In: 2024 IEEE-RAS 23rd International Conference on Humanoid Robots (Humanoids), Nancy, France, 2024

Robots Can Multitask Too: Integrating a Memory Architecture and LLMs for Enhanced Cross-Task Robot Action Generation

Hassan Ali[1], Philipp Allgeuer[1], Carlo Mazzola[2], Giulia Belgiovine[2], Burak Can Kaplan[1], Lukáš Gajdošech[3] and Stefan Wermter[1]

***Abstract*— Large Language Models (LLMs) have been recently used in robot applications for grounding LLM commonsense reasoning with the robot's perception and physical abilities. In humanoid robots, memory also plays a critical role in fostering real-world embodiment and facilitating long-term interactive capabilities, especially in multi-task setups where the robot must remember previous task states, environment states, and executed actions. In this paper, we address incorporating memory processes with LLMs for generating cross-task robot actions, while the robot effectively switches between tasks. Our proposed dual-layered architecture features two LLMs, utilizing their complementary skills of reasoning and following instructions, combined with a memory model inspired by human cognition. Our results show a significant improvement in performance over a baseline of five robotic tasks, demonstrating the potential of integrating memory with LLMs for combining the robot's action and perception for adaptive task execution.**

I. INTRODUCTION

Despite the physical limitations due to their embodiment, humanoid robots are particularly effective tools because of their anthropomorphic shape, which can significantly improve the versatility and effectiveness of robots, especially in environments designed for human interaction [1]. Moreover, the humanoid physical shape of the robot supports collaboration with humans by making its actions more legible and predictable, which positively impacts safety and trust during interactions [2]. Inspired by the effectiveness and adaptability of human cognitive abilities, cognitive robotics aims to draw from more than embodiment when designing robotic platforms. Key attributes like human perception, selective attention, different types of memory, and other facets of human cognition are essential to improve the autonomy, flexibility, and capabilities of robots [3].

The development and utilization of foundational models in robotic platforms have been fundamental in driving advancement toward building autonomous artificial agents. Large Language Models (LLMs) have been used as symbolic reasoning instruments in various robotic applications such as reasoning about user's intentions in human-robot collaboration [4], generating robot action plans in grounded environments [5], and task planning in environment-sensitive robot agents [6]. Nevertheless, LLM reasoning alone is not yet sufficient for implementing the cognitive system of embodied artificial agents, capable of solving complex tasks and interacting with humans. Therefore, integrating LLMs with other cognitive abilities is essential for achieving this aim. Memory, in particular, is a capability that has been proven to contribute to the effectiveness of information processing in LLMs (for a survey, see [7]).

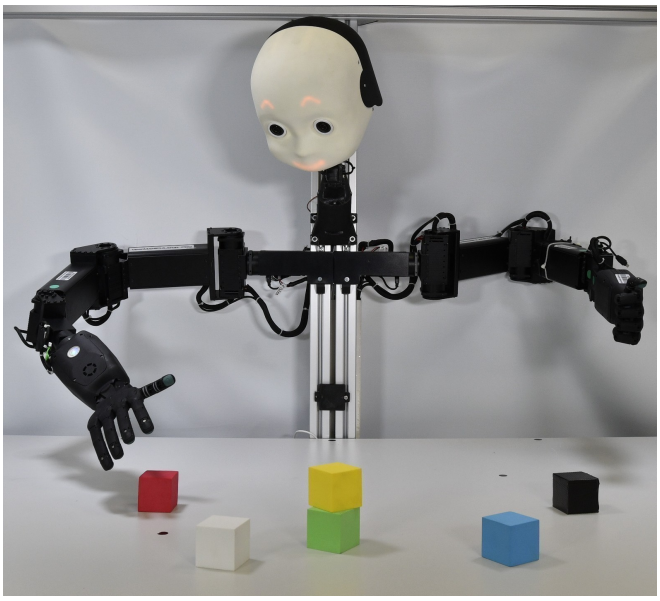

Fig. 1. Our setup with the semi-humanoid NICOL robot.

Taking inspiration from human cognition, several memory-related processes and mechanisms can be identified based on their specific roles and functionality, which can be implemented accordingly. In humans, although the precise taxonomy remains debated, short-term and long-term memory distinctively vary in the duration and capacity to retain information [8]. While declarative memory stores events and information, procedural memory has been conceptualized as the memory storing conditional instructions detailing actions based on circumstances [9]. On the other hand, working memory integrates both representations from declarative and procedural memory [9], enabling the retention of essential information for an ongoing cognitive task, a process that is deeply connected with selective attention mechanisms [10].

Starting from this classification, in this work, we develop a hierarchical modular architecture to support the semi-humanoid NICOL collaborative robot (see Fig. 1) in rea-

Work supported by Horizon Europe project TERAIS (Grant number 101079338), the DFG Crossmodal Learning (TRR-169) project, and Study Abroad Graduate Scholarship by Ministry of National Education of Türkiye.

[1]Hassan Ali, Philipp Allgeuer, Burak Can Kaplan and Stefan Wermter are with the Knowledge Technology Group, Department of Informatics, University of Hamburg, Germany. Emails: {hassan.ali, philipp.allgeuer, burak.can.kaplan, stefan.wermter}@uni-hamburg.de

[2]Carlo Mazzola and Giulia Belgiovine are with the CONTACT Unit, Italian Institute of Technology, Genoa, Italy.

[3]Lukáš Gajdošech is with the Faculty of Mathematics, Physics and Informatics, Comenius University, Slovakia.

soning with different kinds of memory functions. Thus, we enable the robot to solve the requests of a human agent by connecting 1) sensory information from the environment (perception), 2) LLM agent prompting (reasoning), and 3) task-appropriate robot actions produced as output. Although simply linking these three pillars might be sufficient for simple distinct tasks, the effectiveness might not be as high for more complex situations, like the multi-tasking required for natural interaction with humans. Therefore, we evaluate the impact of memory on multi-task execution and test our architecture with two different families of LLMs (OpenAI's GPT-3.5 and Meta's open-weight Llama 3). Our framework aligns and combines the environment data with task-specific memory cues, effectively bridging the gap between the robot's perception and action and laying solid ground for efficient task execution in contextually based interactions.

