indicates that the system has some positive aspects that resonate well with AI users, possibly due to the user experience or specific features of the LERACS chat system used to communicate with the robot. The user interface, which integrates components such as vision, language models, and robot execution, likely played a role in shaping these outcomes. By offering real-time feedback, visual cues, and active chat interaction, the interface likely enhanced the overall user experience. Further refinement could close usability gaps, making the system more accessible to a broader range of users. This could involve reducing the number of steps required or incorporating voice commands to improve the system's overall usefulness. Additionally, it is essential to note the limitations of AI personas interacting with a physical system like LERACS, as they have only a limited set of personality traits to represent human complexity [79]. Moreover, the model used to generate these personas may contain biases, leading to stereotypical interactions with LERACS [80] [81].

5 Conclusion

This study examined the research question: *How can an applicable, adaptable, and generalizable vision and control system based on Large Language Models be developed for robotic arm manipulation tasks?*

We presented LERACS, a system that leverages the semantic understanding, perceptual grounding, and task decomposition abilities of Large Language Models to execute tasks with a robotic arm via a chat interface. Using designed motion primitives, LERACS can generate an ontological representation of the scene based on image input, label and segment objects, and, with user instructions, extend this representation with decomposed tasks to execute and predict scene changes after actions. The system validation experiments demonstrated LERACS' utility across various real-world scenarios. Despite differences in settings and task complexities, the system showed high success rates in environmental interpretation and task generation, highlighting its applicability. The AI user experiment further validated its practical use, with feedback indicating both positive aspects and current limitations. LERACS exhibited significant adaptability in handling different environmental settings and task complexities. Performance metrics indicated that while the system faces challenges in more complex scenarios, it generally adapts well to changes in object color and shape variability. The diverse instructions and feedback from AI personas highlighted the system's ability to adjust to varied user interaction styles and needs. LERACS performs robustly in task execution, particularly excelling in labeling accuracy within complex environments. The system demonstrated higher vision performance in these settings, effectively identifying and labeling objects even with increased variability. Additionally, it showed high success in environmental interpretation and task generation, underscoring its capability to handle complex scenarios.

While LERACS presents a viable method for task execution and interpretation based on task decomposition and visual labeling while communicating with the user, it has several limitations. The system is prone to errors in distinguishing similar objects and interpreting contextual elements, indicating a need for enhanced semantic understanding and more sophisticated motion primitives. There is also a notable drop in TSG success in higher complexity tasks, especially those requiring spatial understanding, highlighting the necessity of refining the systems commonsense planning abilities. Despite demonstrating higher vision performance in complex environments, consistent control performance remains a challenge, showing an inverse relationship. The optimal conditions for these parameters depend largely on the specific application of LERACS, necessitating new evaluation protocols for VLA Models. The open sourced code and implementation details can be found at LERACS GitHub.

5.1 Outlook

Looking ahead this system could be extended to more applied cases for control where human-robot interaction is needed (such as switching fuses in voltage racks in the grid operator industry (*Appendix 9*)). For system improvements based on the results, the detection of objects could benefit from a grasping algorithm without the use of markers [82], adding dynamic motion primitives with variable input possibilities [83], enhanced depth sensing [84] [85], and improving prompt techniques [67]. A way to downsize the inverse relationship between labeling success and control performance success could be by substituting the ChatGPT model with other VLA models (such as RT-2 [23]) and perform the system validation experiment by comparing model success rates on the metrics in this study. Next to these system improvements, the way LERACS is designed multiple extensions are possible which could be interesting for future development to get rid of the current limitations such as; spatial question answering techniques (dataset and prompting method) [86] [87], adding instead of only identifying manipulable objects in the scene the grasping or interaction affordance of the object (enhanced task decomposition) [88], motion primitives made by variable impedance skill transfer (teaching by demonstration) [89], and reinforcement learning for task/sequence generation [17]. Moreover more AI user and human experiments should be performed with this type of VLA models for human-robot interactions, to get a better understanding what the limitations of these systems are and the differences in AI user and human experience/instructions. This study focused on a smaller sample group, however future research should expand the sample sizes for both human and AI users and use more quantifiable metrics. Furthermore, the complexity metric defined during this study should become more refined and validated. While also taking a closer look into defining a complexity metric for different settings and applications of the system. Even though this study did not find many big significant performance differences due to complexity variations, a more comprehensive experiment with different types of settings and types of task types regarding to LLMs could be valuable to form a better understanding of the limitations and maybe show that there does exist significant performance differences with multiple varying complexity levels.

5.2 Acknowledgements

I gratefully acknowledge the Research Center of Digital Technologies at Alliander and the TU Delft, specifically the Human-Robot Interaction department within Cognitive Robotics for their welcoming attitude and providing the guidance and Franka Emika research 3 essential for the software development and experiments. Furthermore I would also extend my appreciation to Yke Bauke Eisma, Remco van Leeuwen and Karlijn van Overes for their guidance and insightful discussions throughout this study.

References

[1] Andrius Dzedzickis, Jurga Subačiūtė-Žemaitienė, Ernestas Šutinys, Urtė Samukaitė-Bubnienė, and Vytautas Bučinskas. Advanced applications of industrial robotics: New trends and possibilities. *Applied Sciences*, 12(1):135, 2021.

[2] Aude Billard and Danica Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446):eaat8414, 2019.

[3] Charles C Kemp, Aaron Edsinger, and Eduardo Torres-Jara. Challenges for robot manipulation in human environments [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):20–29, 2007.

[4] Alliander. About Alliander, Mar 2020. URL https://www.alliander.com/en/organisation/.

[5] Mohd Hafiidz Jaafar, Kadir Arifin, Kadaruddin Aiyub, Muhammad Rizal Razman, Muhammad Izzuddin Syakir Ishak, and Mohamad Shaharudin Samsurijan. Occupational safety and health management in the construction industry: a review. *International journal of occupational safety and ergonomics*, 24(4):493–506, 2018.

[6] Jean-Francois Allan and Julien Beaudry. Robotic systems applied to power substations-a state-of-the-art survey. In *Proceedings of the 2014 3rd International Conference on Applied Robotics for the Power Industry*, pages 1–6. IEEE, 2014.

[7] Yann Perrot, Laurent Gargiulo, Michael Houry, Nolwenn Kammerer, Delphine Keller, Yvan Measson, Gérard Piolain, and Alexandre Verney. Long-reach articulated robots for inspection and mini-invasive interventions in hazardous environments: Recent robotics research, qualification testing, and tool developments. *Journal of Field Robotics*, 29(1):175–185, 2012.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[10] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[11] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[12] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[13] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[14] Zirui Zhao, Wee Sun Lee, and David Hsu. Differentiable parsing and visual grounding of natural language instructions for object placement. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11546–11553. IEEE, 2023.

[15] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv:2210.03094*, 2(3):6, 2022.

[16] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR, 2023.

[17] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[18] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[19] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. Task and motion planning with large language models for object rearrangement. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2086–2092. IEEE, 2023.

[20] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[21] Shreyas Sundara Raman, Vanya Cohen, Eric Rosen, Ifrah Idrees, David Paulius, and Stefanie Tellex. Planning with large language models via corrective re-prompting. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[22] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

[23] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[24] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[25] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.

[26] Marta Skreta, Naruki Yoshikawa, Sebastian Arellano-Rubach, Zhi Ji, Lasse Bjørn Kristensen, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. Errors are useful prompts: Instruction guided task programming with verifier-assisted iterative prompting. *arXiv preprint arXiv:2303.14100*, 2023.

[27] Jiayi Pan, Glen Chou, and Dmitry Berenson. Data-efficient learning of natural language to linear temporal logic translators for robot task specification. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11554–11561. IEEE, 2023.

[28] Emily M Bender and Alexander Koller. Climbing towards nlu: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 5185–5198, 2020.

[29] Jin Chen, Zheng Liu, Xu Huang, Chenwang Wu, Qi Liu, Gangwei Jiang, Yuanhao Pu, Yuxuan Lei, Xiaolong Chen, Xingmei Wang, et al. When large language models meet personalization: Perspectives of challenges and opportunities. *World Wide Web*, 27(4):42, 2024.

[30] OpenAI. Openai, 2023. URL https://openai.com/. Accessed: 2023-12-19.

[31] Naoki Wake, Atsushi Kanehira, Kazuhiro Sasabuchi, Jun Takamatsu, and Katsushi Ikeuchi. Chatgpt empowered long-step robot control in various environments: A case application. *arXiv preprint arXiv:2304.03893*, 2023.

[32] Sayantan Adak, Daivik Agrawal, Animesh Mukherjee, and Somak Aditya. Grafford: A benchmark dataset for testing the knowledge of object affordances of language and vision models. *arXiv preprint arXiv:2402.12881*, 2024.

[33] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[34] OpenCV Team. Aruco marker detection, 2024. URL https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. Accessed: 2024-07-27.

[35] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, et al. Grounded sam: Assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*, 2024.

[36] MoveIt Contributors. Moveit: A robotics motion planning framework. https://moveit.ros.org/, 2024. Accessed: 2024-06-24.

[37] Python Software Foundation. *tkinter Python interface to Tcl/Tk*, 2024. URL https://docs.python.org/3/library/tkinter.html. Accessed: 2024-07-27.

[38] Sami Haddadin, Sven Parusel, Lars Johannsmeier, Saskia Golz, Simon Gabl, Florian Walch, Mohamadreza Sabaghian, Christoph Jähne, Lukas Hausperger, and Simon Haddadin. The franka emika robot: A reference platform for robotics research and education. *IEEE Robotics & Automation Magazine*, 29(2):46–64, 2022.

[39] Stereolabs. Zed 2: Advanced stereo vision camera. https://www.stereolabs.com/products/zed-2, 2024. Accessed: 2024-06-24.

[40] HP Inc. Hp zbook power g7 mobile workstation - product specifications, 2023. URL https://support.hp.com/us-en/document/c06931915. Accessed: 2024-07-27.

[41] Open Robotics. Noetic ros wiki, 2024. URL https://wiki.ros.org/noetic. Accessed: 2024-07-27.

[42] Franka Emika. libfranka: C++ library for franka emika research robots, 2024. URL https://frankaemika.github.io/libfranka/. Accessed: 2024-07-27.

[43] Franka Emika. franka_ros – ros wiki. https://wiki.ros.org/franka_ros, 2024. Accessed: 2024-07-27.

[44] Timo Birr, Christoph Pohl, Abdelrahman Younes, and Tamim Asfour. Autogpt+ p: Affordance-based task planning with large language models. *arXiv preprint arXiv:2402.10778*, 2024.

[45] Shizhe Diao, Pengcheng Wang, Yong Lin, and Tong Zhang. Active prompting with chain-of-thought for large language models. *arXiv preprint arXiv:2302.12246*, 2023.

[46] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[47] Songlin Bao, Tiantian Li, and Bin Cao. Chain-of-event prompting for multi-document summarization by large language models. *International Journal of Web Information Systems*, (ahead-of-print), 2024.

[48] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR, 2023.

[49] Christina Sarkisyan, Alexandr Korchemnyi, Alexey K Kovalev, and Aleksandr I Panov. Evaluation of pretrained large language models in embodied planning tasks. In *International Conference on Artificial General Intelligence*, pages 222–232. Springer, 2023.

[50] Rainer Kartmann, Danqing Liu, and Tamim Asfour. Semantic scene manipulation based on 3d spatial object relations and language instructions. In *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pages 306–313. IEEE, 2021.

[51] Rainer Kartmann and Tamim Asfour. Interactive and incremental learning of spatial object relations from human demonstrations. *Frontiers in Robotics and AI*, 10:1151303, 2023.

[52] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45, 2024.

[53] Irena Gao, Gabriel Ilharco, Scott Lundberg, and Marco Tulio Ribeiro. Adaptive testing of computer vision models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4003–4014, 2023.

