IROSA: Interactive Robot Skill Adaptation using Natural Language

Markus Knauer[1,2] (*IEEE Student Member*), Samuel Bustamante[1,2], Thomas Eiband[1], Alin Albu-Schäffer[1,2] (*IEEE Fellow*), Freek Stulp[1] and João Silvério[1] (*IEEE Member*)

***Abstract*—Foundation models have demonstrated impressive capabilities across diverse domains, while imitation learning provides principled methods for robot skill adaptation from limited data. Combining these approaches holds significant promise for direct application to robotics, yet this combination has received limited attention, particularly for industrial deployment. We present a novel framework that enables open-vocabulary skill adaptation through a tool-based architecture maintaining a protective abstraction layer between the language model and robot hardware. Our approach leverages pre-trained LLMs to select and parameterize specific tools for adapting robot skills without requiring fine-tuning or direct model-to-robot interaction. We demonstrate the framework on a 7-DoF torque-controlled robot performing an industrial bearing ring insertion task, showing successful skill adaptation through natural language commands for speed adjustment, trajectory correction, and obstacle avoidance while maintaining safety, transparency, and interpretability.**

***Index Terms*—Foundation Models, Imitation Learning**

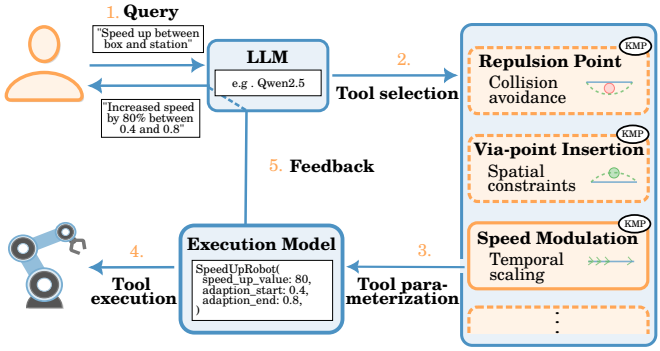

Fig. 1: Overview of our approach **I**nteractive **RO**bot **S**kill **A**daptation using natural language. Showing the interactive selection and parameterization of a tool by a LLM based on a user query leading to a skill adaptation via the used execution model. Some of the tools we are providing are shown. "Respond to User" is a general tool, whereas "Repulsion Point", "Via-Point Insertion" and "Speed Modulation" are specific tools to adapt KMPs.

I. Introduction

ROBOTICS applications increasingly demand flexible, adaptive systems that can be easily reconfigured for varying tasks and environments. Large language models (LLMs) offer significant potential for making robot skill adaptation more accessible to non-expert users, enabling intuitive control through everyday language rather than specialized programming. Our aim is to apply this capability to industrial tasks, where interpretable, verifiable, and predictable robot behaviors are essential. This requires an approach that preserves established control principles while enabling natural language adaptation.

We address this challenge through a tool-based architecture maintaining strict separation between language understanding and robot control. Our approach builds on *function calling* [1]–[3], where language models select and parameterize predefined functions based on natural language input rather than directly generating outputs or actions. We provide LLMs with a well-defined toolbox of modification primitives, where each tool represents a validated, parameterized function that modifies an underlying motion generation model. The LLM selects and parameterizes tools based solely on their textual descriptions, enabling zero-shot adaptation without retraining or fine-tuning.

We map these adaptation primitives to a trajectory generation framework flexible enough to accommodate diverse modifications while maintaining deterministic, interpretable behavior. We realize this through Kernelized Movement Primitives (KMPs) [4]—a non-parametric probabilistic imitation learning framework providing temporal and spatial adaptation capabilities through start-, end-, and via-points (see Section III). Rather than allowing LLMs direct access to robot hardware or trajectory generation, our framework constrains the language model to select and parameterize well-defined, validated robot control functions.

This design contrasts with end-to-end approaches that train models directly on sensorimotor data. While such approaches (see Section II) demonstrate impressive capabilities in some domains, they face challenges in meeting the interpretability, verifiability, and safety compliance requirements of industrial applications. Our approach leverages pre-trained LLMs combined with established probabilistic motion representations, providing a cost-effective, explainable, and immediately de-

Manuscript received: November, 14, 2025; Revised February, 2, 2026; Accepted February, 27, 2026. This paper was recommended for publication by Editor Jens Kober upon evaluation of the Associate Editor and Reviewers' comments. This work was partially funded by the DLR project "ASPIRO"; the European Union's Horizon Research and Innovation Program under Grant 101136067 (INVERSE); the Federal Ministry for Economic Affairs and Climate Protection with DARP funds based on a decision by the German Bundestag and by the European Union - NextGenerationEU; and partially supported by the German Federal Ministry of Research, Technology and Space (BMFTR) under the Robotics Institute Germany (RIG).

[1] All authors are with the German Aerospace Center (DLR), Institute of Robotics and Mechatronics (RMC), Münchener Str. 20, 82234 Weßling, Germany. {first}.{last}@dlr.de

[2] Markus Knauer, Alin Albu-Schäffer and Samuel Bustamante are also with the School of Computation, Information and Technology (CIT), Technical University of Munich (TUM), Arcisstr. 21, 80333 Munich, Germany. m.knauer@tum.de

Digital Object Identifier (DOI): 10.1109/LRA.2026.3671560

ployable solution. Our key contributions (Section IV) are:

1) A tool-based architecture that enables zero-shot natural language adaptation of robot skills through structured function calling, maintaining strict separation between language understanding and robot control.
2) Novel extensions to KMPs for natural language-driven speed modulation and obstacle avoidance via repulsion fields, expanding the adaptation capabilities beyond traditional via-point constraints.
3) Experimental validation (Section V) on a 7-DoF torque-controlled DLR SARA robot [5] performing industrial manipulation tasks, demonstrating reliable and explainable skill adaptation without model fine-tuning or iterative user feedback.

II. Related work

Probabilistic representations of movement primitives: Gaussian Mixture Models (GMMs) [6] capture uncertainty but struggle with via-point adaptation after learning. Probabilistic Movement Primitives (ProMPs) [7] enable via-point adaptation but require basis function definition, becoming cumbersome in high-dimensional spaces. Kernelized Movement Primitives (KMPs) [4] overcome these limitations through a non-parametric kernel-based approach that trivially accommodates via-point constraints. Task-Parameterized KMPs (TP-KMPs) [8] further enable adaptation based on task-relevant coordinate frames. Recently, Knauer *et al.* [9] introduced an interactive incremental learning framework combining TP-KMPs with kinesthetic human feedback. This approach enables real-time skill modification through physical human-robot interaction, where users guide the robot's end-effector to demonstrate desired trajectory corrections. While kinesthetic feedback excels at fine-grained trajectory adjustments, it becomes cumbersome for larger-scale modifications or when specifying abstract behavioral changes such as "avoid the obstacle" or "move faster."