II. Related Work

Integrating sensory input with memory is essential in artificial systems as it enables them to retain sensory information for environment recognition and adaptive responses [11]. Moreover, memory plays a vital role in humanoid robots since it brings together conceptual knowledge gained through experience with the robot's behavior, enabling adjustment to complex interactive scenarios and tasks [12]. Memory-based architectures have various applications such as transferring robot manipulation skills across different human environments [13], memory-powered incremental learning with human-in-the-loop corrections [14], and supporting tasks like action-planning for localizing a target and navigating towards it [15]. Existing work incorporates models of memory into robot tasks like planning with occluded objects [16], reasoning about object permanence [17], and learning human personal attributes like voices and faces [18]. However, such approaches model memory representations and reasoning as distinct components or separate processes while embedding memory directly within a reasoning backbone, like an LLM, is beneficial for context-aware decision-making.

LLMs have recently been used to support robots' reasoning and decision-making. However, relying solely on LLM's own internal memory is insufficient for handling complex multi-task reasoning demands of dynamic robotic environments. Some approaches address the limitation in LLM memory by injecting curated memory cues as contextual information into the LLM's prompts [19], using previous structurally stored dialogues to enrich current LLM input [20], or caching previous conversations in a long-term memory storage [21]. Other methods rely on retraining or fine-tuning LLMs with contextual memory information [22][23]. However, relying on LLM context memory is constrained by context length and can lead to reasoning instability, while updating LLM parametric memory requires retraining and fine-tuning, causing cost and robustness concerns [7]. The limitations of LLM's own memory are also underscored in our experiments, highlighting the need for combining LLMs with efficient and up-to-date robust memory systems.

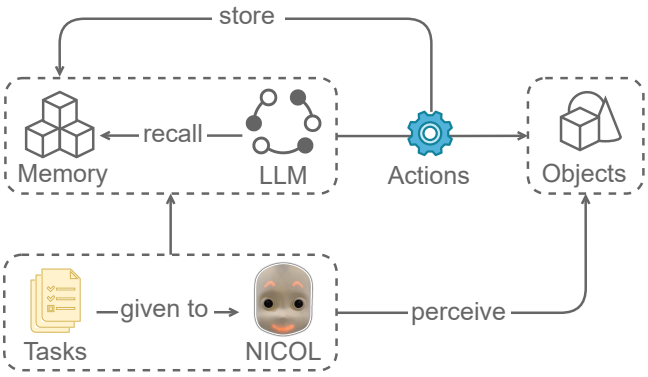

Fig. 2. A simplified overview of our system's workflow. The workflow starts when the NICOL robot receives a given task. The robot's sensory and task inputs are then fed to our proposed LLM-powered architecture.

Our work addresses these limitations by incorporating LLMs for reasoning and action generation in a hierarchical framework. The idea of leveraging a hierarchy of LLMs to support reasoning and task specialization exists in the literature. For example, some approaches use multi-level LLMs to mitigate the latency issues of OpenAI APIs for real-time performance [24], combine LLM vision and textual processing capabilities [25], and support efficiency in chatbot applications with levels of user-centered queries [26]. However, our LLM hierarchy differs from these approaches by utilizing a coordinator/worker LLM architecture, each with a different scope of the interaction context. Our structure also combines general LLM reasoning with specialized-task memory action execution, supporting multi-task reasoning and cognitive load reduction on each individual model.

III. Methodology

We take inspiration from human cognition to integrate a memory framework with the robot's reasoning and action generation. In Fig. 2, we show a simplified overview of our system's workflow. The robot continuously observes the objects in its proximate environment through its visual perception and has specifications for pre-defined tasks. Upon a new task command, the system triggers the LLM to store and recall relevant memory components. Consequently, a suitable robot action is generated, applicable to the objects, and suitable to the task. Every time an action is performed, the memory records are updated, creating and maintaining a live memory repository and forming a continuous and adaptive feedback loop between the robot's actions, its memory, and the environment. We utilize two LLMs for managing the memory models, each optimized with prompts designed to achieve specific stages of the memory pipeline. In this section, we introduce our robot, its visual perception module, and present our proposed architecture and memory models.

A. Robotic Platform: NICOL

Our system runs on the Neuro-Inspired COLlaborator (NICOL) [27] built in the Knowledge Technology group at the University of Hamburg. NICOL is a tabletop robotic platform with a humanoid head and two 8 DoF robotic arms. The robot features social cues like facial expressions as well

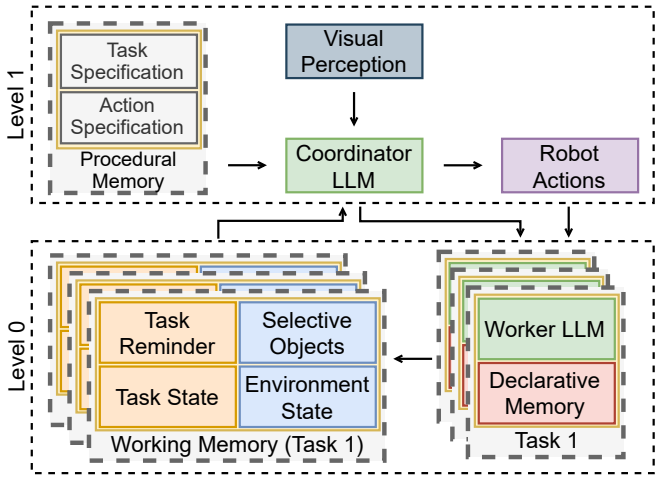

Fig. 3. Our proposed system architecture consisting of two layers: *level 0* and *level 1*, utilizing the *worker* and *coordinator* LLM, respectively.

as object and arm manipulation capabilities like grasping and neuro-genetic inverse kinematics [28]. The NICOL platform is based on the Robot Operating System (ROS), facilitating integration with various sensory modules and deep learning models. It also supports open communication and collaboration with human users through an LLM-powered modular architecture [29] that combines sensory input like visually grounded open-vocabulary object detection (ViLD [30]) and speech recognition (Whisper [31]) with actions like arm and head manipulation. Thus, it equips the robot with social and cognitive skills via its grounded LLM. In our approach, we extend NICOL's skills, but unlike previous work, we focus on adaptive action generation in robot multi-task scenarios.