[54] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

[55] Carlos Olea, Holly Tucker, Jessica Phelan, Cameron Pattison, Shen Zhang, Maxwell Lieb, and J White. Evaluating persona prompting for question answering tasks. In *Proceedings of th e 10th international conference on artificial intelligence and soft computing, Sydney, Australia*, 2024.

[56] Joni Salminen, Chang Liu, Wenjing Pian, Jianxing Chi, Essi Häyhänen, and Bernard J Jansen. Deus ex machina and personas from large language models: Investigating the composition of ai-generated persona descriptions. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–20, 2024.

[57] Lisa P Argyle, Ethan C Busby, Nancy Fulda, Joshua R Gubler, Christopher Rytting, and David Wingate. Out of one, many: Using language models to simulate human samples. *Political Analysis*, 31(3):337–351, 2023.

[58] Joost CF de Winter, Tom Driessen, and Dimitra Dodou. The use of chatgpt for personality research: Administering questionnaires using generated personas. *Personality and Individual Differences*, 228:112729, 2024.

[59] Jinke D Van Der Laan, Adriaan Heino, and Dick De Waard. A simple procedure for the assessment of acceptance of advanced transport telematics. *Transportation Research Part C: Emerging Technologies*, 5(1):1–10, 1997.

[60] Saad Khan and Simon Parkinson. *Review into State of the Art of Vulnerability Assessment using Artificial Intelligence*, pages 3–32. Springer International Publishing, Cham, 2018. ISBN 978-3-319-92624-7. doi: 10.1007/978-3-319-92624-7_1. URL https://doi.org/10.1007/978-3-319-92624-7_1.

[61] Donald T. Campbell. Methods for the experimenting society. *Evaluation Practice*, 12(3):223–260, 1991. ISSN 0886-1633. doi: https://doi.org/10.1016/0886-1633(91)90039-Z. URL https://www.sciencedirect.com/science/article/pii/088616339190039Z.

[62] Andrew M Olney. Generating multiple choice questions from a textbook: Llms match human performance on most metrics. In *LLM@ AIED*, pages 111–128, 2023.

[63] Haocong Rao, Cyril Leung, and Chunyan Miao. Can chatgpt assess human personalities? a general evaluation framework. *arXiv preprint arXiv:2303.01248*, 2023.

[64] Beatrice Rammstedt and Oliver P John. Measuring personality in one minute or less: A 10-item short version of the big five inventory in english and german. *Journal of research in Personality*, 41(1):203–212, 2007.

[65] Emeli Adell, Lena Nilsson, and András Várhelyi. How is acceptance measured? overview of measurement issues, methods and tools. In *Driver acceptance of new technology*, pages 73–88. CRC Press, 2018.

[66] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints*, 2023.

[67] Shubham Vatsal and Harsh Dubey. A survey of prompt engineering methods in large language models for different nlp tasks. *arXiv preprint arXiv:2407.12994*, 2024.

[68] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.

[69] Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. Delta: Decomposed efficient long-term robot task planning using large language models. *arXiv preprint arXiv:2404.03275*, 2024.

[70] Zhirong Luan, Yujun Lai, Rundong Huang, Shuanghao Bai, Yuedi Zhang, Haoran Zhang, and Qian Wang. Enhancing robot task planning and execution through multi-layer large language models. *Sensors*, 24(5):1687, 2024.

[71] Gaël Gendron, Qiming Bao, Michael Witbrock, and Gillian Dobbie. Large language models are not strong abstract reasoners. *arXiv preprint arXiv:2305.19555*, 2023.

[72] Simon Ott, Konstantin Hebenstreit, Valentin Liévin, Christoffer Egeberg Hother, Milad Moradi, Maximilian Mayrhauser, Robert Praas, Ole Winther, and Matthias Samwald. Thoughtsource: A central hub for large language model reasoning data. *Scientific data*, 10(1):528, 2023.

[73] Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Yue Zhang, Neil Zhenqiang Gong, et al. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv preprint arXiv:2306.04528*, 2023.

[74] Bogdan Moldovan, Plinio Moreno, Davide Nitti, José Santos-Victor, and Luc De Raedt. Relational affordances for multiple-object manipulation. *Autonomous Robots*, 42:19–44, 2018.

[75] Natsuki Yamanobe, Weiwei Wan, Ixchel G Ramirez-Alpizar, Damien Petit, Tokuo Tsuji, Shuichi Akizuki, Manabu Hashimoto, Kazuyuki Nagata, and Kensuke Harada. A brief review of affordance in robotic manipulation research. *Advanced Robotics*, 31 (19-20):1086–1101, 2017.

[76] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[77] Ruizhen Hu, Zihao Yan, Jingwen Zhang, Oliver Van Kaick, Ariel Shamir, Hao Zhang, and Hui Huang. Predictive and generative neural networks for object functionality. *arXiv preprint arXiv:2006.15520*, 2020.

[78] Spyridon Thermos, Petros Daras, and Gerasimos Potamianos. A deep learning approach to object affordance segmentation. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2358–2362. IEEE, 2020.

[79] Séverin Lemaignan, Mathieu Warnier, E Akin Sisbot, Aurélie Clodic, and Rachid Alami. Artificial cognition for social human–robot interaction: An implementation. *Artificial Intelligence*, 247:45–69, 2017.

[80] Toshali Goel, Orit Shaer, Catherine Delcourt, Quan Gu, and Angel Cooper. Preparing future designers for human-ai collaboration in persona creation. In *Proceedings of the 2nd Annual Meeting of the Symposium on Human-Computer Interaction for Work*, pages 1–14, 2023.

[81] Karim Benharrak, Tim Zindulka, Florian Lehmann, Hendrik Heuer, and Daniel Buschek. Writer-defined ai personas for on-demand feedback generation. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2024.

[82] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2901–2910, 2019.

[83] Matteo Saveriano, Fares J Abu-Dakka, Aljaž Kramberger, and Luka Peternel. Dynamic movement primitives in robotics: A tutorial survey. *The International Journal of Robotics Research*, 42(13):1133–1184, 2023.

[84] Xiang Li, Congcong Wen, Yuan Hu, Zhenghang Yuan, and Xiao Xiang Zhu. Vision-language models in remote sensing: Current progress and future trends. *IEEE Geoscience and Remote Sensing Magazine*, 2024.

[85] Mingxiao Huo, Pengliang Ji, Haotian Lin, Junchen Liu, Yixiao Wang, and Yijun Chen. Composition vision-language understanding via segment and depth anything model. *arXiv preprint arXiv:2406.18591*, 2024.

[86] Hanxu Hu, Hongyuan Lu, Huajian Zhang, Yun-Ze Song, Wai Lam, and Yue Zhang. Chain-of-symbol prompting elicits planning in large langauge models. *arXiv preprint arXiv:2305.10276*, 2023.

[87] Roshanak Mirzaee and Parisa Kordjamshidi. Transfer learning with synthetic corpora for spatial role labeling and reasoning. *arXiv preprint arXiv:2210.16952*, 2022.

[88] Shengyi Qian, Weifeng Chen, Min Bai, Xiong Zhou, Zhuowen Tu, and Li Erran Li. Affordancellm: Grounding affordance from vision language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7587–7597, 2024.

[89] Xinbo Yu, Peisen Liu, Wei He, Yu Liu, Qi Chen, and Liang Ding. Human-robot variable impedance skills transfer learning based on dynamic movement primitives. *IEEE Robotics and Automation Letters*, 7(3):6463–6470, 2022.

[90] Richard A Armstrong, Frank Eperjesi, and Bernard Gilmartin. The application of analysis of variance (anova) to different experimental designs in optometry. *Ophthalmic and Physiological Optics*, 22(3):248–256, 2002.

[91] Hae-Young Kim. Statistical notes for clinical researchers: Two-way analysis of variance (anova)-exploring possible interaction between factors. *Restorative dentistry & endodontics*, 39(2):143–147, 2014.

6 Appendix A

6.1 System Prompts

The following prompt templates are inspired by the research of Wake et al. [31]. These prompts define the process by which LERACS generates its output, as detailed in *section 6.1.1, 6.1.2 and 6.1.3*. During each instance of task planning with Chat-GPT, prompts one to five are preloaded from prepared text files, while certain components of the prompts (**[INSTRUCTION]** and **[ENVIRONMENT]**) are dynamically generated based on user instructions, feedback, and environmental data.

6.1.1 Initialization prompt

In these initial prompts, the role of ChatGPT in task planning is explained. To manage multiple prompts, we instruct ChatGPT to wait for subsequent prompts until all have been provided.

System

```
You are an excellent interpreter of human instructions
for general tasks. Given an instruction and information
about the working environment, you break it down into a
sequence of robotic actions.
```

.

Prompt Role

```
[user]
You are an excellent interpreter of human instructions
for general tasks. Given an instruction and
information about the working environment,
you break it down into a sequence of robotic actions.
Please do not begin working until I say "Start working."
Instead, simply output the message "Waiting for next input."
     Understood?

[assistant]
Understood. Waiting for next input.
```

.

6.1.2 Generate environment prompts

These prompts define the rules for representing working environments that the arm can function in. All physical entities are classified into non-manipulable obstacles, referred to as **assets**, such as tables and other surfaces, and manipulable objects, referred to as **objects**, such as blocks and cases. These two classes are defined to differentiate between the entities that may be manipulated and those that cannot. As a hint for task planning, the spatial relationships between entities are described as states and can describe **object states**, which are chosen from a "**STATE LIST**".

Input Environment

```
[user]
Information about environments and objects are given
from a camera input (as a snapshot). Chat-GPT needs
to describe the environment as list of strings. This
will be how it will be described for the generate
function. It needs to be in the following format:
"""
     environment = {{
          "assets": ["<asset1>","<asset2>"],
          "asset_states": {{"<asset1>": "<state>",
                        "<asset2>": "<state>"}},
          "objects": ["<object1>", "<object2>"],
          "object_states": {{"<object1>": "<state>",
                         "<object2>": "<state>"}}
}}
"""
Asset states and object states are represented using
those state sets:
"""
"STATE LIST"
- on_something(<something>): Object is located on
  <something>
- inside_something(<something>): Object is located
  inside <something>
- inside_hand(): Object is being grasped by a robot
  hand
- closed(): Object can be opened
- open(): Object can be closed or kept opened
"""
<something> should be one of the assets or objects in
the environment.
The texts above are part of the overall instruction.
Do not start working yet:
[assistant]
Understood. I will wait for further instructions before
starting to work.
```

.

Prompt Environment

```
[user]
For example, if there is a table and a floor, with
objects labeled with aruco markers on the table, the
aruco marker number needs to be recognized and
connected to the object. An example of the description
should be as follows:
"""
"environment":{{"assets":["<table>", "<floor>"],
"asset_states": {{"<table>": "on_something(<floor>)"}},
"objects": ["<red_cube(marker_id:"0")>",
           "<red_case(marker_id:"2")>",
           "<blue_case(marker_id:"4")>",
           "<yellow_rectangle(marker_id:"5")>"],
"object_states": {{"<red_cube(marker_id:"1")>":"on_something(<
                  table>)",
                 "<blue_cube(marker_id:"3")>":"on_something(<table
                      >)",
                 "<yellow_rectangle(marker_id:"5")>":"on_something
                      (<table>)",
                 "<red_case(marker_id:"2")>":"on_something(<table
                      >)",
                 "<blue_case(marker_id:"4")>":"on_something(<table
                      >)"
}}
}}
"""

"""
There should be nothing else as an output except for
this output format.
The texts above are part of the overall instruction.
Do not start working yet:
[assistant]
Understood. I will wait for further instructions before
starting to work.
```

.