End-to-End Language-Conditioned Control: Recent approaches explore direct integration of language conditioning into robot control policies. CLIPORT [10] combines CLIP-based semantic understanding with transporter networks for language-conditioned manipulation. KITE [11] extends this through keypoint-conditioned policies that enable fine-grained manipulation based on object features. LaTTe [12] follows an *end-to-end* approach, employing a pre-trained transformer to directly map natural language instructions to trajectory modifications. While LaTTe achieves fast inference (< 1 second), it is limited to instruction patterns seen during training. Vision-language-action (VLA) models represent the current state-of-the-art. OpenVLA [13] trains large-scale policies on diverse robot datasets for generalized manipulation, while RoboPoint [14] predicts spatial affordances based on natural language commands. While these approaches demonstrate impressive capabilities, their direct application to robot control faces significant challenges. Particularly, they require extensive training data, lack interpretability, and provide limited safety guarantees for industrial settings [15]–[17].

Modular Language-Robot Interfaces: Modular architectures decouple natural language understanding from low-level motion generation, enabling independent validation of each component. These state-of-the-art solutions differ in *how* the LLM interfaces with the robot control layer. OLAF [18] follows a *policy retraining* approach, separating language understanding from visuomotor policy execution. The LLM generates policy update signals rather than direct trajectory modifications, but this requires *retraining* the policy network after each verbal correction. Sharma et al. [19] decouple language processing from motion planning by learning a mapping from language to cost function modifications. However, this requires *offline training*, and cost modifications are applied before replanning rather than enabling real-time adaptation. Merlo et al. [20] separate vision-based plan generation from LLM-based replanning, using the LLM for common-sense reasoning about potential failure modes. The LLM operates at the *action/task level* rather than the trajectory level, adjusting symbolic plans rather than continuous waypoints. Yu et al. [21] separate LLM-based reward specification from policy optimization, where the LLM generates reward function parameters that are then optimized via MPC in simulation. While this improves stability, it *requires simulation-based optimization* rather than direct trajectory adaptation. OVITA [22] follows a *code generation* approach, separating language understanding from trajectory execution through LLM-generated Python code that modifies waypoints. While flexible, this inherits *code generation reliability issues*, requires *cloud-based LLM APIs*, and relies on *iterative user feedback* for validation. Diffusion models have also been explored for language-based skill adaptation [23]. In contrast, KMPs provide analytical uncertainty expressions and require fewer demonstrations to learn.

Positioning of Our Approach: The modular approaches above share common limitations: OLAF and Sharma et al. require retraining or offline learning after corrections; Merlo et al. operate at the symbolic action level rather than on continuous trajectories; Yu et al. require simulation-based optimization loops; and OVITA's code generation introduces reliability issues and cloud dependency. Our *tool-based* approach addresses these limitations by combining LLMs with Kernelized Movement Primitives, leveraging KMPs' flexibility to adapt learned skills to new situations through via-points and temporal modulation. The LLM is constrained to selecting and parameterizing predefined tools, providing safety guarantees through validated tool functions. This enables direct trajectory-level adaptation without retraining, simulation, or iterative code validation, while supporting local LLM deployment for offline industrial environments. A summary comparing our approach with related work is shown in Table I.

III. Preliminaries

Our approach begins with a set of demonstrations $\mathcal{D} = \{\{\boldsymbol{s}_{h,m}, \boldsymbol{\xi}_{h,m}\}_{h=1}^{H}\}_{m=1}^{M}$, where:

- $\boldsymbol{s} \in \mathbb{R}^{\mathcal{I}}$ represents the input variable (typically time or phase variable)
- $\boldsymbol{\xi} \in \mathbb{R}^{\mathcal{O}}$ represents the output variable (e.g., end-effector position, joint angles)
- $\mathcal{I}$, $\mathcal{O}$ denote the dimensions of input and output spaces
- M is the number of demonstrations

TABLE I: Comparison of Language-Conditioned Trajectory Adaptation Approaches. Training-Free: no task-specific (re)training required. Open Vocabulary: handles arbitrary natural language instructions. Zero-shot: no iterative user feedback required to validate model output. Explainable: adaptation steps are traceable and interpretable. Local LLM: operates with locally-deployed language models.

Method	Approach	Training-Free	Open Vocab.	Zero-Shot	Explainable	Local LLM
OLAF [18]	Policy retraining	×	∼	×	×	✓
LaTTe [12]	End-to-end	×	×	✓	×	✓
Merlo [20]	Action Replan.	✓	✓	×	✓	∼
Yu [21]	Reward Synth.	×	✓	×	∼	×
OVITA [22]	Code Gen.	✓	✓	×	✓	×
Ours	Tool-based	✓	✓	✓	✓	✓

- H is the trajectory length (number of datapoints per demonstration)

Following established LfD approaches [4], [6], [7], we extract the relationship between $\boldsymbol{s}$ and $\boldsymbol{\xi}$ from demonstrations obtained through kinesthetic teaching.

Kernelized Movement Primitives (KMPs) [4] provide a probabilistic approach for learning robot skills from limited human demonstrations. The key advantage of KMPs for our application is their ability to learn meaningful skill representations from small datasets (2–5 demonstrations) while enabling principled trajectory modifications through constraint incorporation. From the demonstrations $\mathcal{D}$, a KMP encodes the skill as a *reference trajectory distribution* $\boldsymbol{D} = \{\boldsymbol{s}_n, \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n\}_{n=1}^{N}$ comprising N Gaussians with mean and covariance parameters, computed for inputs $\boldsymbol{s}_{n=1,\dots,N}$ using Gaussian Mixture Models (GMMs). For a query input $\boldsymbol{s}^*$ (typically time), the expectation and covariance of the predicted output $\boldsymbol{\xi}(\boldsymbol{s}^*)$ are given by Eq. (1)–Eq. (2)