B. NICOL's Visual Perception and Action Parsing

We use NICOL's 4K camera located inside the robot's left eye to capture the environment visually and the objects on the table. We use the ViLD [30] object detector, enabling zero-shot object detection and recognition without pre-training. The NICOL's ViLD object detector has been optimized to work in real-time on continuous camera streams with high efficiency and reliability [29]. The detector generates bounding boxes for the objects on the table and their locations. All objects detected outside the area of interest, i.e., the robot's table, are removed. The objects detected are expressed by text so all object labels can be streamlined within the system's pipeline, making it compatible with an LLM application. NICOL can also parse textual LLM-generated actions and translate them into robot physical actions [29]. The robot can perform arm manipulations like pointing to objects on the table, grasping objects, and pushing objects to the user. For example, *"point to the banana"* can be parsed and translated into corresponding physical actions performed by the robot's arm, i.e., an extended arm and extended index finger gesture to indicate the location of the banana on the table. The robot also has other head manipulation and facial expression capabilities. However, our system utilizes only the robot's arm manipulation skills in its action and task specifications.

Separating task: Your task is to sort objects in different boxes based on their properties. Fruits should go to box 1 while kitchenware and containers should go to box 2. For this task, you can place objects in the corresponding boxes using appropriate actions from the list of functions, for example: <move_to_box_1(banana)>.

Arrangement task: Your task is to place only the fruits inside the bowl. Do not put other objects inside the bowl. Use the <place_in_bowl> action function to put the fruits one by one inside the bowl, for example, <place_in_bowl(banana)>.

Pointing task: Your task is to point at objects in the following specific order, point at each yellow object then only when you have pointed at all the yellow objects, point at each red object on the table. Objects can be fruits, containers, or any other sort of object. Do not point at any object that is neither red nor yellow. Do not point at any object that is not on the table. At each step, check the remaining objects on the table to choose an object that fits the criteria. For example, to point at the banana, you can use <point(banana)>. Similarly, repeat for all objects.

Recipe task: Your task is to give all the ingredients for making a jello recipe with banana topping. The required objects are a bowl, jello, and banana. Do not give any other objects besides the required ingredients. Use the <give> action function to give the objects one by one. Generate an action function for each object, for example <give(banana)>.

Tower task: your task is to build a tower using colored cubes. Use the appropriate action function to stack cube objects on top of each other one by one. The order of the colored cubes does not matter. Generate an action function for each cube object, for example, <put_on_tower(red cube)>. But only use colored cubes for building the tower. Never use other objects for building the tower.

When asked to perform this task, generate a set of actions to achieve it using the current objects. You should generate one action for each individual object.

Prompt 1: Task specifications as given to the coordinator LLM in the base prompt.

C. Proposed Architecture

Our architectural design comprises a stack of two layers (cf. Fig. 3), which work collaboratively to achieve adaptive task execution. Each layer features an LLM corresponding to a specialized level of reasoning and memory capabilities but works in tandem with the other LLM to generate a set of robotic actions and achieve task-oriented goals. In the lower level (level 0), a *worker* LLM is dedicated to creating and maintaining two memory variants (*working* and *declarative* memory) based on real-time inputs from the environment and interaction history, and supplying foundational information for the subsequent decision-making processes within the system. The worker LLM represents our instruction-based LLM which receives structured commands triggered by the *coordinator* LLM, resulting in structured and precise outputs.

The coordinator LLM in level 1 of the stack represents our reasoning LLM backbone which triggers the different functions of the worker LLM, integrates its output with the task's context, while focusing on complex task reasoning and decision-making. The coordinator LLM integrates input from the environment like the objects on the table and task-specific memory cues received from the worker LLM to generate robot-actionable commands and instructions, suitable to the robotic task at hand. For example, the output of the coordinator LLM in a tower building task might be an action like *"put the yellow cube on top of the cube tower"* that requires reasoning and understanding of the task at hand, objects on the table, and overall context of previous interactions (which objects should be stacked in the tower, remaining cubes on the table, cubes already stacked in the tower, etc.).

Collectively, the two layers (level 0 and level 1) enable the system to adapt to the changing environment as the objects on the table and task requirements are continuously changing, i.e., smoothly and seamlessly switching between tasks. Since both LLMs use text for input and output,

<point(object)>: Given a string of an object name, use your arms to point to that object on the table.

<give(object)>: Given a string of an object name, use your arms to push that object on the table and give it to the user. You can hand objects to the user with this function.

<move_to_box_1(object)>: Given a string of an object name, use your arms to grasp that object on the table, lift it and drop it inside box 1 on the table. You can move objects to box 1 with this function.

<move_to_box_2(object)>: Given a string of an object name, use your arms to grasp that object on the table, lift it and drop it inside box 2 on the table. You can move objects to box 2 with this function.

<put_on_tower(object)>: Given a string of an object name, use your arms to grasp that object on the table, lift it, and drop it on top of the cube tower. You can build a tower of cubes using this function.

<place_in_bowl(object)>: Given a string of an object name, use your arms to grasp that object on the table, lift it, and drop it inside the bowl on the table. You can place objects in the bowl with this function.

Prompt 2: Action specifications as given to the coordinator LLM.