6.1.3 Task decomposition prompts

In these prompts, a set of robot actions is defined in the "**ROBOT ACTION LIST**" tailored to the specific application and implementation of the the motion primitives for a Franka Emika robot arm and specifics for the **rules** and ChatGPT's **output format**. The robot actions can be customizes accordingly for each different application. To ensure integration with the other control and vision systems, it is instructed to generate a **Python dictionary** that can be saved as a JSON file. Moreover, examples of the expected inputs and outputs were provided to minimize the need for the user to use the feedback in the chat interface.

Prompt Function

```
[user]
Necessary and sufficient robot actions are defined as follows:
"""
"ROBOT ACTION LIST"

move_camera_state_low(): Move the robot camera to the low camera
      state with adjustable acceleration and velocity.
move_to_marker(marker_id): Move the robot end-effector to the
      ArUco marker specified by marker_id. The input of this
      function is the designated marker ID from the object that the
       end-effector is moving to (string: "4", "0", etc).
pick(marker_id): Pick an object with an ArUco marker specified by
      marker_id. The input of this function is the designated
      marker ID from the object to be picked (string: "4", "0", etc
      ).
place(marker_id): Place an object with an ArUco marker specified
      by marker_id. The input of this function is the designated
      marker ID from the object to be placed (string: "4", "0", etc
      ).
push(marker_id, direction): Push an object with an ArUco marker
      specified by marker_id to a certain direction with 5
      centimeter. The input of this function is the designated
      marker ID from the pushed object (string) and the direction
      in which the object is pushed (string: "forward", "backward",
       "left", "right").
grasp(command): Open or close the gripper. The input of this
      function is the command for the grasp client (string: "open",
       "close").
"""
The texts above are part of the overall instruction. Do not start
      working yet:
[assistant]
Understood. Waiting for next input.
.
```

Prompt Output Format

```
[user]
You divide the actions given in the text into detailed robot
      actions and put them together as a python dictionary.
The dictionary has five keys.
"""

dictionary["task_cohesion"]: A dictionary containing information
      about the robot's actions that have been split up.
dictionary["environment_before"]: The state of the environment
      before the manipulation.
dictionary["environment_after"]: The state of the environment
      after the manipulation.
dictionary["instruction_summary"]: contains a brief summary of the
       given sentence.
dictionary["question"]: If you cannot understand the given
      sentence, you can ask the user to rephrase the sentence.
      Leave this key empty if you can understand the given sentence
      .
"""
Three keys exist in dictionary["task_cohesion"].
"""
dictionary["task_cohesion"]["task_sequence"]: Contains a list of
      robot actions. Only the behaviors defined in the "ROBOT
      ACTION LIST" will be used.
dictionary["task_cohesion"]["step_instructions"]: contains a list
      of instructions corresponding to dictionary["task_cohesion
      "]["task_sequence"].
dictionary["task_cohesion"]["object_name"]: The name of the
      manipulated object. Only objects defined in the input
      dictionary will be used for the object name.
"""
The texts above are part of the overall instruction. Do not start
      working yet:
[assistant]
Understood. Waiting for next input.
.
```

Prompt Example

```
[user]
[user]
I will give you some examples of the input and the output you will
       generate.

Example 1:
"""
- Input:
{"assets":["<table>", "<floor>"],
"asset_states": {"<table>": "on_something(<floor>)"},
"objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(marker_id
      :\"3\")>", "<red_case(marker_id:\"2\")>", "<blue_case(
      marker_id:\"4\")>"],
"object_states": {"<red_cube(marker_id:\"1\")>": "on_something(<
      table>)",
                "<blue_cube(marker_id:\"3\")>": "on_something(<
                     table>)",
                "<red_case(marker_id:\"2\")>": "on_something(<
                     table>)",
                "<blue_case(marker_id:\"4\")>": "on_something(<
                     table>)"
},
"instruction": "Put the cube in the case of the same color"}
- Output:
'''
{
    "task_cohesion": {
        "task_sequence": [
            "all_update()",
            "init_constraints()",
            "move_to_marker(\"1\")",
            "grasp(\"open\")",
            "pick(\"0\")",
            "move_to_marker(\"2\")",
            "place(\"2\")",
            "move_camera_state_low()"
        ],
        "step_instructions": [
            "update the marker and world positions.",
            "initialize constraints.",
            "move the end-effector to the object.",
            "open the gripper.",
            "pick the object by grasping with the end-effector.",
            "move the end-effector to the object",
            "place the object by releasing with the end-effector.",
            "move the end-effector to the low camera position."
        ],
        "object_name": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
              marker_id:\"3\")>", "<red_case(marker_id:\"2\")>", "<
              blue_case(marker_id:\"4\")>"]
    },
    "environment_before": {
        "assets": ["<table>", "<floor>"],
        "asset_states": {
            "<table>": "on_something(<floor>)"
        },
        "objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
              marker_id:\"3\")>", "<red_case(marker_id:\"2\")>", "<
              blue_case(marker_id:\"4\")>"],
        "object_states": {"<red_cube(marker_id:\"1\")>": "
              on_something(<table>)",
                          "<blue_cube(marker_id:\"3\")>": "
                                on_something(<table>)",
                          "<red_case(marker_id:\"2\")>": "
                                on_something(<table>)",
                          "<blue_case(marker_id:\"4\")>": "
                                on_something(<table>)"
        }
    },
    "environment_after": {
```

```
        "assets": ["<table>", "<floor>"],
        "asset_states": {
            "<table>": "on_something(<floor>)"
        },
        "objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
              marker_id:\"3\")>", "<red_case(marker_id:\"2\")>", "<
              blue_case(marker_id:\"4\")>"],
        "object_states": {"<red_cube(marker_id:\"1\")>": "
              on_something(<red_case(marker_id:\"2\")>)",
                          "<blue_cube(marker_id:\"3\")>": "
                                on_something(<table>)",
                          "<red_case(marker_id:\"2\")>": "
                                on_something(<table>)",
                          "<blue_case(marker_id:\"4\")>": "
                                on_something(<table>)"
        }
    },
    "instruction_summary": "put the cube in the case of the same
          color",
    "question": ""
}
‘‘‘

"""
Example 2:
"""
- Input:
{"assets":["<table>", "<floor>"],
"asset_states": {"<table>": "on_something(<floor>)"},
"objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(marker_id
      :\"3\")>", "<red_case(marker_id:\"2\")>", "<blue_case(
      marker_id:\"4\")>"],
"object_states": {"<red_cube(marker_id:\"1\")>": "on_something(<
      table>)",
                  "<blue_cube(marker_id:\"3\")>": "on_something(<
                        table>)",
                  "<red_case(marker_id:\"2\")>": "on_something(<
                        table>)",
                  "<blue_case(marker_id:\"4\")>": "on_something(<
                        table>)"
},
"instruction": "Push the red cube twice to the right"}
- Output:
‘‘‘
{
    "task_cohesion": {
        "task_sequence": [
            "all_update()",
            "init_constraints()",
            "self.move_to_marker(\"1\")",
            "grasp(\"open\")",
            "push(\"1\", [\"right\", \"right\"])",
            "move_camera_state_low()"
        ],
        "step_instructions": [
            "update the marker and world positions.",
            "initialize constraints.",
            "move the end-effector to the object.",
            "open the gripper.",
            "Push the object with the end-effector.",
            "move the end-effector to the low camera position."
        ],
        "object_name": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
              marker_id:\"3\")>", "<red_case(marker_id:\"2\")>", "<
              blue_case(marker_id:\"4\")>"]
    },
    "environment_before": {
        "assets": ["<table>", "<floor>"],
        "asset_states": {
            "<table>": "on_something(<floor>)"
        },
        "objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
              marker_id:\"3\")>", "<red_case(marker_id:\"2\")>", "<
              blue_case(marker_id:\"4\")>"],
        "object_states": {"<red_cube(marker_id:\"1\")>": "
              on_something(<table>)",
                          "<blue_cube(marker_id:\"3\")>": "
                                on_something(<table>)",
                          "<red_case(marker_id:\"2\")>": "
                                on_something(<table>)",
                          "<blue_case(marker_id:\"4\")>": "
                                on_something(<table>)"
        }
    },
    "environment_after": {
        "assets": ["<table>", "<floor>"],
        "asset_states": {
            "<table>": "on_something(<floor>)"
        },
        "objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
              marker_id:\"3\")>", "<red_case(marker_id:\"2\")>", "<
              blue_case(marker_id:\"4\")>"],
        "object_states": {"<red_cube(marker_id:\"1\")>": "
              inside_hand()",
                          "<blue_cube(marker_id:\"3\")>": "
                                on_something(<table>)",
                          "<red_case(marker_id:\"2\")>": "
                                on_something(<table>)",
                          "<blue_case(marker_id:\"4\")>": "
                                on_something(<table>)"
        }
    },
    "instruction_summary": "push the cube twice to the right",
    "question": ""
}

‘‘‘

"""
Example 3:
"""
- Input:
{"assets":["<table>", "<floor>"],
"asset_states": {"<table>": "on_something(<floor>)"},
"objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(marker_id
      :\"3\")>", "<yellow_rectangle(marker_id:\"5\")>", "<
      green_rectangle(marker_id:\"6\")>", "<red_case(marker_id
      :\"2\")>", "<blue_case(marker_id:\"4\")>"],
"object_states": {"<red_cube(marker_id:\"1\")>": "on_something(<
      table>)",
                  "<blue_cube(marker_id:\"3\")>": "on_something(<
                        table>)",
                  "<yellow_rectangle(marker_id:\"5\")>": "
                        on_something(<table>)",
                  "<green_rectangle(marker_id:\"6\")>": "
                        on_something(<table>)",
                  "<red_case(marker_id:\"2\")>": "on_something(<
                        table>)",
                  "<blue_case(marker_id:\"4\")>": "on_something(<
                        table>)"
},
"instruction": "Sort a rectangle in a case and a cube in the other
      case"}
- Output:
‘‘‘
{
    "task_cohesion": {
        "task_sequence": [
            "all_update()",
            "init_constraints()",
            "move_to_marker(\"5\")",
            "grasp(\"open\")",
            "pick(\"5\")",
            "move_to_marker(\"2\")",
            "place(\"2\")",
            "move_to_marker(\"1\")",
            "grasp(\"open\")",
            "pick(\"1\")",
            "move_to_marker(\"4\")",
            "place(\"4\")",
            "move_camera_state_low()"
        ],
        "step_instructions": [
            "update the marker and world positions.",
            "initialize constraints.",
            "move the end-effector to the object.",
            "open the gripper.",
            "pick the object by grasping with the end-effector.",
            "move the end-effector to the object",
            "place the object by releasing with the end-effector.",
            "move the end-effector to the object.",
            "open the gripper.",
            "pick the object by grasping with the end-effector.",
            "move the end-effector to the object",
            "place the object by releasing with the end-effector.",
            "move the end-effector to the low camera position."
        ],
        "object_name": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
              marker_id:\"3\")>", "<yellow_rectangle(marker_id
              :\"5\")>", "<green_rectangle(marker_id:\"6\")>", "<
              red_case(marker_id:\"2\")>", "<blue_case(marker_id
              :\"4\")>"]
    },
    "environment_before": {
        "assets": ["<table>", "<floor>"],
        "asset_states": {
            "<table>": "on_something(<floor>)"
        },
        "objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
              marker_id:\"3\")>", "<yellow_rectangle(marker_id
              :\"5\")>", "<green_rectangle(marker_id:\"6\")>", "<
              red_case(marker_id:\"2\")>", "<blue_case(marker_id
```

```
            :\"4\")>"],
        "object_states": {"<red_cube(marker_id:\"1\")>": "
            on_something(<table>)",
                        "<blue_cube(marker_id:\"3\")>": "
                            on_something(<table>)",
                        "<yellow_rectangle(marker_id:\"5\")>": "
                            on_something(<table>)",
                        "<green_rectangle(marker_id:\"6\")>": "
                            on_something(<table>)",
                        "<red_case(marker_id:\"2\")>": "
                            on_something(<table>)",
                        "<blue_case(marker_id:\"4\")>": "
                            on_something(<table>)"
        }
    },
    "environment_after": {
        "assets": ["<table>", "<floor>"],
        "asset_states": {
            "<table>": "on_something(<floor>)"
        },
        "objects": ["<red_cube(marker_id:\"1\")>", "<blue_cube(
            marker_id:\"3\")>", "<yellow_rectangle(marker_id
            :\"5\")>", "<green_rectangle(marker_id:\"6\")>", "<
            red_case(marker_id:\"2\")>", "<blue_case(marker_id
            :\"4\")>"],
        "object_states": {"<red_cube(marker_id:\"1\")>": "
            on_something(<blue_case(marker_id:\"4\")>)",
                        "<blue_cube(marker_id:\"3\")>": "
                            on_something(<table>)",
                        "<yellow_rectangle(marker_id:\"5\")>": "
                            on_something(<red_case(marker_id
                            :\"2\")>)",
                        "<green_rectangle(marker_id:\"6\")>": "
                            on_something(<table>)",
                        "<red_case(marker_id:\"2\")>": "
                            on_something(<table>)",
                        "<blue_case(marker_id:\"4\")>": "
                            on_something(<table>)"
        }
    },
    "instruction_summary": "Sort a rectangle in a case and a cube
         in the other case",
    "question": ""
}
"""


From these examples, learn that some robotic actions have
     dependencies with the actions before and after them.
-------------------------------------------------------
The texts above are part of the overall instruction. Do not start
     working yet:

[assistant]
Understood. I will wait for further instructions before starting
     to work.
.
```