$$\mathbb{E}\left[\boldsymbol{\xi}(\boldsymbol{s}^*)\right] = \boldsymbol{k}^* \left(\boldsymbol{K} + \lambda_1 \boldsymbol{\Sigma}\right)^{-1} \boldsymbol{\mu}, \tag{1}$$

$$\operatorname{cov}\left[\boldsymbol{\xi}(\boldsymbol{s}^*)\right] = \alpha \left(\boldsymbol{k}^{**} - \boldsymbol{k}^* \left(\boldsymbol{K} + \lambda_2 \boldsymbol{\Sigma}\right)^{-1} \boldsymbol{k}^{*\top}\right), \tag{2}$$

where $\boldsymbol{K} = [\hat{\boldsymbol{k}}(\boldsymbol{s}_1)^\top, \dots, \hat{\boldsymbol{k}}(\boldsymbol{s}_N)^\top]$, $\boldsymbol{k}^* = \hat{\boldsymbol{k}}(\boldsymbol{s}^*)$, with $\hat{\boldsymbol{k}}(\boldsymbol{s}_n) = [\boldsymbol{k}(\boldsymbol{s}_n, \boldsymbol{s}_1), \dots, \boldsymbol{k}(\boldsymbol{s}_n, \boldsymbol{s}_N)]$, $\boldsymbol{k}^{**} = \boldsymbol{k}(\boldsymbol{s}^*, \boldsymbol{s}^*)$, $\boldsymbol{k}(\boldsymbol{s}_n, \boldsymbol{s}_{n'}) = k(\boldsymbol{s}_n, \boldsymbol{s}_{n'})\boldsymbol{I}$, where $\boldsymbol{I}$ is an identity matrix, and $k(\boldsymbol{s}_n, \boldsymbol{s}_{n'})$ represents a scalar kernel function. The mean and covariance vectors are $\boldsymbol{\mu} = \left[\boldsymbol{\mu}_n^\top\right]_{n=1}^{N}$ and $\boldsymbol{\Sigma} = \operatorname{blockdiag}\left(\boldsymbol{\Sigma}_n\right)_{n=1}^{N}$, respectively, while λ_1, λ_2, and α are hyperparameters controlling regularization and scaling. A key property of KMPs is their ability to incorporate new constraints through via-points. From Eqs. (1) and (2), if the covariance $\boldsymbol{\Sigma}_n$ is small for a particular $\boldsymbol{\mu}_n$, the predicted expectation at $\boldsymbol{s}_n$ will be close to $\boldsymbol{\mu}_n$. This property enables principled trajectory modulation: to ensure that the predicted trajectory passes through a desired point $\bar{\boldsymbol{\mu}}$ at input $\bar{\boldsymbol{s}}$, one can add the via-point triplet $\{\bar{\boldsymbol{s}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}}\}$ to the reference distribution, provided that $\bar{\boldsymbol{\Sigma}}$ is sufficiently small. This modification forces Eq. (1) to closely match $\bar{\boldsymbol{\mu}}$ while reducing the prediction uncertainty in Eq. (2). We leverage KMPs' kernel formulation that trivially accommodates via-point constraints to enable our tool-based framework to seamlessly incorporate natural language-specified trajectory modifications without requiring explicit basis function definitions.

IV. Approach

Our approach enables users to adapt robot skills through natural language commands, supporting three primary adaptation types relevant for industrial tasks: **Speed Modulation** enables temporal adjustments to trajectory execution (*e.g.* "move faster before reaching the box," "slow down by 50%"), addressing varying task requirements such as adapting to different production speeds or ensuring careful approach to delicate objects. **Via-point Insertion** provides spatial trajectory modifications (*e.g.* "move 10cm to the left," "approach from above"), enabling users to correct trajectories based on changed workspace layouts or to avoid interference with other equipment. **Repulsion Point Generation** supports collision avoidance through obstacle awareness (*e.g.* "avoid the blue box," "stay away from the red object"), accommodating dynamic workspace changes where new obstacles appear or existing objects are relocated.

These adaptations are motivated by common industrial scenarios where robots must accommodate variations in object positions, production rates, and workspace configurations without requiring expert reprogramming.

A. Problem Formulation

Our goal is to adapt a robot skill based on natural language instructions while preserving the demonstrated skill structure and ensuring safe, predictable robot behavior. Given a learned KMP $\boldsymbol{D}$ from demonstrations $\mathcal{D}$ (see Section III), the system receives: (i) the **initial KMP** $\boldsymbol{D}$ encoding the demonstrated skill, (ii) a **user instruction** L_{instruct} in natural language, (iii) **environment observations** $\mathcal{E} = \{\boldsymbol{p}_i, \boldsymbol{d}_i, \ell_i\}_{i=1}^{N_{\text{obj}}}$ where N_{obj} is the number of objects, $\boldsymbol{p}_i \in \mathbb{R}^3$ is the object position, $\boldsymbol{d}_i \in \mathbb{R}^3$ are object dimensions, and ℓ_i is the object semantic label, and (iv) an optional **environment description** E_d in natural language. In the current implementation, environment observations are provided manually, which is common in structured industrial environments. Integration with automatic perception systems (*e.g.* pose estimation [24]) is straightforward. The KMP framework supports full 6-DoF manipulation, encoding both position and orientation; our evaluation focuses on position adaptations, with an orientation modification example demonstrated in the supplementary video. The system modifies the KMP's internal trajectory representation according to L_{instruct} and $\mathcal{E}$ through the five-step workflow detailed below.

B. Five-Step Workflow

The system processes commands through the workflow illustrated in Fig. 1: (1) User Query, (2) Tool Selection, (3) Tool Parameterization, (4) Tool Execution, and (5) Feedback.

1) User Query: The robot executes the learned skill using the KMP $\boldsymbol{D}$. Users observe the execution and provide natural language instructions L_{instruct} via voice commands or text input when adaptation is desired. These instructions can be simple commands (*e.g.* "move faster") or more complex specifications (*e.g.* "slow down by 50% before reaching the box"). The system also receives environment observations $\mathcal{E}$ and optional environment description E_d to provide context for instruction interpretation.

2) Tool Selection: The LLM analyzes the user instruction L_{instruct} and environment observations $\mathcal{E}$ to select appropriate tool(s) $T^* \in \mathcal{T}$ from the available tool set $\mathcal{T}$. This selection leverages the LLM's native function calling capabilities without requiring task-specific training. All available tools are serialized into JSON schemas containing their name, description, and parameter specifications with type annotations. This follows the standard JSON schema-based tool definition format widely adopted across LLM APIs [1]–[3]. These schemas are sent to the LLM alongside the user's natural language command, and the LLM directly returns the selected tool(s) with instantiated parameters in a single inference step, leveraging the LLM's internal semantic understanding to match user intent with tool descriptions.