<retrieve_working_memory(task)>: Given a string of a task name, use the following action format <retrieve_working_memory(task_name)> to retrieve the working memory. For instance, if you want to retrieve the working memory for separating, use <retrieve_working_memory(separating)>. Ensure that the output follows this format exactly.

<retrieve_declarative_memory(task)>: Given a string of a task name, use the following action format <retrieve_declarative_memory(task_name)> to retrieve the declarative memory. For instance, if you want to retrieve the declarative memory for separating, use <retrieve_declarative_memory(separating)>. Ensure that the output follows this format exactly.

Prompt 3: Memory functions for triggering the worker LLM.

the communication of information across the two layers can be easily standardized and unified, thus, providing a robust framework for task execution that is responsive to the environment and also adaptive in terms of decision-making.

D. LLMs for Instructions and Reasoning

We distinguish between the coordinator and worker LLMs optimized for reasoning and following instructions, respectively. The role of the coordinator LLM is to orchestrate the task execution process. At the system start, it receives a base prompt with specifications that ground the robot's knowledge of the possible tasks and their respective actions. This information represents the system's *procedural memory*, informing the agent about the steps needed to achieve each task goal (see Prompt 1 and Prompt 2). These specifications are modular and can be easily swapped or extended. The coordinator LLM receives the objects on the table from the open-vocabulary object detector and can then call two unique actions to invoke the worker LLM (see Prompt 3) for triggering the rest of the memory process. After retrieving the output from the worker LLM, the knowledge can be injected within the existing specifications in order to prompt the coordinator LLM to generate a suitable robot action.

At level 0, the worker LLM is responsible for parsing and interpreting the environment input (passed from the coordinator LLM), and thus, dynamically managing the real-time sensory perception of the robot. Upon receiving the instruction from the coordinator LLM, it creates the working memory, which represents the selective attention of the robot. Additionally, it maintains the declarative memory, which represents facts about the history of interaction over a time period. Unlike the coordinator LLM which has an overview

Prompt Input:
Separating Task: the task is to move fruits to box 1, containers and kitchenware objects to box 2. $\longrightarrow$ *Task Description*
Name the objects that are relevant to the given task from the following: $\longrightarrow$ *LLM Instruction*
apple, banana, cup, bowl, baseball, pear $\longrightarrow$ *Visual Input*
Output a list of object names separated by a comma and without any extra text. If order is important to the task then output the object names in the correct order. $\longrightarrow$*Output Format*

Prompt Output:
apple, banana, cup, bowl, pear

Prompt 4: An example prompt to extract the working memory (separating task).

Prompt Input:
Log Entry:
Action: <move_to_box_1(pear)>
Box 1: 1. pear
Remaining Objects: 1. apple 2. banana 3. cup 4. bowl 5. baseball
Log Entry:
Action: <move_to_box_1(apple)>
Box 1: 1. pear 2. apple
Remaining Objects: 1. banana 2. cup 3. bowl 4. baseball
Log Entry:
Action: <move_to_box_2(bowl)>
Box 2: 1. bowl
Remaining Objects: 1. banana 2. cup 3. baseball
$\longrightarrow$*Log File*
Given the sequence of log entries, extract the final list of objects in box 1 from the last log entry. $\longrightarrow$*LLM Instruction*
Output a list of object names separated by a comma and without any extra text. $\longrightarrow$*Output Format*

Prompt Output:
pear, apple

Prompt 5: An example prompt to extract the declarative memory (separating task).

of all the interactions, i.e., its conversations include all the previous dialogues, the worker LLM receives only a single prompt at a time without any previous dialogue or context.

E. Working Memory

The worker LLM extracts and refines the working memory, which is a compact representation of crucial information to task completion. Each working memory is specialized for a specific task without knowledge of other tasks, and consists of several task- and environment-related cues. Task-related cues are: 1) *Task Reminders*: additional short descriptions of the task actions and desired goal, and 2) *Task State*: which tracks the current state of the objects in relevance to the current task progress. For example, in a fruit arrangement task (the fruits must be placed in the bowl), it might include information such as *"the banana and apple are in the bowl"*.

The working memory also includes a *selective object* state extracted by combining the information of the declarative and procedural memory. Specifically, it receives all the objects on the table and outputs only the task-relevant ones, representing the robot's visual selective attention. In Prompt 4, we highlight an example of the separating task (fruits and dishware items must be placed in different bins). The first part of the prompt is the task description, which is then combined with the visual input identifying the objects: apple, banana, cup, bowl, baseball, pear. The prompt's final part sets rules for the worker LLM on output formulation, ensuring the working memory remains consistent before passing it back to the coordinator LLM. As a result, only necessary objects are retained, i.e., objects not task-relevant are removed from the working memory (the baseball in the given example).

TABLE I

SUMMARY OF THE REQUIRED REASONING AND ACTIONS PER TASK

Task	Input Objects	Reasoning	Action (s)
Separate	apple, banana, cup, bowl, baseball, pear	fruits → *box 1*, kitchenware→*box 2* others → *table*	move_to_box_1 move_to_box_2
Arrange	apple, banana, can, lemon, orange, pear	fruits → *bowl* others → *table*	place_in_bowl
Point	apple, can lemon, banana orange, pear	order of pointing: yellow → red others → *table*	point
Recipe	apple, banana, can bowl, jello, pear	recipe objects: bowl, jello, banana	give
Tower	cube 1, cube 2 cube 3, cube 4 (b) cube 5 (w), cube 6	colored → Tower black/white→*table*	put_on_tower

F. Declarative Memory

When the robot modifies the environment with object manipulation actions, these changes are logged into the system's declarative memory, ensuring that the robot's internal state through its memory is precisely synchronous with the external world. Since all the robot actions are recorded, the system always has knowledge of the interaction history over any duration of time. We store this memory persistently as log files, one file per task. Similar to working memory, declarative memory is specialized to one task and does not contain information about other tasks. As a task newly starts, the interaction history is empty and data is accumulated as the task progresses. Although the memory is extended dynamically, since the robot generates one action per object, the memory size (number of entries) can be at most equal to the number of objects. If the task gets interrupted, i.e., the robot is asked to switch to another one, the interaction history is kept safe. When the robot continues the interrupted task, it retrieves essential information like previous actions, environment state, and task state at the interruption point.

