

LLM Tool Workflows for Robot Explainability and Natural Language Commanding

Tai Mai, Ryo Sakagami, Gabriel Quere, George Mesesan, Robert Schuller, Konrad Fründ, Jörn Vogel, Annette Hagengruber, Jinoh Lee, Andreas Dömel, Freck Stulp, Samuel Bustamante

Abstract—Robotics and language models occupy two fundamentally separate domains. Commanding or extracting information from embodied robotic systems typically requires a certain level of proficiency in operating the system’s interfaces, be it interfaces for physical interaction (like controllers and other input methods), or programmatic interfaces for information. This can be a hurdle for anyone unfamiliar with the system. Meanwhile, anyone capable of speaking or writing a supported natural language can interact with language models and use them to request, process or provide information.

To make robotic systems more accessible, we propose *Robotic Tool Workflows* (RTWs) to bridge the gap between an LLM-powered natural language frontend and a robotic backend that is home to pre-existing functionality of the robot. This enables natural language to be an intuitive user interface to both the robot’s physical skills, as well as its internal state and information. We present preliminary insights deploying RTWs on three complex robotic systems; the wheelchair-based mobile manipulator MAYA, the humanoid torque-controlled robot TORO and the autonomous industrial mobile manipulator AIMM. We show how RTWs can be used for robot command, task planning, and requesting information, enabling the systems to be commandable and explainable.

I. INTRODUCTION

Many robotic systems employed today use well-defined and modular cognitive architectures, including, for instance, world models¹ [1] and action representations [2]. While such modules are tightly integrated into systems and have been tried and tested over time, interacting with them (e.g., retrieving explanations or starting a task, as illustrated in Fig. 1), is often challenging as it requires a certain level of expertise with robotic middleware APIs or other low-level interfaces.

Inspired by advances in foundation models such as large language models (LLMs), user frontends based on natural language have emerged as promising alternatives for robot users [3], [4], offering accessible and intuitive ways to interact with a robot. However, to employ LLMs to the robotics use case, they need to be given the ability to interact with all relevant modules in the robot’s backend. In the example in Fig. 1, an LLM would need to either request information from a world model or call internal functions to start a task.

To facilitate this integration, we propose *Robotic Tool Workflows* (RTWs) that bridge the gap between an LLM-powered natural language frontend and a robotic backend that is home to pre-existing functionality of the robot. The key element are so-called *LLM tools*, i.e., code functionalities that enable LLMs to interact with pre-existing functionality, wrapped to

¹We use the term from a robotics perspective: “A world model is an internal representation that reflects relevant parts of the physical world” [1].

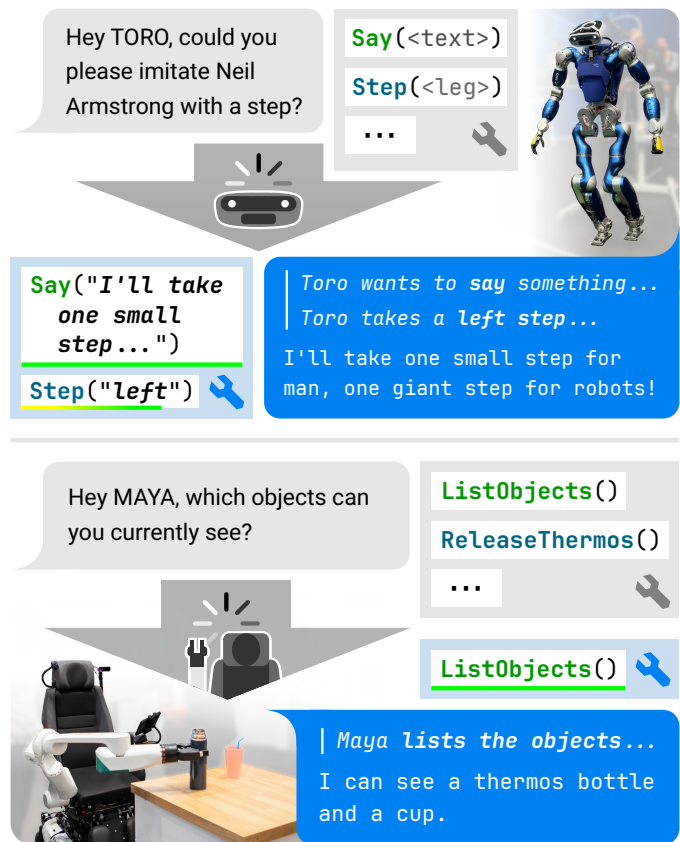


Fig. 1. High-level visualization of a tool-based workflow in robotics. **Top:** Example action request for the humanoid robot TORO. **Bottom:** Example information request for the wheelchair-based robot MAYA. Depending on the choice of tool implementation, the underlying LLM can also be given the capability to choose appropriate parameters of parameterized tools, e.g., to choose the appropriate number of steps mentioned in the user’s query.