Tool Descriptions: Each tool includes a description specifying its function and parameters. For example, the `SpeedUpRobot` tool is defined as:

```python
SpeedUpRobot(speed_up_value: int, adaption_start: float,
             adaption_end: float):
    """
    Detailed tool description for the LLM.
    Containing parameter description. E.g.
    :param speed_up_value: percentual speedup of the robot.
    ...
    """

    # Type and range validation for all parameters
    if adaption_start >= adaption_end:
        raise ValidationError("adaption_start must be less than
                               adaption_end")
        ...

    #Tool execution
    try:
        #kmp: KMPWrapper is a global variable
        return kmp.change_predicting_frequency(speed_up_value,
                                    adaption_start, adaption_end)
    except Exception as e:
        raise ExecutionError("Error speeding up robot: " + str(e))
```

Listing 1: Example tool definition with natural language description

These descriptions serve as the interface between human intent and machine execution, allowing the LLM to match user requests with appropriate tools.

3) Tool Parameterization: Once the appropriate tool T^* is selected, the LLM extracts parameters θ^* for the selected tool based on L_{instruct}, $\mathcal{E}$, and the tool description. For example, if the user command is "slow down by 50% before reaching the box", the LLM parameterizes the `SpeedUpRobot` tool with: `speed_up_value`$=50$ (indicating slowdown), `adaption_start`$=0.0$, and `adaption_end` determined based on the box position in $\mathcal{E}$.

Trajectory segment determination follows a two-step process: (1) the LLM identifies the relevant spatial references from the user's request (*e.g.* "box" and "station"); (2) inside the tool function, the system iterates over the predicted trajectory and returns the closest time point for each referenced object (within the hard-coded proximity threshold d_{prox}). When multiple objects are specified (*e.g.* "between box and station"), these time points define t_{start} and t_{end}. This constrains the LLM to semantic understanding while delegating numerical computation to deterministic algorithms inside the tool.

Parameter validation provides multiple safety layers to ensure tool execution remains within safe operational bounds. These validation checks are implemented within each tool's execution method: *type validation* verifies that extracted parameters match expected data types (*e.g.* numerical values for speed adjustments), and *range validation* confirms parameters fall within pre-defined safe operational bounds (*e.g.* speed changes within $\pm 200\%$ of baseline). When validation fails, the system provides specific feedback to guide user refinement, maintaining transparent operation.

4) Tool Execution: Once the tool T^* and parameters θ^* are validated, the tool executes and modifies the KMP's internal representation: $\boldsymbol{D} \leftarrow T^*(\boldsymbol{D}, \theta^*)$. The KMP model serves as the foundation for our tool-based interface, providing three primary adaptation mechanisms that leverage the inherent flexibility of KMPs:

- **Speed Modulation:** Adjusts the execution speed of trajectory segments by modifying the time intervals between trajectory points. The tool parameters are $\theta = \{\boldsymbol{s}_{\text{start}}, \boldsymbol{s}_{\text{end}}, \gamma\}$: trajectory segment start/end (normalized time inputs) and percentage speed change. Given a baseline trajectory with time steps δ_{th} between consecutive points at normalized time inputs $\boldsymbol{s}_h \in [0, 1]$, we scale the time intervals within a segment $[\boldsymbol{s}_{\text{start}}, \boldsymbol{s}_{\text{end}}]$ by a factor $f(\gamma)$:

$$\delta'_{th} = \begin{cases} \delta_{th} \cdot f(\gamma) & \text{if } \boldsymbol{s}_{\text{start}} \leq \boldsymbol{s}_h < \boldsymbol{s}_{\text{end}} \\ \delta_{th} & \text{otherwise} \end{cases} \tag{3}$$

$$f(\gamma) = \begin{cases} \frac{|\gamma|+100}{100} & \text{if } \gamma \geq 0 \\ \frac{100}{|\gamma|+100} & \text{if } \gamma < 0 \end{cases} \tag{4}$$

  Here $\gamma > 0$ indicates slowdown and $\gamma < 0$ indicates speedup. For example, $\gamma = 50$ increases time intervals by $1.5\times$, slowing execution by 50%, while $\gamma = -50$ decreases intervals to $0.67\times$, speeding up by 50%. Points outside the specified segment maintain their original timing. This enables commands such as "move faster before reaching the box" or "slow down by 50%," while preserving the spatial characteristics of the learned skill.
- **Via-point Insertion:** Adds via-points $\{\bar{\boldsymbol{s}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}}\}$ to steer the trajectory through desired regions (see Section III). The parameters $\theta = \{\mathbf{p}_i, s_i, \gamma_\Sigma\}$ specify via-point positions, normalized time inputs, and covariance scaling. The number of via-points is determined by the desired minimum dwell time at the target, a hard-coded system parameter. This supports commands like "move more to the left" or "approach from above".
- **Repulsion Point Generation:** Introduces repulsive constraints that encourage the robot to avoid specified regions in the workspace. The tool parameters are $\theta = \{\boldsymbol{c}, r_{\text{obs}}, \delta_{\text{safe}}\}$: obstacle center position, radius, and safety margin. Given an obstacle with center $\boldsymbol{c} \in \mathbb{R}^3$ and radius r_{obs}, we define a signed distance field (SDF) [25]

$$d(\boldsymbol{p}) = \|\boldsymbol{p} - \boldsymbol{c}\| - r_{\text{obs}}, \tag{5}$$

  where $\boldsymbol{p} \in \mathbb{R}^3$ is a point in workspace. While we use a spherical SDF for simplicity, the SDF representation enables arbitrary obstacle shapes, allowing exact object geometries to be incorporated when available, as in our experiments (Section V-D). The predicted trajectory is checked for collision at discrete inputs $\boldsymbol{s}_h$, where $\mathbb{E}[\boldsymbol{\xi}(\boldsymbol{s}_h)]$ denotes the expected position from the KMP

at input $\boldsymbol{s}_h$ (Eq. (1)): if $d(\mathbb{E}[\boldsymbol{\xi}(\boldsymbol{s}_h)]) < \delta_{\text{safe}}$ for safety margin δ_{safe}, we compute corrected positions $\boldsymbol{x}'(\boldsymbol{s}_h)$ by pushing points outside the obstacle:

$$\boldsymbol{x}'(\boldsymbol{s}_h) = \boldsymbol{c} + \frac{\mathbb{E}[\boldsymbol{\xi}(\boldsymbol{s}_h)] - \boldsymbol{c}}{\|\mathbb{E}[\boldsymbol{\xi}(\boldsymbol{s}_h)] - \boldsymbol{c}\|} \cdot (r_{\text{obs}} + \delta_{\text{safe}}). \quad (6)$$