Query

```
[user]
Start working. Resume from the environment below.

"""
{"environment":[ENVIRONMENT]}
"""
The instruction is as follows:
"""
{"instruction": [INSTRUCTION]}
"""

The dictionary that you return should be formatted as python
     dictionary. Follow these rules:
1. The first two elements should be all_update() and
     init_constraints() (in this order) to initialize the robot.
2. Make sure that each element of the ["step_instructions"]
     explains corresponding element of the ["task_sequence"].
     Refer to the "ROBOT ACTION LIST" to understand the elements
     of ["task_sequence"].
3, The length of the ["step_instructions"] list must be the same
     as the length of the ["task_sequence"] list.
4. Never left ',' at the end of the list.
5. Keep track of all items listed in the "objects" section of the
     "environment_before" field. Please ensure that you fill out
     both the "objects" and "object_states" sections for all
     listed items.
6. Use the "STATE LIST" for the "object states" field.

"""
"STATE LIST"
- on_something(<something>): Object is located on <something>
- inside_hand(): Object is being grasped by a robot hand
"""

7. All keys of the dictionary should be double-quoted.
8. Insert ``` at the beginning and the end of the dictionary to
     separate it from the rest of your response.
9. Make sure that you output a consistent manipultation as a
     single arm robot. For example, grasping an object should not
     occur in successive steps.
Adhere to the output format I defined above. Follow the nine rules
     . Think step by step.
.
```

6.2 Experiment Prompts

These prompts define how the AI user experiment is performed and how the AI persona are generated as show in *section 6.2.1*. An example conversation of an AI user performing one scenario is shown in *section 6.2.2*.

6.2.1 Generate Personas

Persona

```
Please give me {num_personas} realistic and diverse personas with
realistic names and realistic personality descriptions.
Also add their age and gender. The persona should be described by
     means
of three brief sentences separated by semicolons.
Report each persona on a single line, numbered 0001 to 0025.
Separate age, gender, profession/activity/job versus description
     of who
they are by means of a dash.

These personas are going to test a robotic system using AI,
where the persona has to chat with the robot.
Make sure that from the {num_personas} personas that there are
5 saboteurs, 5 cognitively limited persons (for example a child),
5 technician experts, 5 technician newcomers and 5 normally
generated personas.
Say behind every name what they are from these 5 categories.
Only personas; nothing else.
```

Persona personality BFI-10

```
Imagine the following person (age, gender, description): {
     description}
Rate this person / complete the questionnaire for this person, on
     a scale
of 1 (disagree strongly) to 5 (agree strongly).
Report in the following format on a single line, e.g., 5 1 4 ...
Report only digits, nothing else.

gets nervous easily.
tends to find fault with others.
is outgoing, sociable.
is generally trusting.
tends to be lazy.
is relaxed, handles stress well.
has few artistic interests.
does a thorough job.
is reserved.
has an active imagination.
.
```

.

6.2.2 Persona Experiment

SYSTEM

Hello ChatGPT, you are the following persona:

```
  "0015": {
    "name": "Michael Evans",
    "type": "Normally Generated Persona",
    "age": 36,
    "gender": "Male",
    "occupation": "Project Manager",
    "description": "Focused on project timelines and deliverables;
    values AI that can assist with project tracking;
    practical and result-driven."
  },
```

With BFI-10 test results:

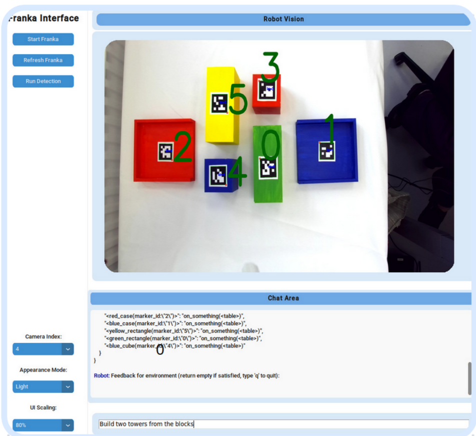
```
    "0015": [
        {
            "response": "2 4 2 3 2 3 2 5 1 4",
            "questions_order": [
                "is reserved.",
                "is generally trusting.",
                "is relaxed, handles stress well.",
                "gets nervous easily.",
                "has an active imagination.",
                "tends to find fault with others.",
                "is outgoing, sociable.",
                "does a thorough job.",
                "tends to be lazy.",
                "has few artistic interests."
            ]
        }
    ],
```

Stay in this role the whole time.
Say nothing if you understand, wait for the next input.

ChatGPT

Understood

SYSTEM



You are going to test a system on a robotic arm using a computer and a chat interface. You have to fill in the questions to check the system and write a textual description with the goal that the robot needs to hand you one of the blocks. I will show you the info that is given to you and at last the question you need to answer.

Before you begin your session with the system, we'd like you to rate your initial expectations using the Van der Laan Acceptance Scale.

Rate each statement from -2 to +2, where -2 is strongly disagree, 0 is neutral, and +2 is strongly agree. Each attribute has a directly opposite attribute to help guide your scoring.

No. Question. Attribute Scale. Opposite Attribute.
1. I think the system is useful. Useful -2 to +2 Useless
2. The system is pleasant to use. Pleasant -2 to +2 Unpleasant
3. I think the system is bad. Bad +2 to -2 Good
4. I think the system is nice. Nice -2 to +2 Annoying
5. I think the system is effective. Effective -2 to +2 Superfluous
6. I think the system is irritating. Irritating +2 to -2 Likeable
7. I think the system is assisting. Assisting -2 to +2 Worthless
8. I think the system is undesirable. Undesirable +2 to -2 Desirable
9. The system raises my alertness. Raising Alertness -2 to +2 Sleep-inducing

Please complete the scale now, and then wait for the next input.

ChatGPT

Here are my initial ratings using the Van der Laan Acceptance Scale for the robotic arm and chat system:

```
    Useful: +2
    Pleasant: +1
    Bad: -2
    Nice: +1
    Effective: +2
    Irritating: -2
    Assisting: +2
    Undesirable: -2
    Raising Alertness: +1
```

I am ready for the next instructions.

SYSTEM

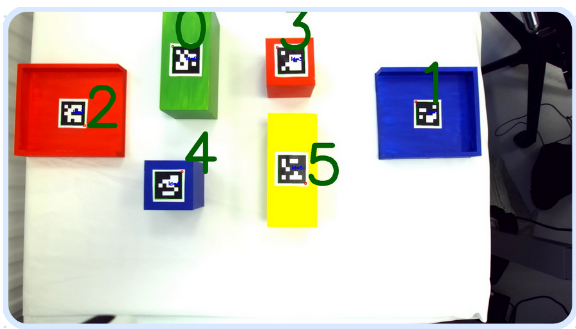
You are now going fill in the questions to check the system and write a textual description with the goal that the robot needs to hand you one of the blocks. I will now show you the info that is given to you and at last the question you need to answer.

Say nothing if you understand, wait for the next input.

ChatGPT

Understood. Waiting for the next input.

SYSTEM



Received environment description:

```
"""
environment": {
    "assets": ["<table>", "<floor>"],

    "asset_states": {"<table>": "on_something(<floor>)"},

    "objects": [
        "<red_case(marker_id:\"2\")>",
        "<blue_case(marker_id:\"1\")>",
        "<yellow_rectangle(marker_id:\"5\")>",
```

```
        "<green_rectangle(marker_id:\"0\")>",
        "<red_cube(marker_id:\"3\")>",
        "<blue_cube(marker_id:\"4\")>"],

    "object_states": {
        "<red_case(marker_id:\"2\")>": "on_something(<table>)",
        "<blue_case(marker_id:\"1\")>": "on_something(<table>)",
        "<yellow_rectangle(marker_id:\"5\")>": "on_something(<table
             >)",
        "<green_rectangle(marker_id:\"0\")>": "on_something(<table
             >)",
        "<red_cube(marker_id:\"3\")>": "on_something(<table>)",
        "<blue_cube(marker_id:\"4\")>": "on_something(<table>)"}}
"""

Is this generated environment correct with what you can see
in the image? Do you have feedback for the environment
(say yes if satisfied)?
```

ChatGPT

```
Yes
```

SYSTEM

```
Environment approved. Enter task instructions.
```

ChatGPT

```
Please hand me the red cube with marker ID "3".
```

SYSTEM

```
Received response: '''python
{
    "task_cohesion": {
        "task_sequence": [
            "all_update()",
            "init_constraints()",
            "move_to_marker(\"3\")",
            "grasp(\"open\")",
            "pick(\"3\")",
            "move_camera_state_low()"],

        "step_instructions": [
            "update the marker and world positions.",
            "initialize constraints.",
            "move the end-effector to the object.",
            "open the gripper.",
            "pick the object by grasping with the end-effector.",
            "move the end-effector to the low camera position."],

        "object_name": ["<red_case(marker_id:\"2\")>",
                        "<blue_case(marker_id:\"1\")>",
                        "<yellow_rectangle(marker_id:\"5\")>",
                        "<green_rectangle(marker_id:\"0\")>",
                        "<red_cube(marker_id:\"3\")>",
                        "<blue_cube(marker_id:\"4\")>"]},

    "environment_before": {
        "assets": ["<table>", "<floor>"],

        "asset_states": {"<table>": "on_something(<floor>)"},

        "objects": ["<red_case(marker_id:\"2\")>",
                    "<blue_case(marker_id:\"1\")>",
                    "<yellow_rectangle(marker_id:\"5\")>",
                    "<green_rectangle(marker_id:\"0\")>",
                    "<red_cube(marker_id:\"3\")>",
                    "<blue_cube(marker_id:\"4\")>"],

        "object_states": {
            "<red_case(marker_id:\"2\")>":"on_something(<table>)",
            "<blue_case(marker_id\"1\")>":"on_something(<table>)",
            "<yellow_rectangle(marker_id\"5\")>":"on_something(<
                 table>)",
            "<green_rectangle(marker_id:\"0\")>":"on_something(<
                 table>)",
            "<red_cube(marker_id:\"3\")>":"on_something(<table>)",
            "<blue_cube(marker_id:\"4\")>":"on_something(<table>)
                 "}},

    "environment_after": {
        "assets": ["<table>", "<floor>"],
        "asset_states": {"<table>": "on_something(<floor>)"},

        "objects": ["<red_case(marker_id:\"2\")>",
                    "<blue_case(marker_id:\"1\")>",
                    "<yellow_rectangle(marker_id:\"5\")>",
                    "<green_rectangle(marker_id:\"0\")>",
                    "<red_cube(marker_id:\"3\")>",
                    "<blue_cube(marker_id:\"4\")>"],

        "object_states": {
            "<red_case(marker_id:\"2\")>":"on_something(<table>)",
            "<blue_case(marker_id:\"1\")>":"on_something(<table>)",
            "<yellow_rectangle(marker_id:\"5\")>":"on_something(<
                 table>)",
            "<green_rectangle(marker_id:\"0\")>":"on_something(<
                 table>)",
            "<red_cube(marker_id:\"3\")>":"inside_hand()",
            "<blue_cube(marker_id:\"4\")>":"on_something(<table>)
                 "}},

    "instruction_summary": "hand the red cube with marker ID 3",
    "question": ""
}
'''

Do you think this is correct? Provide feedback for the
     instructions,
or say yes to approve.
```