abstract the inner logic away from the LLM and documented in a human-readable way that tells the LLM the intended usage of that tool. To increase oversight, tools are deployed in pre-defined code paths called workflows [5], enabling controlled access to the flexibility of LLMs. As illustrated in Fig. 1, this enables natural language to be an intuitive user interface to both the robot’s physical skills, as well as its internal state and information.

The goal of this paper is to report on our experience deploying RTWs on three complex robotic systems, each with its own pre-existing modules and thus with tools adapted to their use-case: the wheelchair-based mobile manipulator MAYA, the humanoid torque-controlled robot TORO, and the autonomous industrial mobile manipulator AIMM. To summa-

size our contribution, (i) we introduce the concept of RTWs, and provide implementational details for practitioners; (ii) we describe the RTW for the three robots above; and (iii) we show a set of demonstrations beyond the examples in Fig. 1, illustrating how RTWs can be used for intuitively commanding the robots, as well as requesting and providing information to their pre-existing internal models.

II. ROBOTIC TOOL WORKFLOWS

A. Background: LLM Tools

Tools are a means to impart upon LLMs a level of agency, extending their capabilities beyond just language output. In our implementation, LLM tools are self-documenting Python classes² that tell the LLM *when* and *how* to use the tool, serialized into a standardized JSON format.

Example: Consider a toy use case of determining which one of two decimal numbers is the larger one. Numbers with decimal points have in the past been a weakness of LLMs, resulting in LLMs responding $0.19 > 0.2$ because $19 > 2$. By alleviating language models from having to do math and providing a Python function with which to compare the two numbers, the LLM’s responsibility is then simplified to just identifying the two numbers of interest from the query and then parameterizing the tool with those numbers. The task of actually comparing the numbers is completed using a Python function inside `CompareNumbers.execute()`. We note that the tool’s logic for comparing the numbers is excluded from its serialization, and thus withheld from the LLM. All the relevant information that it needs to complete the task is given through descriptive documentation strings.

```
import pydantic
import typing

class CompareNumbers(pydantic.BaseModel):
    """
    Use this tool to compare the value of two numbers.
    """
    num_1: typing.Annotated[float, "1st number to compare"]
    num_2: typing.Annotated[float, "2nd number to compare"]

    def execute(self):
        if self.num_1 == self.num_2:
            print("They're equal.")
        else:
            smaller_num: float = min(self.num_1, self.num_2)
            bigger_num: float = max(self.num_1, self.num_2)
            print(f"{smaller_num} is smaller than {bigger_num}.")
```

Sparing the LLM all the implementation details and lower-level logic can prevent context cluttering and eventually hallucinations, as well as speed up the interaction. This pattern becomes critical when the `.execute()` methods contain lots of non-trivial logic that is not instrumental for the LLM in its simple task of choosing and parameterizing the tool.

B. Using Tools in Robotic Workflows

To adapt LLM tools to our robotics use case, we implement our communication logic with the robotic backend within the `.execute()` methods. A helpful design pattern for the

²Inspired by previous work [6], we implement the tools as Pydantic classes, allowing type-checking and parameter validation. This also facilitates automatic error handling, by feeding error messages back to the LLM in the case that its chosen parameterizations are invalid.

roboticist is to encapsulate lower-level communication logic with the robotic backend within a robot interface, which can be passed to the `.execute()` method as a parameter via dependency injection.

```
class RoboticTool(pydantic.BaseModel):
    ...
    def execute(self, robot_interface):
        ...
        robot_interface.encapsulated_logic()
    ...
```

This has the additional benefit of allowing multiple LLM tools to cleanly share robotic functionalities with each other without code duplication. We utilize LLM tools in the form of *workflows*, where tool calls and their execution follow pre-defined code paths [5]. This paradigm affords the developer a level of predictability and control over the actions triggered by the LLM.