Via-points are placed at regular intervals along the affected trajectory segment: for all indices where $\|\mathbb{E}[\boldsymbol{\xi}(\boldsymbol{s}_h)] - \boldsymbol{x}'(\boldsymbol{s}_h)\| > \varepsilon_{\text{thresh}}$ with threshold $\varepsilon_{\text{thresh}}$, we add via-points $\{\bar{\boldsymbol{s}}_h = \boldsymbol{s}_h, \bar{\boldsymbol{\mu}}_h = [\boldsymbol{x}'(\boldsymbol{s}_h), \boldsymbol{q}(\boldsymbol{s}_h)], \bar{\boldsymbol{\Sigma}}_h\}$ to the KMP. The number of via-points depends on the collision segment length; we select representative points to ensure smooth trajectory deformation while keeping modifications localized. This leverages the via-point property of KMPs (Section III) to generate collision-free trajectories, enabling commands like "avoid the blue box".

These adaptation mechanisms leverage the flexibility of KMPs to incorporate new constraints while maintaining the learned skill structure, ensuring that adaptations remain consistent with the original demonstrations. Due to space constraints, we only present the key functionalities, but other functions were also implemented, including hyperparameter setting via natural language.

5) Feedback and Iterative Refinement: After the modified KMP is executed by the robot, the system provides feedback to the user and awaits further instructions. This enables iterative refinement where users can observe the adapted behavior and provide additional commands to fine-tune the skill. The robot executes the modified skill using the updated $\boldsymbol{D}$ and the cycle returns to Step 1, allowing continuous adaptation based on user feedback.

Key advantages of this architecture include: **(1) Safety through Abstraction:** the LLM never directly controls robot trajectories but interfaces only with well-defined, deterministic tools that implement safe modifications to the underlying model, helping mitigate potential issues from LLM hallucinations while ensuring predictable robot behavior. **(2) Interpretability and Transparency:** unlike end-to-end learned policies where decision-making lacks transparency, our system provides clear traceability through explicit tool selection and parameter extraction, allowing users to observe which tool was selected, what parameters were extracted, and why validation succeeded or failed. **(3) Modular Design:** the system employs a modular architecture with separate functions for different capabilities, loading only relevant tools at runtime to improve efficiency. **(4) Customization:** users can easily adapt the system by defining custom tools for specific applications, replacing the underlying control model, or extending the interface to support different modalities such as simulation environments.

C. Complete Adaptation Workflow

The complete workflow is formalized in Algorithm 1, integrating tool-based architecture, tool selection and validation, and KMP adaptation tools. We denote the LLM as $\mathcal{L}$, which maps natural language instructions to tool selections and parameters. The algorithm shows how the LLM selects and parameterizes well-defined tools that modify the KMP's internal trajectory representation.

Algorithm 1 *LLM-based trajectory adaptation with natural language feedback*

Input: $\boldsymbol{D} = \{\boldsymbol{s}_n, \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n\}_{n=1}^{N}$ from $\mathcal{D}$, LLM $\mathcal{L}$, tool set $\mathcal{T}$, environment $\mathcal{E}$
while user interaction continues **do**
  $\boldsymbol{\xi}(\boldsymbol{s}) \leftarrow$ Predict$(\boldsymbol{D})$ via Eqs. (1)–(2) ▷ Execute trajectory
  Receive L_{instruct} from user ▷ Natural language instruction
  $T^* \leftarrow \mathcal{L}_{\text{select}}(L_{\text{instruct}}, \mathcal{E}, \mathcal{T})$ ▷ Select tool from $\mathcal{T}$
  $\theta^* \leftarrow \mathcal{L}_{\text{param}}(L_{\text{instruct}}, \mathcal{E}, T^*)$ ▷ Extract parameters
  if Validate$(\theta^*, \boldsymbol{D})$ fails **then** ▷ Type, range, workspace, physics checks
    Provide feedback and **continue**
  $\boldsymbol{D} \leftarrow T^*(\boldsymbol{D}, \theta^*)$ ▷ Apply tool modification:
  ▷ Speed: $\delta'_{th} = \delta_{th} \cdot f(\gamma)$ for $\boldsymbol{s} \in [\boldsymbol{s}_{\text{start}}, \boldsymbol{s}_{\text{end}}]$
  ▷ Via-point: $\boldsymbol{D} \leftarrow \boldsymbol{D} \cup \{\bar{\boldsymbol{s}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}}\}$
  ▷ Repulsion: Add via-points $\{\boldsymbol{s}_h, \boldsymbol{x}'(\boldsymbol{s}_h)\}$ where $d(\mathbb{E}[\boldsymbol{\xi}(\boldsymbol{s}_h)]) < \delta_{\text{safe}}$
Output: Adapted $\boldsymbol{D}$ for robot execution

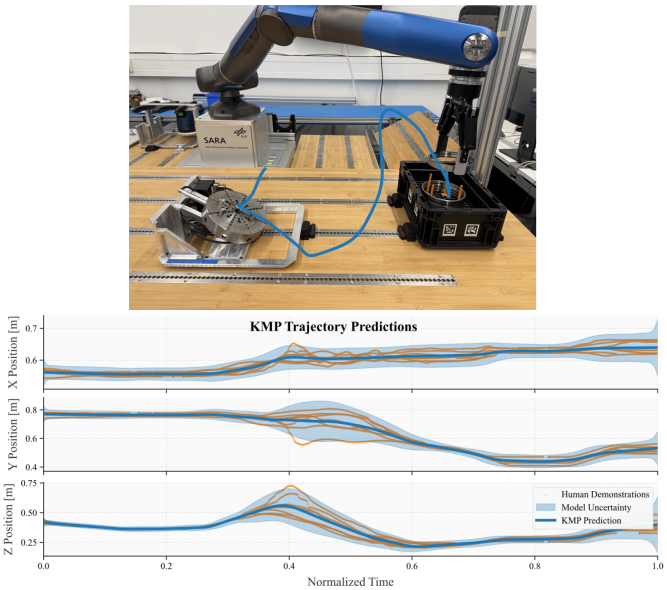

Fig. 2: Demonstration and prediction analysis for the pick-and-insert task. (Top) Trajectory in robot frame. (Bottom) Demonstrations and KMP model predictions with uncertainty visualization.