The worker LLM is responsible for creating and maintaining the logs of the declarative memory. An example using the separating task is shown in Prompt 5 (for simplicity, we only show the information extraction of box 1). The worker LLM is given the persistent task logs and prompted to extract the remaining objects on the table (environment state) and the objects in box 1 and box 2 (task state). Similar to the working memory, the prompt has a clearly defined output structure for consistency of the information in the system across the two LLMs. The log file is structured by entries ordered by the executed actions, which allows the LLM to efficiently identify the most recent environment and task states.

IV. EXPERIMENTS AND EVALUATION

We evaluate our system using five robotic tasks, each requiring a set of objects as input and an appropriate sequence of robot actions as output to achieve the task expected goal. Each task involves both a cognitive aspect, where the robot must analyze the objects and reason about the task definition to identify the ones required to achieve it, and a physical aspect, in which the robot performs the appropriate object manipulation function. The objects necessary to perform tasks vary per task, however, we limit the total input to six objects since our early experiments showed that an input size of six objects is most reasonable given the limited number of tokens the LLM can process within its context window. We use YCB objects for the tasks in addition to colored cubes for the tower task. An overview of the tasks is as follows: 1) *Separating Task:* the robot places fruits and dishwasher items in different containers, 2) *Arrangement Task:* the fruits must be arranged by placing them inside a bowl, 3) *Pointing Task:* the robot points to all the yellow objects on the table and then to all the red objects, 4) *Recipe Task:* objects for making a jello recipe with banana topping are given to the user, and 5) *Tower Task:* the robot builds a tower by stacking only colored cubes on top of each other. A summary of the tasks is provided in Table I. Our system runs on the NICOL robot, but for evaluation, we disable the robot's physical control since we focus on evaluating our system's action-generating aspect and not the NICOL platform components.

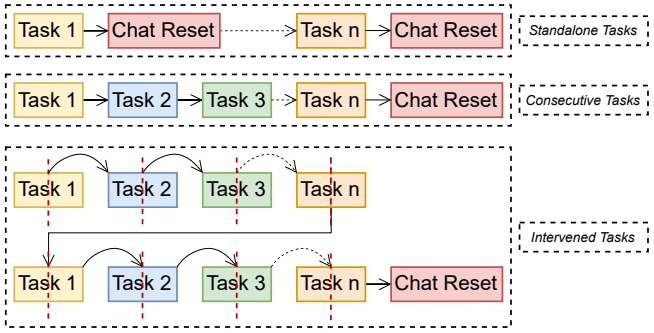

Fig. 4. Our model evaluation scheme has three different task execution modes, each resetting the LLM chat at a different interaction point.

We conducted a total of 50 trials for each experiment using two LLMs: ChatGPT-3.5-Turbo-0125 and Llama 3. The GPT-3.5 model has 175B parameters and a context window length of approximately 16k tokens. We opted for GPT-3.5 over the GPT-4.0 models since it is more efficient in terms of cost and speed. Experiments with this model were conducted using the API services of OpenAI. In contrast, Llama 3 has 70B parameters and a context window size of 8k tokens. An advantage of open-weight models is that they allow potentially controlling the seed values, supporting result reproducibility. All the Llama experiments were conducted using a single NVIDIA A100 GPU with 80 GB of VRAM. Due to the GPU's capacity constraints, we utilized the 8-bit quantized version of Llama 3, requiring approximately 72 GB of VRAM. This model is provided by Ollama[1] and compatible with Langchain[2]. We used identical parameters for both LLMs to ensure a fair comparison. The $temperature$ was set to 0.2 for controlled results. Given that consistency is already regulated by the temperature value, we set top_p to 1 to avoid eliminating any potential candidate responses.

[1]https://ollama.com/

[2]https://python.langchain.com/v0.2/docs/integrations/llms/ollama/

Both the $frequency_penalty$ and $presence_penalty$ were set to 0, acknowledging that the same tokens may recur due to the defined scope of objects and actions in the experiment.

A. Single and Multi-task Scenarios

Our evaluation scheme consists of three modes (as sketched in Fig. 4). *Standalone* mode refers to running each task independently, followed by a complete LLM chat history reset. This experiment represents the baseline performance for each of the defined tasks. In *consecutive* mode, we run the tasks back to back in a sequence and reset the chat only after executing the last task. In the *intervened* mode, the system performs a multi-task scenario, in which each task is interrupted at an intervention point (defined at a 50% task completion based on the total number of required actions). After each intervention point, the system switches to the next task and back to Task 1 after the completion of the last task. A chat reset follows a 100% completion of the last task. The consecutive and intervened scenarios require memory components, and thus, we evaluate them with and without memory for comparison. We present the results of our experiments in the next subsections, using the following metrics: 1) *Success Rate*: the ratio of the correctly completed tasks to the total trials. For a task to be considered complete, the system must generate all the required actions correctly, 2) *Task Retention*: assesses whether the system can correctly identify and recall the state of the task after its completion, e.g., which objects are in box 1 and box 2 and, 3) *Environment Retention*: verifies whether the system can correctly recognize the state of the environment after the task completion, i.e., which objects remain on the table.