The most basic general interaction loop between user and LLM starts with the user giving a query to the LLM³. The LLM is then given a list of tools from which it must choose one to answer the query. When the LLM fails to select one of these tools correctly, or parameterizes a tool incorrectly, an error message is raised/caught and passed to the LLM, requiring it to self-correct. After the LLM has correctly selected a tool (and parameterized it, if applicable), the `.execute()` method is called, executing whatever logic the developer has implemented therein.

While this blueprint is common among our demonstrators, we implement further steps to fit the robot’s individual use cases, such as tool hierarchies, tools generated (or enabled) dynamically from the context, and robot-specific guardrails. These are described with the specific demonstrators below.

C. Demonstrations with Three Robots

1) **TORO:** TORO [9], [10] (Fig. 2) is a humanoid robot with legs and hands capable of passive walking, stair climbing, and manipulating objects. Furthermore, its high mobility allows it to complete human-oriented tasks and to make expressive gestures. It is used as a platform for terrestrial and extra-terrestrial surface applications.

a) *Tool Workflow:* TORO implements the aforementioned basic common LLM interaction loop and serves as a baseline. The features that are accessible to the LLM involve executing simple pre-planned motions without variation, as well as reproducing static information about itself, mainly for demonstration purposes. This demonstrator served as a proof-of-concept for our approach, which we expanded further on the other demonstrators.

b) *Example Tools:* As seen in Fig. 1 and Fig. 2 (center), TORO can `TakeAStep(leg, x, y, yaw)`, where `leg` \in `{"left", "right"}`, `x` \in `[-0.15m, +0.15m]`, `y` \in `[-0.1m, +0.1m]`, `yaw` \in `[-45°, +45°]` are parameters the LLM can set freely, which get validated at runtime. We also depict in Fig. 2 other tools used in TORO’s demonstration at the 2025 Automatica trade fair, such as waving or celebrating a football goal (the latter with an optional parameter to celebrate in the style of Cristiano Ronaldo, a popular demo at trade fairs).

³We note that we utilize on-premises (local) LLMs exclusively. Our demonstrations use either the Mistral Small (e. g., [7]) or the Qwen & Qwen-VL (e. g., [8]) families.