V. Evaluation

A. Evaluation on real robot

We evaluate our approach on a torque-controlled 7-DoF robot in an industrial scenario where an inner bearing ring must be transferred from its box to a measurement device on a workbench. We use Qwen2.5-VL-72B-Instruct as the LLM backend. We provide $M = 6$ demonstrations for training the KMP model (Fig. 2). We use a time-driven representation with

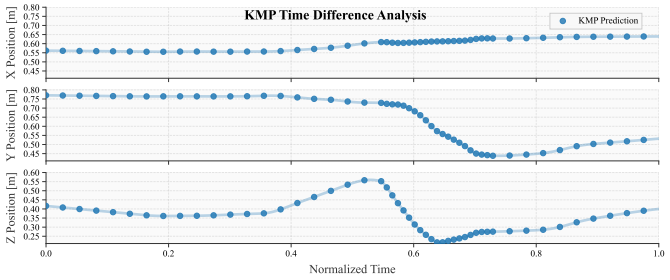

Fig. 3: Speed adaptation results showing temporal trajectory modification through natural language commands. The plot shows the adapted trajectories following command "slow down between box and station," demonstrating precise temporal control while preserving spatial trajectory characteristics.

$\boldsymbol{s}_{h,m} = t_{h,m}/T_m$, where t is a time step, and learn the end-effector position $\boldsymbol{\xi}_{h,m} = \boldsymbol{x}_{h,m}$. We map all the inputs to the interval $[0, 1]$ through division by the duration of each demonstration T_m to easily re-scale skill duration. The KMP is initialized from a GMM with 12 components and $H = 500$ inputs. We chose a Matérn kernel ($\nu = 5/2$) with length scale $l = 0.1$, noise variance 1.0 [26], and KMP hyperparameters $\lambda_1 = 0.1$, $\lambda_2 = 1$, $\alpha = 1$, chosen empirically. The experiments are also shown in the accompanying video.

Evaluation Metrics: We assess our system using *Command Success Rate (CSR)*, *Interpretation Success Rate (ISR)*, and *Task Completion Rate (TCR)*—representing the percentage of executions without failures, correctly interpreted commands, and successfully completed tasks, respectively—alongside *Response Time*, measuring average overall adaptation time.

B. Experiment 1: Speed adaptation via natural language

This experiment evaluates the system's ability to modify execution speed through natural language commands. The robot initially performs the pick-and-insert task at a baseline speed learned from demonstrations. The user provides the command "slow down between box and station". The environment observations $\mathcal{E}$ include object positions and semantic labels, provided manually. Dense-pose estimation [24] could alternatively be used for automatic perception. Using the trajectory segment determination method described in Section IV with proximity threshold $d_{\text{prox}} = 2$ cm, the system identifies the temporal boundaries for adaptation.

The system applies the speed modulation tool (see Section IV) to adjust the time intervals δ_{th} between time steps $t = 0.55$ and $t = 0.72$, corresponding to the phase after picking up the ring and before reaching the measurement station. As seen in Fig. 3, the adapted trajectory successfully slows down in the specified region while preserving the spatial characteristics, demonstrating robust natural language understanding for temporal skill adaptation. Our approach achieved perfect performance on this task with 100% CSR, ISR, and TCR across all test variations (Tab. III).

C. Experiment 2: Trajectory correction through language feedback

After the baseline demonstrations, we introduce a new object to the workspace: a camera positioned to the left

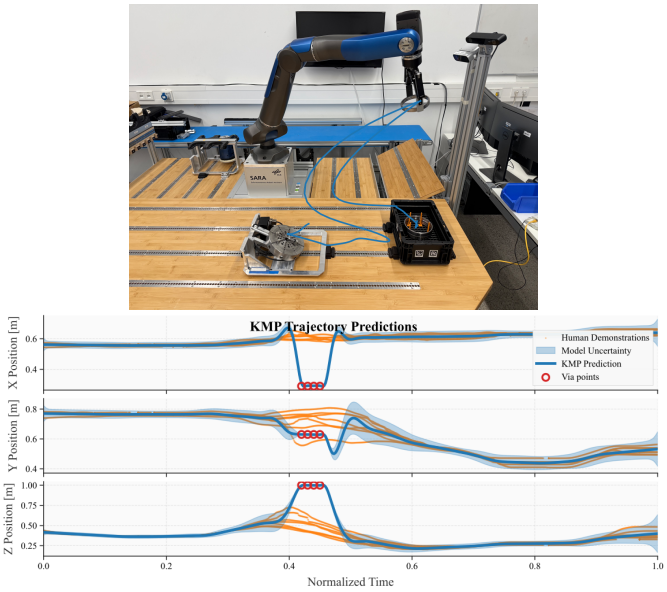

Fig. 4: Trajectory adaptation through natural language command showing (top) experimental setup with new object (camera) placement and (bottom) resulting KMP trajectory modification with via-point insertion to reach the new object while maintaining task completion.

of the robot (visible in Fig. 4, top). The user provides the command "Check the ring with the camera on the left". The LLM identifies the camera object from $\mathcal{E}$, extracting its position $\boldsymbol{p}_{\text{camera}}$ and semantic label. Using the trajectory segment determination method described in Section IV, the system determines when the ring is picked up. The via-point insertion tool (see Section IV) then places a via-point at the camera location, with temporal placement between this identified pickup time and the insertion at the measurement station. Fig. 4 (bottom) shows the adapted trajectory with the via-point at the camera location. The robot successfully executes the modified trajectory, demonstrating that the system can incorporate new task requirements while maintaining the learned skill structure. Our approach achieved 100% CSR and TCR, with 80% ISR (Tab. III). The reduced ISR occurred when the LLM additionally applied speed modulation alongside via-point insertion, which, while functional, was not explicitly requested.