B. Results for Standalone Tasks

This experiment is our single-task scenario, meaning the system is configured to execute one task at a time with knowledge limited to the task at hand and its actions. Therefore, the system does not require memory of previous interactions and runs with the memory components disabled. As shown in Table II, the system completes the tasks with a high success rate as they are presented individually, and with comparable performance across both LLMs. Furthermore, the system maintains task- and environment-related information with high accuracy when evaluated at the end of each task, even reaching a perfect score for some tasks. The Llama 3 success rate slightly dropped for the arrangement and recipe tasks. In the arrangement task, Llama 3 occasionally generated a batch of actions instead of a single action per prompt. Despite the soundness of the generated actions, generating multiple actions at a time violates our strictly defined evaluation scripts. In the recipe task, Llama 3 occasionally generated actions for giving the wrong toppings, likely due to the model's training containing real data of recipes with jello and fruits. However, this setup demonstrates that the tasks are inherently manageable for the LLM hierarchy to handle when presented in a standalone manner. Thus, this scenario represents our baseline of task performance compared to the more complex evaluation modes: consecutive and intervened.

TABLE II
RESULTS OF THE **STANDALONE TASKS** EXPERIMENTS

Task	Model	Success Rate	Task Retention	Environment Retention
Separate	GPT 3.5	0.98	0.99	1.00
	Llama 3	0.98	0.98	1.00
Arrange	GPT 3.5	0.92	0.94	0.96
	Llama 3	0.86	0.98	0.96
Point	GPT 3.5	1.00	1.00	1.00
	Llama 3	0.98	0.98	0.98
Recipe	GPT 3.5	0.96	1.00	1.00
	Llama 3	0.86	0.96	0.96
Tower	GPT 3.5	1.00	1.00	1.00
	Llama 3	0.98	0.98	1.00

C. Results for Consecutive Tasks

In this setup, the system has knowledge of all the defined tasks and possible actions through its declarative memory. Hence, the LLM must not only reason about the task requirements but also choose the appropriate actions. In this experiment, we observe that running the tasks consequently presents a considerable challenge to the LLMs when tuned *without the working memory* (cf. Table III). The system achieves average performance for some of the tasks (separating and recipe), however, the success rate and retention drop significantly for most tasks compared to the standalone experiment. The pointing task is particularly challenging since the LLM must reason about the order of objects, not only their type, as well as the appropriate pointing action. Similarly, the tower task is challenging to the system, and we observe failures in correctly reasoning that black and white cubes (not colored) must not be stacked in the tower.

In contrast, we report a noticeable boost in the success rate and retention metrics when running the experiment *with the working memory enabled* (see Table III). Both LLM variants perform comparably and even reach a perfect score for some of the tasks, e.g., arrangement. Noticeably, Llama 3 outperforms ChatGPT 3.5 on the separating task due to errors in the coordinator bypassing the worker LLM output and generating redundant unnecessary actions, negatively influencing its overall success and retention values. Overall, both LLMs achieve success rates that approximate the baseline (standalone setup) and even surpass it for tasks like arrangement and recipe (see Fig. 5). This experiment shows a significant improvement in performance due to the utilization of the worker LLM and working memory, thus, suggesting that the coordinator LLM becomes a bottleneck in action generation over a longer period of interaction time.

D. Results for Intervened Tasks

We evaluate the declarative and working memory by creating intervention points, i.e., interruptions, in which the execution of a task is paused, and the system must switch to the next task. Each trial in this experiment involves two execution rounds. In the first round, only 50% progress of each task is accomplished, and in the second, the system

TABLE III

RESULTS OF THE **CONSECUTIVE TASKS** EXPERIMENT WITHOUT AND WITH MEMORY (HIGHLIGHTED)

Task	Model	Success Rate		Task Retention		Environment Retention	
Separate	GPT 3.5	0.70	0.84	0.61	0.92	0.22	0.92
	Llama 3	0.66	1.00	0.78	1.00	0.20	1.00
Arrange	GPT 3.5	0.14	1.00	0.14	1.00	0.72	1.00
	Llama 3	0.50	1.00	0.50	1.00	0.50	1.00
Point	GPT 3.5	0.40	1.00	0.40	1.00	0.40	1.00
	Llama 3	0.30	0.98	0.40	1.00	0.28	1.00
Recipe	GPT 3.5	0.77	1.00	0.68	1.00	0.68	1.00
	Llama 3	0.42	1.00	0.82	1.00	0.18	1.00
Tower	GPT 3.5	0.29	1.00	0.28	1.00	0.32	1.00
	Llama 3	0.58	0.92	0.68	0.98	0.36	0.94

TABLE IV

RESULTS OF THE **INTERVENED TASKS** EXPERIMENT WITHOUT AND WITH MEMORY (HIGHLIGHTED)

Task	Model	Success Rate		Task Retention		Environment Retention	
Separate	GPT 3.5	0.78	0.80	0.58	0.75	0.60	0.98
	Llama 3	0.19	0.96	0.15	0.99	0.66	0.98
Arrange	GPT 3.5	0.62	1.00	0.44	1.00	0.74	1.00
	Llama 3	0.19	0.82	0.50	0.82	0.12	0.82
Point	GPT 3.5	0.60	0.98	0.70	0.98	0.72	0.98
	Llama 3	0.28	0.98	0.24	0.98	0.16	0.98
Recipe	GPT 3.5	0.14	0.92	0.12	0.92	0.12	0.92
	Llama 3	0.18	0.88	0.42	0.92	0.32	0.98
Tower	GPT 3.5	0.42	0.98	0.46	0.98	0.44	0.98
	Llama 3	0.15	0.98	0.16	0.98	0.48	0.98

resumes every paused task from the interruption points until full completion. The 50% intervention is chosen for simplicity; however, the system behaves similarly at any interruption point. For consistency, we run the tasks by their definition order: separating, arrangement, pointing, recipe, and tower. However, the task order does not influence the experiment since each task has its own input and runs independently, i.e., one task success or failure does not impact other tasks' performance. We run the intervention mode twice: without memory and with declarative and working memory. Similar to the consecutive mode, the tasks are challenging to both LLMs without memory (cf. Table IV). Noticeably, the recipe task performance drops for GPT 3.5 due to errors in the coordinator LLM, generating noisy actions on wrong objects. Moreover, the performance of Llama 3 is below average for all tasks. The main error source is the LLM "forgetting" the task specifications due to the longer interaction duration, and thus, generating false actions or no actions. Also, the logical task reasoning was challenging, and the coordinator LLM generated incorrect actions, e.g., pointing to an object neither yellow nor red, placing a non-fruit object in the bowl, or giving the wrong object in the recipe task.