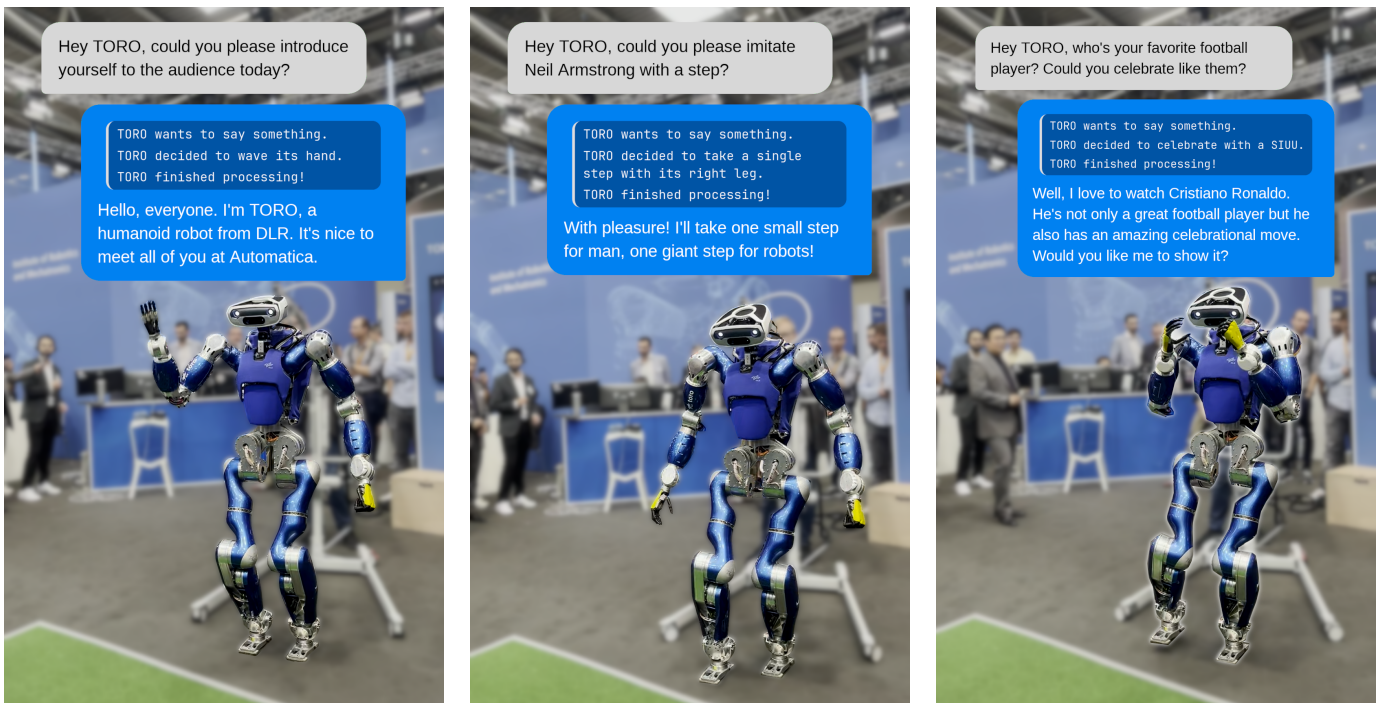


Fig. 2. TORO exhibit at the 2025 Automatica trade fair. We highlight that the choice of tools and parameters is chosen by the LLM from around 17 tools.

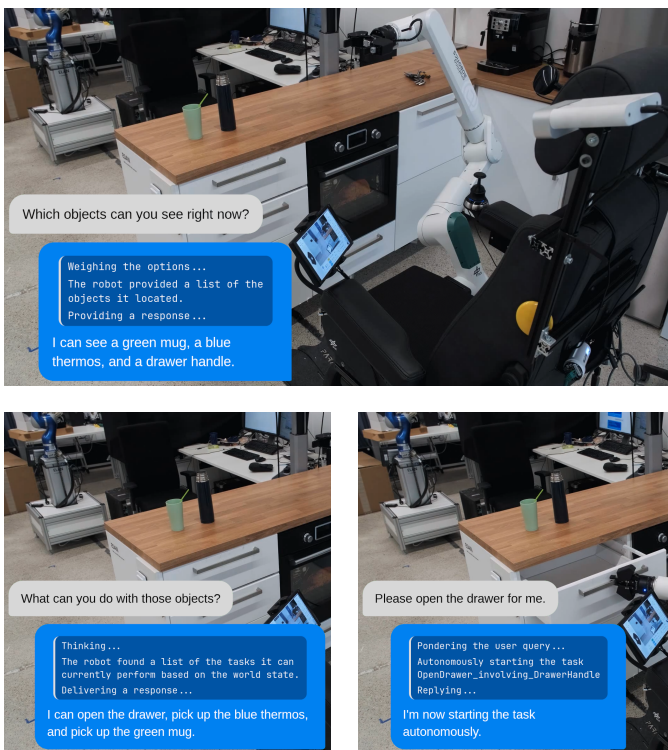


Fig. 3. MAYA relays information from its world model and skill engine, then opens a drawer upon request.

2) **MAYA**: MAYA (Fig. 3), like its predecessor EDAN [11], is a wheelchair-based mobile manipulator featuring a robotic arm and gripper, designed to help wheelchair-bound and motor-impaired users complete everyday tasks around the household, such as opening doors and drawers, and providing

drinks. With cameras and object detection, MAYA can recognize known objects in its environment and add them to its world model [12]. Depending on which objects are currently present in the robot's world model, the robot's skill inference engine exposes the skills to interact with those objects in the form of shared control skills [2] through the robotic middleware. These skills can then either be executed by the user in shared control, or triggered to be executed in supervised autonomy with an automaton [13].

a) *Tool Workflow*: While TORO's tools allow it to respond with basic, static information and pre-planned motions, MAYA's also enable the retrieval of dynamic information and the execution of tasks that are dependent on the current state of the world. Since the list of available skills is dynamic, the tools cannot be given to the LLM as a static list. Therefore, (1) we generate and enable LLM tools dynamically from the robot's available skills at runtime; and (2) as the dynamic list of tools can grow large and potentially confusing to the LLM, we opt not to present all tools to the LLM as a flat list, as has been done in recent works [14], but rather hierarchically. One example is first grouping the tools into *information tools* and *task tools*: Once the LLM has completed the first step of classifying the user's query as either an information request or an action request, it is presented with the expanded list of available tools from only the relevant category. Finally, we are able to keep the conversation history, enabling the LLM to refer to previous messages.