D. Experiment 3: Avoiding novel obstacles

This experiment evaluates the system's ability to adapt trajectories to avoid obstacles introduced to the workspace after the initial demonstrations. A blue industrial box is placed next to the box where the robot picks up the bearing ring, creating a collision hazard with the original learned trajectory (visible in Fig. 5, top). The user provides the command "Please avoid the blue box". The LLM identifies the obstacle from $\mathcal{E}$, extracting its position $\boldsymbol{p}_{\text{obstacle}}$, dimensions $\boldsymbol{d}_{\text{obstacle}}$, and semantic label (provided manually). The repulsion point generation tool constructs a signed distance field (SDF) representing the obstacle volume from its dimensions (see Section IV for SDF formulation). The SDF is modeled using the box dimensions ($15 \times 22.5 \times 23.5$ cm) with safety margin $\delta_{\text{safe}} = 2$ cm. The tool evaluates the predicted trajectory for potential collisions and

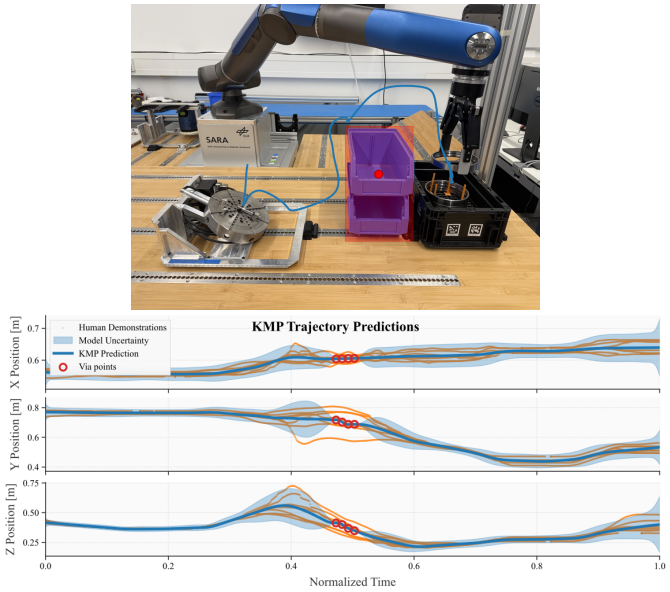

Fig. 5: Obstacle avoidance through natural language command showing (top) experimental workspace with introduced obstacle and (bottom) KMP trajectory modification using repulsion points. The system generates collision-free paths while maintaining the overall task structure and successfully completing the pick-and-insert operation.

inserts via-points along the affected segment where corrections are needed, generating a collision-free path that passes over the blue box. Fig. 5 (bottom) shows the adapted trajectory with the generated repulsion points. The robot successfully avoids the obstacle while completing the task, achieving 100% CSR, ISR, and TCR (Tab. III).

E. Comparison with OVITA

We compare our approach with OVITA [22], a recent open-vocabulary method for trajectory adaptation. While both systems enable language-driven trajectory modifications, they differ fundamentally in their underlying representations and adaptation mechanisms. We evaluate OVITA in two configurations: (1) the original implementation using cloud-based LLMs (we use Gemini), and (2) a modified version using the same local LLM deployed in our approach (Qwen2.5-VL-72B-Instruct) to ensure fair comparison under identical language understanding capabilities. This dual evaluation isolates the contributions of the underlying trajectory representation and adaptation mechanism from differences in LLM performance. The results are shown in Tabs. II and III. OVITA with cloud-based LLMs achieved performance consistent with its original publication, demonstrating effective code generation across all scenarios. However, transitioning to a local LLM revealed significant performance degradation, particularly in speed adaptation and trajectory correction. Low ISR values indicate that generated code successfully modified trajectories but failed to match user intent. Degraded TCR values reflect substantial deviations from task-critical waypoints, preventing successful task completion. In contrast, our tool-based approach maintains consistent performance with the same local LLM, demonstrating the robustness of structured tool interfaces over code generation.

TABLE II: Comparison with OVITA baseline

Method	**Time (s)**
OVITA - LLM: Gemini (cloud)	72.1
OVITA - LLM: Qwen2.5-VL-72B-Instruct (local)	27.1
IROSA - LLM: Qwen2.5-VL-72B-Instruct (local)	**15.4**

TABLE III: Quantitative comparison across experiments (10 prompts × 5 runs each). Prompts collected from public demonstration visitors and colleagues. Full prompt set available in our repository.

Experiment	**OVITA (cloud)**			**OVITA (local)**			**IROSA (local)**		
	CSR	**ISR**	**TCR**	**CSR**	**ISR**	**TCR**	**CSR**	**ISR**	**TCR**
Speed Adapt.	**100.0**	**100.0**	**100.0**	70.0	10.0	30.0	**100.0**	**100.0**	**100.0**
Traj. Correct.	90.0	40.0	30.0	40.0	10.0	0.0	**100.0**	**80.0**	**100.0**
Obst. Avoid.	80.0	40.0	30.0	60.0	60.0	0.0	**100.0**	**100.0**	**100.0**

VI. Discussion

A. Analysis of Results

Our experiments demonstrate that users can effectively modify robot skills through natural language without specialized training. The system interprets diverse commands, from relative ("faster") to absolute ("20% faster"), and translates them into appropriate KMP modifications while preserving skill structure. Compared to OVITA, our approach achieves superior performance across all metrics (CSR, ISR, TCR) while reducing response time by 43% when using the same local LLM, demonstrating that structured tool interfaces outperform code generation for skill adaptation.

B. Scalability Analysis and Performance Considerations

We tested our approach with different local LLM models (*Qwen2.5-Omni-7B*, *Qwen2.5-VL-72B-Instruct*, and *Pixtral-12B-2409*) and experienced the following points:

Tool set scaling faces context length constraints when exceeding 15 tools with smaller LLMs (*e.g.* 8K context models), where extensive tool descriptions can exceed prompt limits and lead to processing errors. Our modular approach addresses this by dynamically loading only task-relevant tools, maintaining performance while staying within context boundaries. Larger models can handle more tools simultaneously, indicating that scaling limitations are primarily determined by the LLM's context capacity rather than our architectural design. **Language model scaling** benefits from larger models' improved natural language understanding capabilities. However, the tool-based interface provides consistent behavior across different model sizes, as tool selection relies on semantic matching rather than model-specific training. This design enables system deployment with various LLM backends without architecture modifications.