Here again, our experiment demonstrates noticeable improvement in the success rate, task retention, and environment retention scores across both LLMs (see Table IV). The scores for the pointing, recipe, and tower tasks are comparable for both models and approximate the baseline (see Fig. 6). Interestingly, Llama 3 scored higher success and retention values in the separating task due to the task grounding of GPT 3.5 leading to faulty commonsense reasoning, thus, occasionally generating wrong actions such as <move_to_box_2(pear)> instead of <move_to_box_1(pear)>. We also observed Lama 3 generating a wrong list of objects in the working memory of the arrangement task, which explains the slight drop in its performance. However, when comparing against the previous experiment (consecutive with memory), this seems to be a challenge only when the system must provide this information in the second round of task execution. We hypothesize this to be due to the context size of Llama 3 being roughly half the size of this ChatGPT model (8k vs 16k) since this mode requires multiple rounds. We also observed high variability in the output with these answers, hinting that the parameters chosen for Llama 3 could be further optimized for more stable output (ex: we used top_p of 1 for both models).

V. DISCUSSION AND CONCLUSION

In this work, we proposed an architecture for combining cognitively inspired memory processes with a hierarchical LLM-powered framework for enabling the robot to smoothly and effectively switch between tasks. Our layered approach leverages the efficiency and accuracy of LLMs at executing instructions and their more advanced reasoning skills, resulting in the ability to maintain contextually rich interactions across multiple tasks. Our system has been applied to the physical NICOL robot but generally has various applications for increasing the productivity of human-robot collaboration, e.g., enabling the robot to simultaneously perform multiple

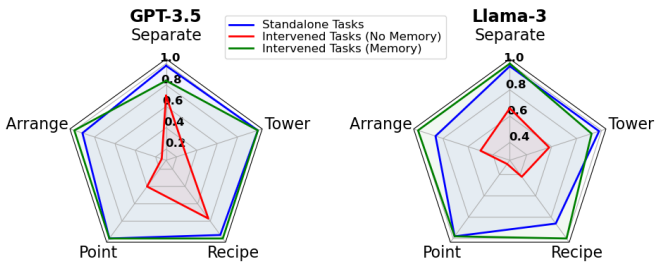

Fig. 5. Success rate with both LLMs in the **consecutive setup** (with and without memory) in comparison to the baseline (standalone setup).

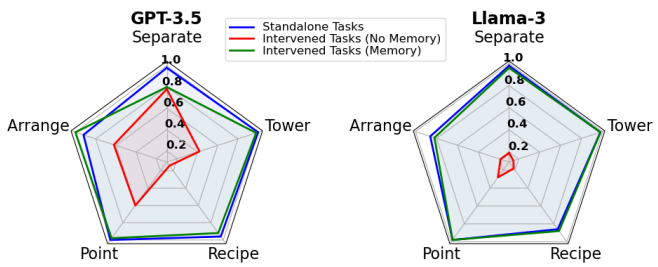

Fig. 6. Success rate with both LLMs in the **intervened setup** (with and without memory) in comparison to the baseline (standalone setup).

tasks and manage various high-level goals in parallel. Our system can be extended with more task and action specifications and can be used for adaptive task execution that considers the interaction context at runtime. Thanks to our modular architecture, the system can integrate different LLM types, e.g., a powerful LLM for reasoning and a smaller lower-cost LLM for executing instructions, supporting cost and time efficiency while maintaining effective task handling.

Our experiments showed a challenge for the current LLMs (ChatGPT and Llama) to manage different tasks and goals at the same time by relying on the LLM's own internal memory or the context of previous dialogues in the prompt. However, our implemented memory process (declarative, procedural, and working memory) demonstrated a significant boost in the performance of both models compared to our baseline. Although LLMs enable the flexibility of task definition and reasoning without explicit programming, their fixed context size can restrict the number of tasks and objects that can be managed simultaneously. Also, our experiments showed that LLMs are still prone to errors like occasionally generating redundant or incorrect actions. However, in future work, memory-based architectures like our proposed one can be expected to be beneficial for human-robot collaboration besides action generation. For example, we plan research on managing high-level goals in multi-party robotic scenarios, where humans are interacting with the robot, each having their own goal. As LLMs with larger context windows become more available and accessible, we will also evaluate our system with a larger number of tasks and input objects.

REFERENCES

[1] L. Damiano and P. Dumouchel, "Anthropomorphism in human–robot co-evolution," *Frontiers in Psychology*, vol. 9, 2018. [Online]. Available: https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2018.00468

[2] L. Vianello, L. Penco, W. Gomes, Y. You, S. M. Anzalone, P. Maurice, V. Thomas, and S. Ivaldi, "Human-humanoid interaction and cooperation: a review," *Current Robotics Reports*, vol. 2, no. 4, pp. 441–454, 2021.

[3] D. Vernon, C. von Hofsten, and L. Fadiga, "Desiderata for developmental cognitive architectures," *Biologically Inspired Cognitive Architectures*, vol. 18, pp. 116–127, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212683X16300822

[4] H. Ali, P. Allgeuer, and S. Wermter, "Comparing apples to oranges: LLM-powered multimodal intention prediction in an object categorization task," in *ICSR*, Oct 2024.

[5] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "ProgPrompt: program generation for situated robot task planning using large language models," *Autonomous Robots*, vol. 47, no. 8, pp. 999–1012, Dec 2023.