b) *Example Tools*: General information requests can be made regardless of the state of the world, such as `ExplainSafetyFeatures` and `ListAvailableTasks`, the latter illustrated in Fig. 3. As for the dynamically created task tools, commonly demonstrated tasks executed with supervised autonomy include `OpenDrawer` (Fig. 3), `OpenCabinet`, and

PickMug, among others. Finally, WhyNoObjectsOrTasks can give useful information for troubleshooting.

3) **A IMM**: The Autonomous Industrial Mobile Manipulator (A IMM) [15] is a system focusing on autonomous logistic tasks in partially unstructured environments such as factories and laboratories. To achieve complete autonomy in such cluttered environments, A IMM employs a navigation system, object detectors for both known and unknown objects, a motion planner for collision-free manipulation, a flow-control system to orchestrate the behavior, and a robotic world model named A IMM-WM [16] to represent the world relevant to all of the other software components. A IMM-WM is a tree-based world model, having nodes that represent both concrete (e. g., boxes and trays) and abstract objects (e. g., approaches and grasps) and directed edges that represent a 6D rigid body transformation between nodes [16].

a) *Tool Workflow*: The goal is to enable operators to request and provide⁴ information from the A IMM-WM. As with MAYA, tools for A IMM are categorized hierarchically. The LLM is first instructed to decide between pre-existing Getters (for retrieving information) and Setters (for updating information) from the world model’s API. If none of these are deemed sufficient to answer the query, the LLM is presented with a new set of tools for generating queries in the underlying database’s query language Cypher. Generating queries through these tools are done with query templates to guardrail the generation and ensure integrity of the database. Once again, keeping the chat history allows for natural interactions that refer to previous conversation turns (Fig. 4).

b) *Example Tools*: Using the world model API’s getter and setter tools, the operator can ask the LLM to `GetObjectPoses(object_name)`, `GetEmptyStorages()`, `AddObject(object_name, parent_name)` and `SetObjectProperty(object_name, property_name, value)`, among others. The LLM can also fall back to generating guarded Cypher queries in order to `RetrieveNodes(node_type, match_attribute, return_attribute)` or `RetrieveParentOrChild(...)` with analogous parameters in order to retrieve information from single nodes or node tuples. Parameterizations by the LLM are validated against the database’s schema. This allows operators to ask for properties of objects (such as weight and color), as well as object relations (such as “in” or “on”).

D. Discussion

Tool workflows are implemented as pre-defined and curated paths, where each iteration within the interaction loop fundamentally functions the same as the next one. We follow different designs, tailored to each robot’s use-case. For instance, MAYA’s task tools are dynamic and their availability is dependent on the state of the world. Therefore, the LLM is only allowed to call a single tool per iteration, as a second tool might not be available anymore after the execution of the first. In contrast, iterations in TORO’s interaction loop are somewhat liberal in the sense that the LLM can decide to execute more

⁴This has an architectural advantage that the knowledge provided by operators can contribute to the system autonomy without additional overhead.

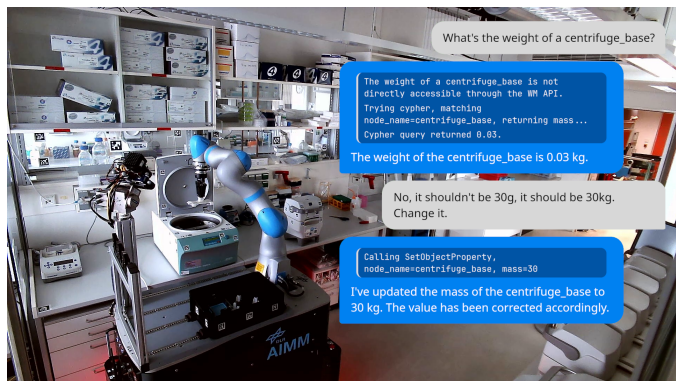


Fig. 4. The A IMM system in a medical laboratory, performing PCR assay tasks. The LLM interaction (right) was created from a digital twin of the robot’s world model recorded in a similar setting.

than one action, and we permit a number of actions until a pre-defined limit.