C. Flexibility-Safety Trade-off

Our tool-based approach trades flexibility for safety and predictability. OVITA's code generation offers greater flexibility for open-ended adaptations (*e.g.* "execute a spiral") that our predefined tools cannot express. Our approach targets industrial scenarios where certification and operator trust are paramount. New tools can expand capabilities while preserving safety guarantees. For instructions requiring multiple

modifications (*e.g.* "slow down near the station and avoid the blue box"), the LLM can select multiple tools from a single command; however, the system is limited to compositions of available tools. We observed cases where the LLM applied additional modifications not explicitly requested. While these remained within safe bounds, they represent unintended behavior. Stricter prompt engineering and post-processing filters that flag multi-tool responses for user review can mitigate this. Parameter validation within tools and retry mechanisms handle edge cases where the LLM produces invalid outputs.

D. Limitations and Future Work

While our approach demonstrates promising results, several areas warrant further investigation. The current system relies on pre-defined tool functions, which constrains adaptability to user intents not captured by the existing tool set. Future work could explore automatic tool generation or more flexible function composition mechanisms to broaden system capabilities. The framework currently focuses on adapting a single given skill through natural language commands but does not address selection and switching between different skills or primitives. Future research should investigate natural language-based skill selection and composition mechanisms to enable broader robot behavior adaptation. The interpretability of natural language feedback for end-users was not experimentally evaluated; assessing whether users can verify their intent from feedback alone remains future work. The modular architecture enables choosing whether physical interaction [9] or natural language is used to achieve skill adaptations. Future research should investigate which modality is better suited for different scenarios and user groups, and how to effectively combine both approaches.

VII. Conclusion

We presented a novel framework for natural language-based robot skill adaptation using a tool-based architecture that maintains strict separation between language understanding and robot control. The framework leverages local, pre-trained LLMs without additional training, interfacing with Kernelized Movement Primitives through validated tool functions to enable speed adjustments, trajectory corrections, and obstacle avoidance through everyday language. Experimental validation on a 7-DoF robot performing industrial manipulation demonstrates effective interpretation of diverse commands while preserving skill structure and ensuring predictable execution.

References

[1] OpenAI, "Function calling and other API updates," https://openai.com/index/function-calling-and-other-api-updates/, 2023.

[2] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, 2023.

[3] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, and M. Sun, "ToolLLM: Facilitating large language models to master 16000+ real-world APIs," in *Int. Conf. on Learning Representations (ICLR)*, 2024.

[4] Y. Huang, L. Rozo, J. a. Silvério, and D. G. Caldwell, "Kernelized movement primitives," *Int. J. Robot. Res. (IJRR)*, vol. 38, no. 7, pp. 833–852, 2019.

[5] M. Iskandar, C. Ott, O. Eiberger, M. Keppler, A. Albu-Schäffer, and A. Dietrich, "Joint-level control of the DLR lightweight robot SARA," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[6] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 37, no. 2, pp. 286–298, 2007.

[7] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 26, 2013, pp. 2616–2624.

[8] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.

[9] M. Knauer, A. Albu-Schäffer, F. Stulp, and J. Silvério, "Interactive incremental learning of generalizable skills with local trajectory modulation," *IEEE Robot. Autom. Lett. (RA-L)*, vol. 10, no. 4, pp. 3398–3405, 2025.

[10] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Proc. 5th Conf. Robot Learning (CoRL)*, 2021.

[11] P. Sundaresan, S. Belkhale, D. Sadigh, and J. Bohg, "Kite: Keypoint-conditioned policies for semantic manipulation," in *Proc. 7th Conf. Robot Learning (CoRL)*, 2023, pp. 1006–1021.

[12] A. Bucker, L. Figueredo, S. Haddadin, A. Kapoor, S. Ma, S. Vemprala, and R. Bonatti, "Latte: Language trajectory transformer," in *2023 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2023, pp. 7287–7294.

[13] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. P. Foster, P. R. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang, and C. Finn, "Openvla: An open-source vision-language-action model," in *Proc. The 8th Conf. Robot Learning (CoRL)*, ser. Proceedings of Machine Learning Research, vol. 270, 2025, pp. 2679–2713.

[14] W. Yuan, J. Duan, V. Blukis, W. Pumacay, R. Krishna, A. Murali, A. Mousavian, and D. Fox, "Robopoint: A vision-language model for spatial affordance prediction in robotics," in *Proc. of the 8th Conf. Robot Learning (CoRL)*, ser. Proceedings of Machine Learning Research, vol. 270, 2025, pp. 4005–4020.

[15] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 297–330, 2020.

[16] C. Celemin, R. Pérez-Dattari, E. Chisari, G. Franzese, L. de Souza Rosa, R. Prakash, Z. Ajanović, M. Ferraz, A. Valada, and J. Kober, "Interactive imitation learning in robotics: A survey," *Foundations and Trends in Robotics*, vol. 10, no. 1-2, pp. 1–197, 2022.

[17] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta *et al.*, "Open X-embodiment: Robotic learning datasets and RT-X models," in *2024 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2024, pp. 6892–6903.

[18] H. Liu, A. Chen, Y. Zhu, A. Swaminathan, A. Kolobov, and C.-A. Cheng, "Interactive robot learning from verbal correction," 2023.

[19] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox, "Correcting robot plans with natural language feedback," in *Robotics: Science and Systems (RSS)*, 2022.

[20] E. Merlo, M. Lagomarsino, and A. Ajoudani, "A human-in-the-loop approach to robot action replanning through LLM common-sense reasoning," *IEEE Robot. Autom. Lett. (RA-L)*, pp. 10 767–10 774, 2025.

[21] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia, "Language to rewards for robotic skill synthesis," in *Proc. 7th Conf. Robot Learning (CoRL)*, ser. Proceedings of Machine Learning Research, vol. 229, 2023, pp. 374–404.

[22] A. Maurya, T. Ghosh, A. Nguyen, and R. Prakash, "Ovita: Open-vocabulary interpretable trajectory adaptations," *IEEE Robot. Autom. Lett.*, vol. 10, no. 11, pp. 11 054–11 061, 2025.

[23] W. K. Kim, Y. Lee, J. Kim, and H. Woo, "LLM-based skill diffusion for zero-shot policy adaptation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[24] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, "Implicit 3d orientation learning for 6d object detection from rgb images," in *European Conf. on Computer Vision (ECCV)*, 2018.

[25] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations," *J. Comput. Phys.*, vol. 79, no. 1, pp. 12–49, 1988.

[26] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.