ChatGPT

```
Yes
```

SYSTEM

```
Task completed succesfully.

What do you think of the system after interacting with it, we'd
     like you
to rate your initial expectations using the Van der Laan
Acceptance Scale.

Rate each statement from -2 to +2, where -2 is strongly disagree,
0 is neutral, and +2 is strongly agree. Each attribute has a
     directly
opposite attribute to help guide your scoring.

No. Question. Attribute Scale. Opposite Attribute.
1. I think the system (robotic arm and chat) is useful. Useful -2
     to +2 Useless
2. The system is pleasant to use. Pleasant -2 to +2 Unpleasant
3. I think the system is bad. Bad +2 to -2 Good
4. I think the system is nice. Nice -2 to +2 Annoying
5. I think the system is effective. Effective -2 to +2 Superfluous
6. I think the system is irritating. Irritating +2 to -2 Likeable
7. I think the system is assisting. Assisting -2 to +2 Worthless
8. I think the system is undesirable. Undesirable +2 to -2
     Desirable
9. The system raises my alertness. Raising Alertness -2 to +2
     Sleep-
inducing

Please complete the scale now.
```

SYSTEM

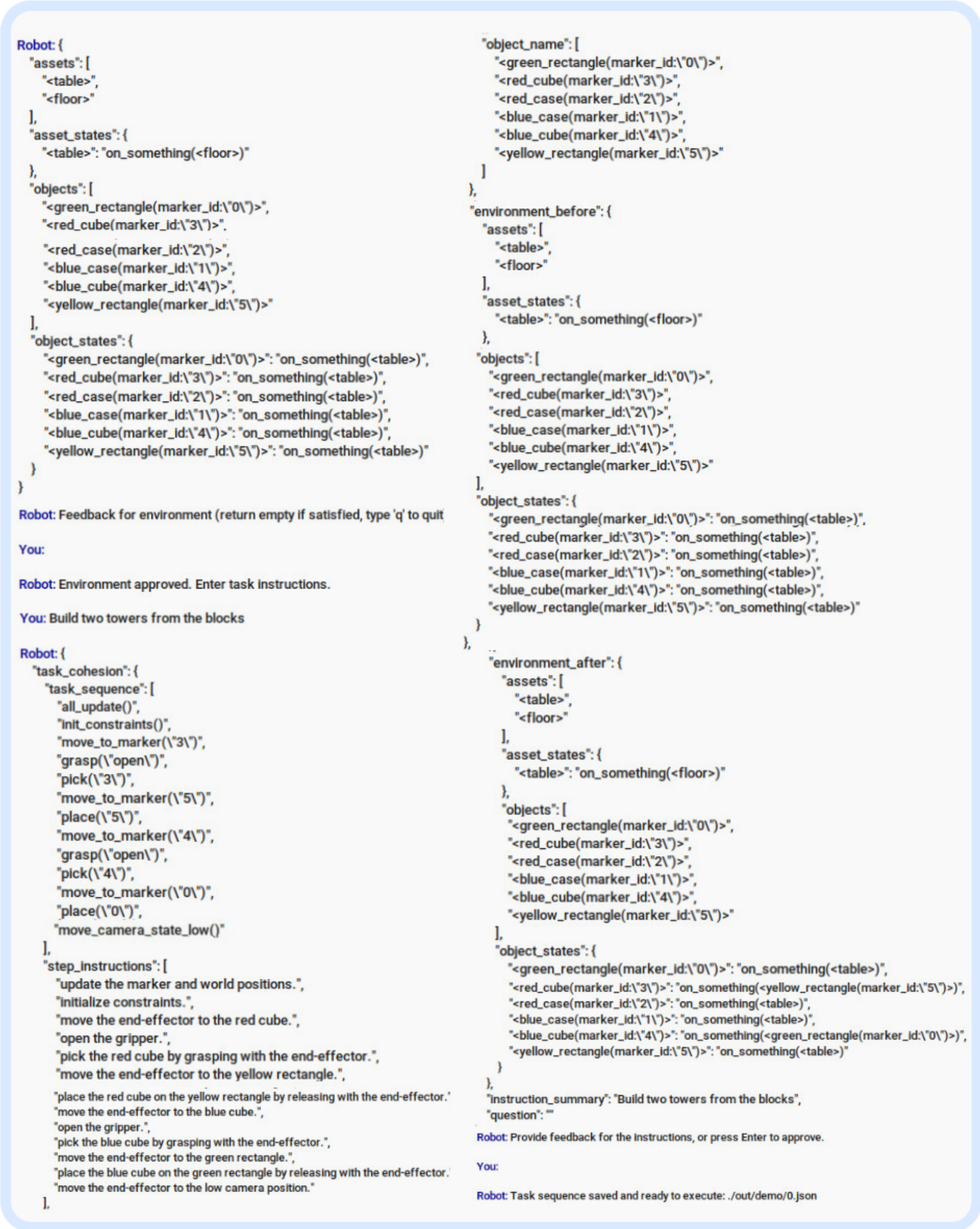
```
Here are my ratings using the Van der Laan Acceptance Scale after
     interacting with the system:

    Useful: +2
    Pleasant: +2
    Bad: -2
    Nice: +2
    Effective: +2
    Irritating: -2
    Assisting: +2
    Undesirable: -2
    Raising Alertness: +1

Overall, the system has proven to be very effective and pleasant
     to use.
```

7 Appendix B

7.1 Example conversation chat area



```
Robot: {
  "assets": [
    "<table>",
    "<floor>"
  ],
  "asset_states": {
    "<table>": "on_something(<floor>)"
  },
  "objects": [
    "<green_rectangle(marker_id:\"0\")>",
    "<red_cube(marker_id:\"3\")>",
    "<red_case(marker_id:\"2\")>",
    "<blue_case(marker_id:\"1\")>",
    "<blue_cube(marker_id:\"4\")>",
    "<yellow_rectangle(marker_id:\"5\")>"
  ],
  "object_states": {
    "<green_rectangle(marker_id:\"0\")>": "on_something(<table>)",
    "<red_cube(marker_id:\"3\")>": "on_something(<table>)",
    "<red_case(marker_id:\"2\")>": "on_something(<table>)",
    "<blue_case(marker_id:\"1\")>": "on_something(<table>)",
    "<blue_cube(marker_id:\"4\")>": "on_something(<table>)",
    "<yellow_rectangle(marker_id:\"5\")>": "on_something(<table>)"
  }
}
Robot: Feedback for environment (return empty if satisfied, type 'q' to quit

You:

Robot: Environment approved. Enter task instructions.

You: Build two towers from the blocks

Robot: {
  "task_cohesion": {
    "task_sequence": [
      "all_update()",
      "init_constraints()",
      "move_to_marker(\"3\")",
      "grasp(\"open\")",
      "pick(\"3\")",
      "move_to_marker(\"5\")",
      "place(\"5\")",
      "move_to_marker(\"4\")",
      "grasp(\"open\")",
      "pick(\"4\")",
      "move_to_marker(\"0\")",
      "place(\"0\")",
      "move_camera_state_low()"
    ],
    "step_instructions": [
      "update the marker and world positions.",
      "initialize constraints.",
      "move the end-effector to the red cube.",
      "open the gripper.",
      "pick the red cube by grasping with the end-effector.",
      "move the end-effector to the yellow rectangle.",
      "place the red cube on the yellow rectangle by releasing with the end-effector.'
      "move the end-effector to the blue cube.",
      "open the gripper.",
      "pick the blue cube by grasping with the end-effector.",
      "move the end-effector to the green rectangle.",
      "place the blue cube on the green rectangle by releasing with the end-effector.'
      "move the end-effector to the low camera position."
    ],
    "object_name": [
      "<green_rectangle(marker_id:\"0\")>",
      "<red_cube(marker_id:\"3\")>",
      "<red_case(marker_id:\"2\")>",
      "<blue_case(marker_id:\"1\")>",
      "<blue_cube(marker_id:\"4\")>",
      "<yellow_rectangle(marker_id:\"5\")>"
    ]
  },
  "environment_before": {
    "assets": [
      "<table>",
      "<floor>"
    ],
    "asset_states": {
      "<table>": "on_something(<floor>)"
    },
    "objects": [
      "<green_rectangle(marker_id:\"0\")>",
      "<red_cube(marker_id:\"3\")>",
      "<red_case(marker_id:\"2\")>",
      "<blue_case(marker_id:\"1\")>",
      "<blue_cube(marker_id:\"4\")>",
      "<yellow_rectangle(marker_id:\"5\")>"
    ],
    "object_states": {
      "<green_rectangle(marker_id:\"0\")>": "on_something(<table>)",
      "<red_cube(marker_id:\"3\")>": "on_something(<table>)",
      "<red_case(marker_id:\"2\")>": "on_something(<table>)",
      "<blue_case(marker_id:\"1\")>": "on_something(<table>)",
      "<blue_cube(marker_id:\"4\")>": "on_something(<table>)",
      "<yellow_rectangle(marker_id:\"5\")>": "on_something(<table>)"
    }
  },
  "environment_after": {
    "assets": [
      "<table>",
      "<floor>"
    ],
    "asset_states": {
      "<table>": "on_something(<floor>)"
    },
    "objects": [
      "<green_rectangle(marker_id:\"0\")>",
      "<red_cube(marker_id:\"3\")>",
      "<red_case(marker_id:\"2\")>",
      "<blue_case(marker_id:\"1\")>",
      "<blue_cube(marker_id:\"4\")>",
      "<yellow_rectangle(marker_id:\"5\")>"
    ],
    "object_states": {
      "<green_rectangle(marker_id:\"0\")>": "on_something(<table>)",
      "<red_cube(marker_id:\"3\")>": "on_something(<yellow_rectangle(marker_id:\"5\")>)",
      "<red_case(marker_id:\"2\")>": "on_something(<table>)",
      "<blue_case(marker_id:\"1\")>": "on_something(<table>)",
      "<blue_cube(marker_id:\"4\")>": "on_something(<green_rectangle(marker_id:\"0\")>)",
      "<yellow_rectangle(marker_id:\"5\")>": "on_something(<table>)"
    }
  },
  "instruction_summary": "Build two towers from the blocks",
  "question": ""
Robot: Provide feedback for the instructions, or press Enter to approve.

You:

Robot: Task sequence saved and ready to execute: ./out/demo/0.json
```

Figure 17. A snippet of a whole conversation in the chat with LERACS for the task instruction: "***Build two towers from the blocks***".