Our implementation of RTWs takes advantage of how tool workflows can easily facilitate guardrails around the LLM. The intent and decision making always lies with the user and programmer, while the LLM is only responsible for translating that intent from natural language to robotic functions. This philosophy aims to maximize predictability and reliability, which is especially valuable for mission critical applications. However, this approach limits the potential of LLMs to do more intelligent decision making of their own. In more agentic approaches (e. g., [17]), LLMs are not guardrailed by interaction loops but rather given the freedom to call conditional chains of tools, where human intervention is less necessary. While that is more conducive to autonomy, tool workflows allow us to prioritize predictability and safety with our physical robots. To explore both approaches, we intend to compare tool workflows with agentic approaches for our robotics use case in future work, carrying out experiments on the real robots to evaluate flexibility, robustness in instruction following, and safety.

III. RELATED WORK

Our robotic tool are akin in philosophy to Model Context Protocols (MCP [18]), a widely adopted community standard to interface LLMs to with real databases and web APIs. We are also inspired by work in the robotics community that provide a natural language interface with tools from the Robot Operating System (ROS), such as the AI-powered agent ROSA [3], the robotic MCP software RoboNeuron [19], the robotic explainability system NealAI [20], and the skill adaptation system for manufacturing IROSA [14].

Similarly to our approach, other work [21] has been done to hierarchically interface with a robot’s skill library using natural language but with the addition of VLMs for object grounding, whereas our object grounding is rooted in a world model shared with other robotic modules. While our implementations of RTWs serve as a self-documenting natural language interface to existing skills, LLM-Cap2Skill [22] aim to generate the code for robotic skills from existing capability ontology descriptions and middleware API documentation.