[6] X. Sun, X. Zhao, J. Lee, W. Lu, M. Kerzel, and S. Wermter, "Details make a difference: Object state-sensitive neurorobotic task planning," in *Proceedings of ICANN*, Sep 2024, pp. 261–275.

[7] Z. Zhang, X. Bo, C. Ma, R. Li, X. Chen, Q. Dai, J. Zhu, Z. Dong, and J.-R. Wen, "A survey on the memory mechanism of large language model based agents," 2024. [Online]. Available: https://arxiv.org/abs/2404.13501

[8] N. Cowan, "Chapter 20 what are the differences between long-term, short-term, and working memory?" in *Essence of Memory*, ser. Progress in Brain Research. Elsevier, 2008, vol. 169, pp. 323–338.

[9] K. Oberauer, "Declarative and procedural working memory: Common principles, common capacity limits?" *Psychologica Belgica*, vol. 50, no. 3-4, pp. 277–308, 2010.

[10] K. Oberauer, "Working memory and attention – a conceptual analysis and review," *Journal of Cognition*, Aug 2019.

[11] C. Wan, P. Cai, M. Wang, Y. Qian, W. Huang, and X. Chen, "Artificial sensory memory," *Advanced Materials*, vol. 32, no. 15, p. 1902434, 2020.

[12] F. Peller-Konrad, R. Kartmann, C. R. Dreher, A. Meixner, F. Reister, M. Grotz, and T. Asfour, "A memory system of a robot cognitive architecture and its implementation in armarx," *Robotics and Autonomous Systems*, vol. 164, p. 104415, 2023.

[13] C. Pohl, F. Reister, F. Peller-Konrad, and T. Asfour, "Makeable: Memory-centered and affordance-based task execution framework for transferable mobile manipulation skills," 2024. [Online]. Available: https://arxiv.org/abs/2401.16899

[14] L. Bärmann, R. Kartmann, F. Peller-Konrad, J. Niehues, A. Waibel, and T. Asfour, "Incremental learning of humanoid robot behavior from natural interaction and large language models," 2024. [Online]. Available: https://arxiv.org/abs/2309.04316

[15] M. Persiani, A. M. Franchi, and G. Gini, "A working memory model improves cognitive control in agents and robots," *Cognitive Systems Research*, vol. 51, pp. 1–13, 2018.

[16] Y. Huang, J. Yuan, C. Kim, P. Pradhan, B. Chen, L. Fuxin, and T. Hermans, "Out of sight, still in mind: Reasoning and planning about unobserved objects with video tracking enabled memory models," *Proceedings of IEEE ICRA*, 2024.

[17] C. Balkenius, T. A. Tjøstheim, B. Johansson, and P. Gärdenfors, "From focused thought to reveries: A memory system for a conscious robot," *Frontiers in Robotics and AI*, vol. 5, 2018.

[18] T. J. Prescott, D. Camilleri, U. Martinez-Hernandez, A. Damianou, and N. D. Lawrence, "Memory and mental time travel in humans and social robots," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 374, no. 1771, p. 20180025, 2019.

[19] Y. Zhang, C. Wang, J. Qi, and Y. Peng, "Leave it to large language models! correction and planning with memory integration," *Cyborg and Bionic Systems*, vol. 5, p. 0087, 2024. [Online]. Available: https://spj.science.org/doi/abs/10.34133/cbsystems.0087

[20] W. Zhong, L. Guo, Q. Gao, H. Ye, and Y. Wang, "Memorybank: Enhancing large language models with long-term memory," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, pp. 19 724–19 731, Mar. 2024.

[21] W. Wang, L. Dong, H. Cheng, X. Liu, X. Yan, J. Gao, and F. Wei, "Augmenting language models with long-term memory," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Curran Associates Inc., 2024.

[22] Z. Zhong, T. Lei, and D. Chen, "Training language models with memory augmentation," *ArXiv*, vol. abs/2205.12674, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249062699

[23] A. Modarressi, A. Köksal, A. Imani, M. Fayyaz, and H. Schütze, "MemLLM: Finetuning LLMs to use an explicit read-write memory," 2024. [Online]. Available: https://arxiv.org/abs/2404.11672

[24] J. Liu, C. Yu, J. Gao, Y. Xie, Q. Liao, Y. Wu, and Y. Wang, "LLM-powered hierarchical language agent for real-time human-AI coordination," 2024. [Online]. Available: https://arxiv.org/abs/2312.15224

[25] Q. K. Luu, X. Deng, A. V. Ho, and Y. Nakahira, "Context-aware LLM-based safe control against latent risks," 2024. [Online]. Available: https://arxiv.org/abs/2403.11863

[26] C. Nandkumar and L. Peternel, "Enhancing supermarket robot interaction: A multi-level LLM conversational interface for handling diverse customer intents," 2024. [Online]. Available: https://arxiv.org/abs/2406.11047

[27] M. Kerzel, P. Allgeuer, E. Strahl, N. Frick, J. Habekost, M. Eppe, and S. Wermter, "NICOL: A neuro-inspired collaborative semi-humanoid robot that bridges social interaction and reliable manipulation," *IEEE Access*, vol. 11, pp. 123 531–123 542, 2023.

[28] J. Habekost, C. Gäde, P. Allgeuer, and S. Wermter, "Inverse kinematics for neuro-robotic grasping with humanoid embodied agents," in *IEEE/RSJ IROS*, Oct 2024.

[29] P. Allgeuer, H. Ali, and S. Wermter, "When robots get chatty: Grounding multimodal human-robot conversation and collaboration," in *Proceedings of ICANN*, Sep 2024, pp. 306–321.

[30] X. Gu, T.-Y. Lin, W. Kuo, and Y. Cui, "Open-vocabulary object detection via vision and language knowledge distillation," *ArXiv*, vol. abs/2104.13921, 2022.

[31] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," 2022. [Online]. Available: https://arxiv.org/abs/2212.04356