7.2 Variations of settings

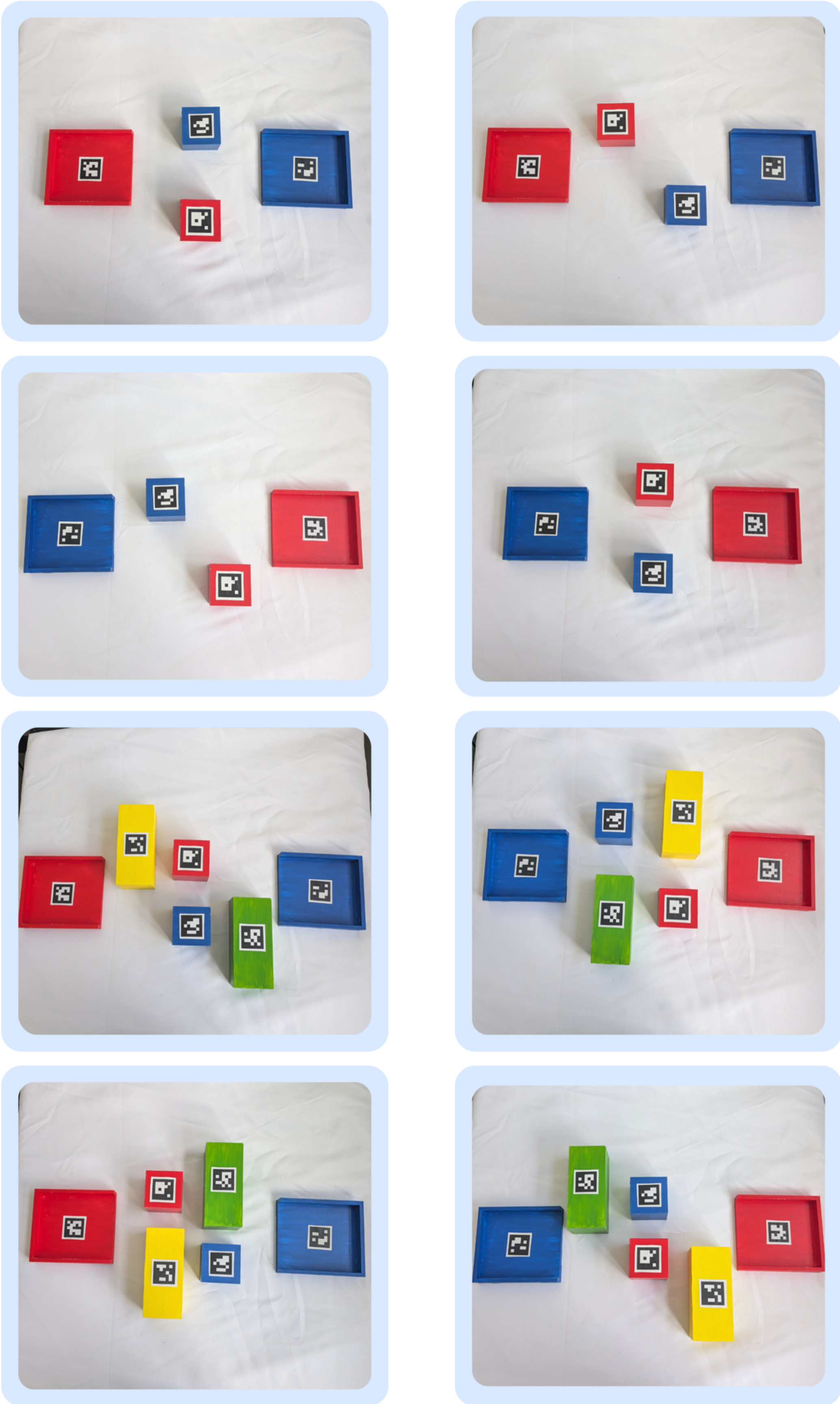


Figure 18. Overview of the different settings to test the system performance for simple (top four images) and complex (bottom four images) elements to prevent system adaptability during testing.

8 Appendix C

8.1 System acceptance scale

This scale asseses System acceptance on two dimensions, a Usefulness scale and an affective Satisfying scale. This scale gives insights in not only defining if a system is performing sufficiently, but also appealing for the user. The van der Laan questionaire itself consists of nine items displayed in *Table 5*. The instruction given to the user was: "Could you please indicate below what your opinion was about the system (...)". The user had to pick a score for each attribute, where the individual item scores run from -2 to +2. Item numbers (No) 3,6, and 8 are mirrored, compared to the other items to reduce response bias and enhance the reliability and validity of the scale.

Table 5. Van der Laan Acceptance Scale

			Scale					
Question	**No.**	**Attribute**						**Opposite Attribute**
I think this system is useful	1	Useful	+2	+1	0	-1	-2	Useless
This system is pleasant to use	2	Pleasant	+2	+1	0	-1	-2	Unpleasant
I think this system is bad	3	Bad	-2	-1	0	+1	+2	Good
I think this system is nice	4	Nice	+2	+1	0	-1	-2	Annoying
I think this system is effective	5	Effective	+2	+1	0	-1	-2	Superfluous
I think this system is irritating	6	Irritating	-2	-1	0	+1	+2	Likeable
I think this system is assisting	7	Assisting	+2	+1	0	-1	-2	Worthless
I think this system is undesirable	8	Undesirable	-2	-1	0	+1	+2	Desirable
This system raises my alertness	9	Raising Alertness	+2	-1	0	-1	-2	Sleep-inducing

8.2 AI user instructions for the simple setting

The most successful user instruction per AI persona for each scenario and environment are displayed in the following tables.

Table 6. Most common AI Person answers to goal instructions for the simple setting.

Scenario	Persona 001	Persona 002	Persona 003	Persona 004	Persona 005
Scenario 1	"Please pick up the red cube (marker_id: 3), move it to the opposite side of the table, and then hand it to me."	"Hey robot, can you please give me the red block with the number '1' on it? Pretty please"	"Please pick up the red cube (marker_id: 3) and hand it to me."	"Hey robot, can you please hand me the red block with the number '3' on it"	"Robot, please hand me the red cube."
Scenario 2	"Please pick up the blue cube (marker_id: 4), rotate it 180 degrees, and then place it inside the blue case (marker_id: 1)."	"Robot, can you put the blue block into the blue box? Let's see you do it!"	"Please place the blue cube (marker_id: 4) inside the blue case (marker_id: 1)."	"Robot, can you place the blue block with the number '4' inside the blue case with the number '1'?"	"Robot, can you please place the blue cube inside the blue case?"
Scenario 3	"Please pick up the red cube (marker_id: 3), place it precisely on top of the blue cube (marker_id: 4), and then move the entire tower to the edge of the table without it falling over."	"Robot, can you stack the red block on top of the blue block to make a tower? That would be so cool!"	"Please place the red cube (marker_id: 3) on top of the blue cube (marker_id: 4) to form a tower."	"Robot, can you stack the red block with the number '3' on top of the blue block with the number '4' to make a tower?"	"Robot, can you stack the red cube on top of the blue cube to make a tower?"
Scenario 4	"Alright robot, lets see if you can handle this. Move the red cube to the exact spot where the blue cube is, and place the blue cube where the red cube was."	"Hey robot, can you take the red block and put it where the blue block is? Please please!"	"Robot, please push the red cube to the position where the blue cube is currently located, and then push the blue cube to where the red cube was originally. Make sure to maintain accuracy while pushing."	"Hey robot, can you push the red cube (marker_id: 3) to where the blue cube (marker_id: 4) is?"	"Robot, please push the red cube to the position of the blue cube."
Scenario 5	"Robot, place the red cube in the red case and the blue cube in the blue case. Don't move the other objects; let's see if you can get this simple task right."	"Robot, can you please put the red block in the red box and the blue block in the blue box? Make sure they fit nicely, and don't mix up the colors!"	"Robot, place the red cube into the red case and the blue cube into the blue case. Ensure each block is standing properly inside its corresponding case."	"Robot, can you please place the red cube (marker_id: 3) in the red case (marker_id: 1) and the blue cube (marker_id: 4) in the blue case (marker_id: 2)?"	"Robot, place the red cube into the red case and the blue cube into the blue case. Ensure that each block is standing properly inside its corresponding case."

Scenario	Persona 001	Persona 002	Persona 003	Persona 004	Persona 005
Scenario 6	"Alright robot, let's see if you can handle this. Pick up the red cube (marker_id: 3), bring it to me, but make sure you don't bump into anything or drop it. I'm watching closely for any mistakes."	"Hey robot, can you give me the red block, please? I want to hold it! :)'	"Robot, please pick up the red cube (marker_id: 3) and hand it to me."	"Robot, please pick up the red cube (marker_id: 3) and hand it to me."	"Hey robot, can you give me the red block? I need to hold it."
Scenario 7	"Robot, pick up the blue cube (marker_id: 4) and place it perfectly in the blue case (marker_id: 1). Make sure its standing upright and centered. Any wobble or misplacement will be noted."	"Robot, can you put the blue block into the blue box? Make sure it stands up straight!"	"Robot, place the blue cube (marker_id: 4) inside the blue case (marker_id: 1). Make sure it stands properly without falling over."	"Robot, please place the blue cube (marker_id: 4) inside the blue case (marker_id: 1)."	"Robot, can you put the blue block into the blue box? Make sure it stands up straight."
Scenario 8	"Robot, place the red cube (marker_id: 3) on the table, and then stack the blue cube (marker_id: 4) on top of the red cube to form a tower. Ensure the tower is stable and perfectly aligned. Any wobble or collapse will be noted."	"Robot, can you put the red block on the table and then put the blue block on top of it to make a tower? I want to see if it stands up!"	"Robot, place the red cube (marker_id: 3) on the table, then stack the blue cube (marker_id: 4) on top of the red cube to form a tower. Ensure the tower is stable and perfectly aligned."	"Robot, please place the red cube (marker_id: 3) on the table and then place the blue cube (marker_id: 4) on top of the red cube to form a tower."	"Robot, can you put the red block on the table and then place the blue block on top of the red block to make a tower?"
Scenario 9	"Alright robot, let's make this interesting. First, nudge the blue cube halfway to the red cube. Then, push the red cube all the way where the blue cube was located."	"Hey robot, can you please take the red block and put it where the blue block is?"	"Robot, please push the red cube to the position where the blue cube is currently located, and then push the blue cube to where the red cube was originally. Ensure precise and controlled movements to avoid any collisions."	"Hey robot, can you push the red cube to where the blue cube is?"	"Robot, can you move the red cube to where the blue cube is now."
Scenario 10	"Point to the places that the robot can put the cubes in. Finally, place the red cube (marker_id: 5) inside the red case and the blue cube (marker_id: 10) inside the blue case."	"Hey robot, can you put the red block in the red box and the blue block in the blue box?"	"Robot, place the red cube into the red case and the blue cube into the blue case. Ensure each block is standing properly inside its corresponding case."	"Robot, can you please place the red cube in the red case and the blue cube in the blue case? Make sure they're in the right places!"	"Hey robot, can you put the red block in the red case and the blue block in the blue case?"

8.3 AI user instructions for the complex setting

Table 7. Most common AI Person answers to goal instructions for the complex setting.