REFERENCES

- [1] R. Sakagami, F. S. Lay, A. Dömel, M. J. Schuster, A. Albu-Schäffer, and F. Stulp, “Robotic world models – conceptualization, review, and engineering best practices,” *Frontiers in Robotics and AI*, vol. 10, November 2023.
- [2] G. Quere, A. Hagenhuber, M. Iskandar, S. Bustamante, D. Leidner, F. Stulp, and J. Vogel, “Shared control templates for assistive robotics,” in *2020 ICRA*, 2020, Conference Proceedings, pp. 1956–1962.
- [3] R. Royce, M. Kaufmann, J. Beckett, S. Moon, K. Carpenter, K. Pak, A. Towler, R. Thakker, and S. Khattak, “Enabling novel mission operations and interactions with rosa: The robot operating system agent,” 2025. [Online]. Available: <https://arxiv.org/abs/2410.06472>
- [4] S. Bustamante, M. Knauer, J. Thun, S. Schneyer, A. Albu-Schäffer, B. Weber, and F. Stulp, “Raccoon: Grounding embodied question-answering with state summaries from existing robot modules,” in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025, pp. 4322–4329.
- [5] E. Schlutz and B. Zhang. (2024) Building effective agents. <https://www.anthropic.com/engineering/building-effective-agents>. Engineering at Anthropic blog. [Online]. Available: <https://www.anthropic.com/engineering/building-effective-agents>
- [6] J. Liu. How to use pydantic for llms: Schema, validation & prompts. Pydantic. [Online]. Available: <https://pydantic.dev/articles/llm-intro>
- [7] M. AI, “Mistral-small-3.2-24b-instruct-2506,” <https://docs.mistral.ai/models/model-cards/mistral-small-3-2-25-06>, 2025.
- [8] S. Bai, Y. Cai, R. Chen, K. Chen, X. Chen, Z. Cheng, L. Deng, W. Ding, C. Gao, C. Ge *et al.*, “Qwen3-v1 technical report,” *arXiv preprint arXiv:2511.21631*, 2025.
- [9] J. Engelsberger, A. Werner, C. Ott, B. Henze, M. A. Roa, G. Garofalo, R. Burger, A. Beyer, O. Eiberger, K. Schmid *et al.*, “Overview of the torque-controlled humanoid robot toro,” in *IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2014, pp. 916–923.
- [10] G. Mesesan, R. Schuller, J. Engelsberger, M. A. Roa, J. Lee, C. Ott, and A. Albu-Schäffer, “Motion planning for humanoid locomotion: Applications to homelike environments,” *IEEE Robotics & Automation Magazine*, vol. 32, no. 1, pp. 35–48, 2025.
- [11] J. Vogel, A. Hagenhuber, M. Iskandar, G. Quere, U. Leipscher, S. Bustamante, A. Dietrich, H. Höppner, D. Leidner, and A. Albu-Schäffer, “Edan: An emg-controlled daily assistant to help people with physical disabilities,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4183–4190.
- [12] A. Hagenhuber, G. Quere, M. S. Z. Iskandar, S. Bustamante Gomez, J. Feng, D. Leidner, A. O. Albu-Schäffer, F. Stulp, and J. Vogel, “An assistive robot that enables people with amyotrophy to perform sequences of everyday activities,” *Scientific Reports*, vol. 15, no. 1, p. 8426, March 2025. [Online]. Available: <https://elib.dlr.de/214291/>
- [13] S. Bustamante, G. Quere, K. Hagmann, X. Wu, P. Schmaus, J. Vogel, F. Stulp, and D. Leidner, “Toward seamless transitions between shared control and supervised autonomy in robotic assistance,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3833–3840, 2021.
- [14] M. Knauer, S. Bustamante, T. Eiband, A. Albu-Schäffer, F. Stulp, and J. Silvério, “Irosa: Interactive robot skill adaptation using natural language,” *IEEE Robotics and Automation Letters*, vol. 11, no. 4, p. 5246–5253, Apr. 2026. [Online]. Available: <http://dx.doi.org/10.1109/LRA.2026.3671560>
- [15] A. Dömel, S. Kriegel, M. Kaßecker, M. Brucker, T. Bodenmüller, and M. Suppa, “Toward fully autonomous mobile manipulation for industrial environments,” *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, 2017.
- [16] R. Sakagami, A. Dömel, P. Lehner, S. Riedel, S. G. Brunner, A. Albu-Schäffer, and F. Stulp, “A tree-based world model for reducing system complexity in autonomous mobile manipulation,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 9, no. 7, pp. 6680–6687, 2024.
- [17] A. L. Zhang, T. Kraska, and O. Khattab, “Recursive language models,” 2026. [Online]. Available: <https://arxiv.org/abs/2512.24601>
- [18] Anthropic, “Introducing the model context protocol,” Website, Nov 2024, accessed: 2026-04-23. [Online]. Available: <https://www.anthropic.com/news/model-context-protocol>
- [19] W. Guan, Q. Hu, H. Xi, C. Zhang, A. Li, and J. Cheng, “Roboneuron: A middle-layer infrastructure for agent-driven orchestration in embodied ai,” 2026. [Online]. Available: <https://arxiv.org/abs/2512.10394>
- [20] P. G. Knestel, L. Mayershofer, A. Köpken, N. Batti, F. S. Lay, T. Bachmann, S. Brinkman, J. Butterfass, E. den Exter, T. H. Ehlert, W. Friedl, T. Gumpert, X. Luo, A. N. Manaparampil, T. Mai, A. Raffin, F. Schmidt, A. Schmidt, D. Seidel, L. E. Schürmann, N. Wenzinger, B. Weber, A. O. Albu-Schäffer, A. S. Bauer, D. Leidner, R. Luz, P. Schmaus, F. Stulp, T. Krüger, S. Bustamante Gomez, and N. Y.-S. Lii, “Nealai: A rag-based chat assistant supporting command of a robotic team from the iss,” in *2025 International Conference on Space Robotics, iSpaRo 2025*. IEEE, March 2026. [Online]. Available: <https://elib.dlr.de/218946/>
- [21] B. Nemeč, M. Simonič, B. Kuster, L. Žlajpah, and A. Ude, “Towards using natural language to perform robotic tasks,” in *Advances in Service and Industrial Robotics*, K. Jovanović, A. Rodić, and M. Raković, Eds. Cham: Springer Nature Switzerland, 2025, pp. 3–11.
- [22] L. M. V. Da Silva, A. Köche, N. König, F. Gehlhoff, and A. Fay, “Capability-driven skill generation with llms: A rag-based approach for reusing existing libraries and interfaces,” in *2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2025, pp. 1–8.