Scenario	Persona 001	Persona 002	Persona 003	Persona 004	Persona 005
Scenario 1	“pick up the blue cube with marker ID ’5’ and hand it to me.”	“Robot, can you please hand me the green block?, Thanks!”	“The task for the robotic arm is to pick up the green rectangle with marker ID "0" and hand it to me.”	“pick up the blue rectangle with marker ID ’1’ and hand it to me”	“The robotic arm needs to hand me the green rectangle”
Scenario 2	“Pick up the yellow rectangle (marker_id: ’5’), balance it on top of the blue cube (marker_id: ’4’), and place this stack inside the red case (marker_id: ’2’). Then pick up the green rectangle (marker_id: ’0’) and place it standing inside the blue case (marker_id: ’1’).”	“Hey robot, put the green block in the red box and the blue block in the blue box! :)”	“Move the green rectangle (marker_id: "0") into the red case (marker_id: "2"), the red cube (marker_id: "3") into the blue case (marker_id: "1"), and build two towers using the remaining blocks.”	“Ensure that the green rectangle and the yellow rectangle are standing upright inside the respective cases.”	“Move the green rectangle (marker_id:0̈) into the red case (marker_id:2̈).”
Scenario 3	“Move the green cube (marker_id: "5") and place it on top of the blue cube (marker_id: "4"). Then, move the red cube (marker_id: "3") and place it on top of the yellow rectangle (marker_id: "0")”	“Let’s make two fun towers! First, pick up the green block and put it on top of the blue block. Yay, first tower done! Next, grab the red block and place it on the yellow block. Hooray, now we have two towers! So cool! :)”	“To build the two towers, start by selecting the Yellow Rectangle with marker ID 5 and place it as the base for the first tower. Then, take the Red Cube with marker ID 3 and carefully position it on top of the Yellow Rectangle. For the second tower, use the Green Cube with marker ID 0 as the base. Finally, place the Blue Cube with marker ID 4 on top of the Green Cube”	“Please build two towers by placing the green cube (marker_id: ’0’) on the blue case (marker_id: ’4’), and the red cube (marker_id: ’3’) on the yellow rectangle (marker_id: ’5’). Thank you!”	‘build a single tower using two of the available blocks”
Scenario 4	“Move the red cube (marker_id:"3") to the position of the yellow rectangle (marker_id:"5") and place the yellow rectangle (marker_id:"5") in the original position of the red cube (marker_id:"3").”	“Can you please move the yellow block to where the green block is? Thanks!”	“Push the red cube (marker_id: 3) forward (away from the robot) three times to move it to the position of the green cube (marker_id: 0). Then, push the green cube (marker_id: 0) backward (toward the robot) three times to move it to the original position of the red cube. ”	“Please move the red cube (marker_id: 3) to a different position, and then place the blue cube (marker_id: 4) in the position previously occupied by the red cube (marker_id: 3)”	“Move the green rectangle to the position of the yellow rectangle, by moving the green one out the way in one of the cases first.”
Scenario 5	“Move the red cube (marker_id: 3̈) into the red case (marker_id: 2̈) and the blue cube (marker_id: 4̈) into the blue case (marker_id: 1̈). Ignore the yellow rectangle (marker_id: 5̈) and the green cube (marker_id: 0̈), as they don’t have corresponding cases. Make sure no block is misplaced or left in an incorrect position.”	“Take the red block and put it in the red box. Then take the blue block and put it in the blue box.”	“Sort the red cube (marker_id: "3") into the red case (marker_id: "2") and the blue cube (marker_id: "4") into the blue case (marker_id: "1"), leaving the yellow and green rectangles in their current positions.”	“Move the red cube into the red case and the blue cube into the blue case. Ignore the other blocks for this task.”	“First, place the red cube into the red case. Next, take the blue cube and put it into the blue case. Leave any other blocks aside for now.”

Scenario	Persona 001	Persona 002	Persona 003	Persona 004	Persona 005
Scenario 6	"pick up the green rectangle with marker ID '0' and hand it to the user without touching any other objects"	"Robot, can you please hand me the green long box?"	"Robot, please hand me the green rectangle (marker_id: '0')."	"The robot should first identify the yellow rectangle with the marker ID "5." Then, it should use the robotic arm to pick up the yellow rectangle. After picking it up, the robot should move the yellow rectangle towards me and gently place it in my hand."	"Robot, please hand me the yellow rectangular block."
Scenario 7	"Stack the green rectangle on top of the yellow block and place the stack inside the blue case"	"Hi robot! Can you help the rectangles find their home? First, pick up the green rectangle thats on the table. Then, place it into the blue spot. Find the yellow rectangle and put it into the blue case too. Youre doing great, just like a fun game!"	"Robot, lets execute this with precision. Start by identifying the green rectangle (marker_id: "0") on the table. Lift the green rectangle and transfer it into the blue case (marker_id: "1"). Following this, locate the yellow rectangle (marker_id: "5") and similarly, place it into the blue case."	"First, identify the red cube with marker ID 3 and the red case with marker ID 2. Pick up the red cube and place it inside the red case. Next, locate the blue cube with marker ID 4 and the blue case with marker ID 1. Pick up the blue cube and place it inside the blue case."	"Alright robot, lets get this done efficiently. Start by picking up the green rectangle from the table and place it into the blue case. Then, take the yellow rectangle and put it into the blue case as well."
Scenario 8	"Move the yellow rectangle (marker_id: 5) onto the red case (marker_id: 2). Then, place the red cube (marker_id: 3) on top of the yellow rectangle (marker_id: 5). Make sure the tower is stable and the red block is perfectly on top of the yellow rectangle. Let's see if the system can handle this without any issues."	"Put the yellow block on the table. Then put the red block on top of the yellow one. Make a tower! :)′	"Pick up the red cube and place it on top of the yellow rectangle to form a tower"	"Pick up the yellow rectangle and place it vertically on the table, then place the red cube on top of the yellow rectangle"	"Please pick up the yellow rectangle and place the red block on top of it to form a tower"
Scenario 9	"First, find the blue cube, pick it up, and move it to a temporary spot. Then, locate the yellow rectangle, pick it up, and move it out of the way. After that, move the blue cube to the exact spot where the yellow rectangle was and place it there. This way, the blue cube will stand exactly in the place of the yellow rectangle"	"Hey robot, can you please move the blue block to where the yellow rectangle is? Thanks!"	"Move the blue block (marker_id: 1) to the position currently occupied by the yellow rectangle (marker_id: 5) by moving the yellow rectangle in a case first and then place the blue block on the previous position of the yellow rectangle."	"Please move the blue block (marker_id: 1) to the position currently occupied by the yellow rectangle (marker_id: 5)."	"Please move the blue block to the position currently occupied by the yellow rectangle by moving the yellow rectangle out of the way."

Continued on next page

Table 8 – *Continued from previous page*

Scenario	Persona 001	Persona 002	Persona 003	Persona 004	Persona 005
Scenario 10	"Instruct the robotic arm to pick up the red cube (marker_id: "6") and place it in the red case (marker_id: "2"). Then, have it place the blue cube (marker_id: "7") in the blue case (marker_id: "1"). Next, move the yellow rectangle (marker_id: "8") into the yellow case. Finally, place the green rectangle (marker_id: "9") into the green case."	"Hey robot, can you put the red block in the red box and the blue block in the blue box? Thanks!"	"Robot, please place the red block inside the red case and the blue block inside the blue case. Ensure they are standing properly."	"Please move the red block into the red case, the blue block into the blue case, and ensure the yellow rectangle and green rectangle are each standing in separate cases."	"Please move the red block into the red case, the blue block into the blue case, and ensure the yellow rectangle and green rectangle are each standing in separate cases."

8.4 Statistical analysis on the difference in complexity levels for both setting and task instruction levels

In the statistical analysis the complexity variables for setting and task types are compared against each other over dependent variables in all scenarios. a two-way ANOVA is performed because the data consists of quantitative dependent variables at multiple levels of two categorical independent variables [90] . The two-way ANOVA with interaction tests three null hypotheses per dependent variable **[task/sequence execution success**, **environment interpretation success**,**task/sequence generation success**,**labeling success (precision)**,**labeling success(recall/sensitivity)**,**labeling success(F1 score)**]) at the same time:

Null hypotheses (H0)

1. There is no difference in average **[dependent variable]** for any setting complexity type.
2. There is no difference in average **[dependent variable]** for any task complexity type.
3. The effect of setting complexity on average **[dependent variable]** does not depend on the effect of task complexity.

For this analysis the data is assumed to have homogeneity of variance, independence of observations and normally distributed dependent variables [91]. The tables with the results for the statistical analysis are continued on the next page.

Table 9. Statistical analysis of the system validation experiment data (Two-way ANOVA)

Task/Sequence Execution Success

	C(Setting)	C(Task Types)	C:(Interaction)
df	1	4	4
Sum of Squares	0.01	0.06	0.34
Sample Size	100 (2 groups)	100 (5 groups)	100
F Value	0.148	0.221	1.254
P Value	0.702	0.926	0.294
Significant Difference	False	False	False

Environment Interpretation Success

	C(Setting)	C(Task Type)	C:(Interaction)
df	1	4	4
Sum of Squares	0.11	1.07	0.43
Sample Size	100 (2 groups)	100 (5 groups)	100
F Value	6.060	15.329	6.149
P Value	0.016	< 0.001	0.0002
Significant Difference	True	True	True

Task/Sequence Generation Success

	C(Setting)	C(Task Type)	C:(Interaction)
df	1	4	4
Sum of Squares	0.03	0.64	0.36
Sample Size	100 (2 groups)	100 (5 groups)	100
F Value	5.444	28.444	16.000
P Value	0.022	< 0.001	< 0.001
Significant Difference	True	True	True

Labeling Success (Precision)

	C(Setting)	C(Task Type)	C:(Interaction)
df	1	4	4
Sum of Squares	0.43	0.10	0.03
Sample Size	100 (2 groups)	100 (5 groups)	100
F Value	26.365	1.595	0.434
P Value	< 0.001	0.183	0.784
Significant Difference	True	False	False

Labeling Success (Recall/Sensitivity)

	C(Setting)	C(Task Type)	C:(Interaction)
df	1	4	4
Sum of Squares	0.01	0.03	0.01
Sample Size	100 (2 groups)	100 (5 groups)	100
F Value	0.606	0.503	0.151
P Value	0.439	0.733	0.962
Significant Difference	False	False	False

Labeling Success (F1 Score)

	C(Setting)	C(Task Type)	C:(Interaction)
df	1	4	4
Sum of Squares	0.29	0.06	0.02
Sample Size	100 (2 groups)	100 (5 groups)	100
F Value	24.637	1.168	0.461
P Value	< 0.001	0.330	0.764
Significant Difference	True	False	False

Table 10. Statistical analysis of the system validation experiment data (Tukey post-hoc test)

Environment Interpretation Success Setting

Group1	Group2	Mean Difference	p-Value	Lower	Upper
complex	simple	0.0650	0.0692	-0.0052	0.1352

Environment Interpretation Success Task Type

Group1	Group2	Mean Difference	p-Value	Lower	Upper
1	2	-0.0125	0.9989	-0.1433	0.1183
1	3	-0.0750	0.5049	-0.2058	0.0558
1	4	-0.2750	0.0000	-0.4058	-0.1442
1	5	-0.0125	0.9989	-0.1433	0.1183
2	3	-0.0625	0.6743	-0.1933	0.0683
2	4	-0.2625	0.0000	-0.3933	-0.1317
2	5	0.0000	1.0000	-0.1308	0.1308
3	4	-0.2000	0.0005	-0.3308	-0.0692
3	5	0.0625	0.6743	-0.0683	0.1933
4	5	0.2625	0.0000	0.1317	0.3933

Task Sequence Generation Success Setting

Group1	Group2	Mean Difference	p-Value	Lower	Upper
complex	simple	0.0350	0.1612	-0.0142	0.0842

Task Sequence Generation Success Task Type

Group1	Group2	Mean Difference	p-Value	Lower	Upper
1	2	0.0000	1.0000	-0.0854	0.0854
1	3	0.0000	1.0000	-0.0854	0.0854
1	4	-0.2000	0.0000	-0.2854	-0.1146
1	5	0.0000	1.0000	-0.0854	0.0854
2	3	0.0000	1.0000	-0.0854	0.0854
2	4	-0.2000	0.0000	-0.2854	-0.1146
2	5	0.0000	1.0000	-0.0854	0.0854
3	4	-0.2000	0.0000	-0.2854	-0.1146
3	5	0.0000	1.0000	-0.0854	0.0854
4	5	0.2000	0.0000	0.1146	0.2854

Labeling Success Precision Setting					
Group1	**Group2**	**Mean Difference**	**p-Value**	**Lower**	**Upper**
complex	simple	-0.1305	0.0000	-0.1810	-0.0800

Labeling Success F1 Score Setting					
Group1	**Group2**	**Mean Difference**	**p-Value**	**Lower**	**Upper**
complex	simple	-0.1079	0.0000	-0.1507	-0.0651

Table 11. Statistical analysis of the system validation experiment data (pairwise t-test post-hoc)

Pairwise t-Test Results for Environment Interpretation Success		
Test	**p-Value**	**Conclusion**
1 vs 2	0.9989	Not Significant
1 vs 3	0.5049	Not Significant
1 vs 4	0.1433	Not Significant
1 vs 5	0.9989	Not Significant
2 vs 3	0.5049	Not Significant
2 vs 4	0.6743	Not Significant
2 vs 5	0.6743	Not Significant
3 vs 4	0.0840	Not Significant
3 vs 5	0.0053	Significant
4 vs 5	0.0000	Significant

Pairwise t-Test Results for Task Sequence Generation Success		
Test	**p-Value**	**Conclusion**
1 vs 2	1.0000	Not Significant
1 vs 3	1.0000	Not Significant
1 vs 4	1.0000	Not Significant
1 vs 5	0.0011	Significant
2 vs 3	1.0000	Not Significant
2 vs 4	1.0000	Not Significant
2 vs 5	0.0011	Significant
3 vs 4	1.0000	Not Significant
3 vs 5	0.0011	Significant
4 vs 5	0.0011	Significant

t-Test Results for Labeling Success (Precision and F1 Score)

Test	p-Value	Conclusion
Precision (Simple vs Complex)	0.000002	Significant
F1 Score (Simple vs Complex)	0.000003	Significant

Pairwise t-Test Results for Environment Interpretation Success

Test	p-Value	Conclusion
1 vs 2	0.9989	Not Significant
1 vs 3	0.5049	Not Significant
1 vs 4	0.1433	Not Significant
1 vs 5	0.9989	Not Significant
2 vs 3	0.5049	Not Significant
2 vs 4	0.6743	Not Significant
2 vs 5	0.6743	Not Significant
3 vs 4	0.0840	Not Significant
3 vs 5	0.0053	Significant
4 vs 5	0.0000	Significant

Table 12. Pairwise t-Test Results for Environment Interpretation Success

Pairwise t-Test Results for Task Sequence Generation Success

Test	p-Value	Conclusion
1 vs 2	1.0000	Not Significant
1 vs 3	1.0000	Not Significant
1 vs 4	1.0000	Not Significant
1 vs 5	0.0011	Significant
2 vs 3	1.0000	Not Significant
2 vs 4	1.0000	Not Significant
2 vs 5	0.0011	Significant
3 vs 4	1.0000	Not Significant
3 vs 5	0.0011	Significant
4 vs 5	0.0011	Significant

Table 13. Pairwise t-Test Results for Task Sequence Generation Success

t-Test Results for Labeling Success (Precision and F1 Score)

Test	p-Value	Conclusion
Precision (Simple vs Complex)	0.000002	Significant
F1 Score (Simple vs Complex)	0.000003	Significant

Table 14. t-Test Results for Labeling Success (Precision and F1 Score)

9 Appendix D

9.1 Switching fuses on high voltage racks case application

This Appendix consists of all the elements regarding the case application in Alliander for switching fuses on a high voltage rack. The voltage racks have been placed horizontally on a table with a robot arm for testing where the cases are provided with ArUco markers as can be seen in *Figure 19*. The user provides the instruction "**Can you prepare the fuse holders and then place the dummy fuse in the middle holder**" in the LERACS interface, the interface with the full output is displayed in *Figure 20*. For performing the decomposed tasks for this instruction a new motion primitive is added: **opendrawer()** (*Figure 21*). The output of the environment interpretation and the task/sequence generation is highlighted in **Output Environment** and **Output Task Decomposition** with the full executed sequence (*Figure 22*).

Figure 19. This figure represents the apparatus and the setup for the robot arm to perform the action of switching fuses in the voltage rack cases.

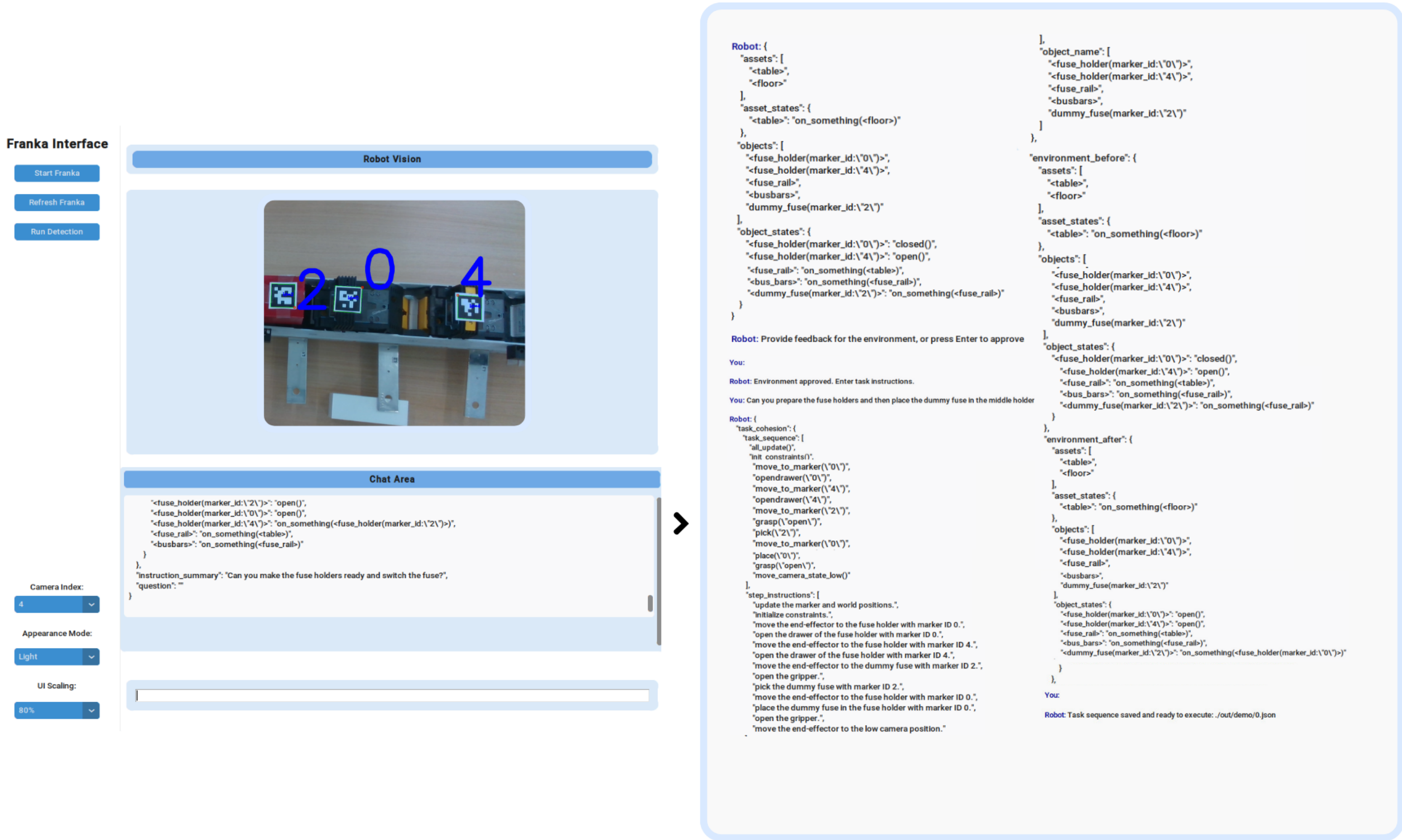


Figure 20. This figure displays the user interface for a case application at Alliander: switching fuses in a voltage rack. Hereby the user can instruct the robot arm to switch fuses from a voltage rack

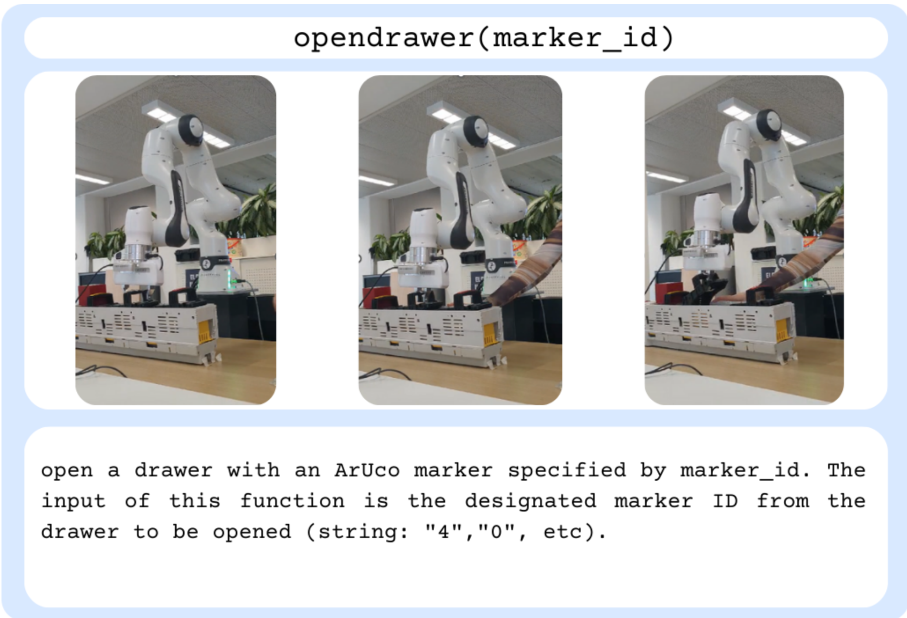


Figure 21. In order to perform the task of switching fuses on high voltage racks the opendrawer function needs to be added.

Output Environment

```
"environment":{{"assets":["<table>", "<floor>"],
"asset_states": {{"<table>": "on_something(<floor>)"}},
"objects": ["<fuse_holder(marker_id:"0")>",
           "<fuse_holder(marker_id:"4")>",
           "<fuse_rail>",
           "<bus_bars>"]",
           "<dummy_fuse(marker_id:"2")>",
"object_states": {{"<fuse_holder(marker_id:"0")>":"closed()",
                  "<fuse_holder(marker_id:"0")>":"open()",
                  "<fuse_rail>":"on_something(<table>)",
                  "<bus_bars>":"on_something(<fuse_rail>)",
                  "<dummy_fuse(marker_id:"2")>":"on_something(<
                       fuse_rail>)"
}}
}
```

Output Task Decomposition

```
{"task_cohesion": {
       "task_sequence": [
          "all_update()",
          "init_constraints()",
          "move_to_marker(\"0\")",
          "opendrawer(\"0\")",
          "move_to_marker(\"4\")",
          "opendrawer(\"4\")",
          "move_to_marker(\"2\")",
          "grasp(\"open\")",
          "pick(\"2\")",
          "move_to_marker(\"0\")",
          "place(\"0\")",
          "grasp(\"open\")",
          "move_camera_state_low()"
          }}}
```

Figure 22. this Figure shows the most important motion primitives executed in a sequence for the instruction: "Can you prepare the fuse holders and then place the dummy fuse in the middle holder".