DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Enabling Task-Parameterized Imitation Learning in Unstructured Environments Using Visual Foundation Models

Valentin Gieraths

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Enabling Task-Parameterized Imitation Learning in Unstructured Environments Using Visual Foundation Models

Aufgabenparametrisiertes Imitationslernen in unstrukturierten Umgebungen mit visuellen Foundation Models

Author:	Valentin Gieraths
Examiner:	Prof. Dr.-Ing. Alin Albu-Schäffer
Supervisor:	Markus Knauer
External Supervisor:	Dr. João Silvério
Submission Date:	01.12.2025

I confirm that this master's thesis in robotics, cognition, intelligence is my own work and I have documented all sources and material used.

Munich, 01.12.2025 Valentin Gieraths

Acknowledgments

I would like to express my sincere gratitude to all those who contributed to the successful completion of this master thesis.

First and foremost, I am deeply grateful to my supervisor, Markus Knauer, for his exceptional support throughout this work. His continuous guidance, valuable hints, and constructive feedback greatly contributed to the development and refinement of the final results. His expertise and dedication were instrumental in shaping this research.

I would also like to extend my thanks to Samuel Bustamante for his substantial assistance in improving the tool calling and dynamic tool calling components of this work. His technical insights were invaluable to the implementation.

Special thanks are due to João Silvério for his regular feedback during our monthly status meetings. His thoughtful input helped ensure that the research stayed on track and met the required standards.

Furthermore, I am grateful to the German Aerospace Center (DLR) for providing the facilities and resources necessary for this thesis, particularly the opportunity to implement and validate the developed framework on a robotic platform. I would like to thank Freek Stulp, head of deparment Cognitive Robotics, for his support and for creating an excellent research environment.

Finally, I would like to thank my examiner, Professor Alin Albu-Schäffer, Head of the Institute and Professor at the Technical University of Munich, for making this collaboration possible.

Abstract

The direct deployment of robots in dynamic environments remains a challenging problem and is traditionally achieved by manual and often complex programming, especially in industry. Different approaches have been proposed to address this challenge, including training or fine-tuning Vision-Language-Action Models (VLAs), reinforcement learning policies, and combined hybrid methods. However, these typically require vast amounts of data and computational resources, and cannot be easily adapted to new tasks. Alternative approaches such as Imitation Learning (IL) and Task-Parameterized Imitation Learning (TPIL) can train new skills rapidly with limited data, but they struggle to adapt to dynamic environments without additional information. This thesis presents a framework integrating Task-Parameterized Kernelized Movement Primitives (TP-KMPs) with pretrained Vision-Language Models (VLMs) to enable natural language-based robot control with minimal demonstration requirements. The modular architecture separates perception, reasoning, and execution, leveraging classical robotics methods and modern foundation models. Skills are acquired through kinesthetic demonstration (3–6 examples per skill) and executed via natural language commands. The framework employs dynamic tool generation to enable seamless integration, eliminating foundation model fine-tuning. As a TPIL approach, TP-KMPs generate smooth trajectories conditioned on task parameters set automatically by the VLM, enabling execution across diverse environmental configurations. Key contributions include the implementation of the complete framework along with a perception pipeline for 6D object pose estimation. Additionally, a probabilistic skill combination mechanism leverages the covariance structure of TP-KMPs to synthesize novel behaviors from existing skills without additional demonstrations. A covariance manipulation strategy addresses compatibility constraints when individual Kernelized Movement Primitives (KMPs) exhibit insufficient variation in the demonstration data. Experimental validation on a torque-controlled, 7-DoF German Aerospace Center (DLR) Safe, Autonomous Robotic Assistant (SARA) robot demonstrates robust skill execution, generalization to novel configurations, and successful skill composition for industrial and household manipulation. The framework achieves data efficiency comparable to traditional IL approaches while providing intuitive natural language interaction. Results confirm that pretrained VLMs can serve as the reasoning layer when provided with appropriately structured interfaces, maintaining zero-shot capabilities while addressing spatial reasoning limitations.

Kurzfassung

Der direkte Einsatz von Robotern in dynamischen Umgebungen stellt nach wie vor eine Herausforderung dar und wird traditionell durch manuelle und oft komplexe Programmierung erreicht, insbesondere in der Industrie. Um diese Herausforderung zu bewältigen, wurden verschiedene Ansätze entwickelt, darunter das Training oder fine-tuning von VLAs, Reinforcement-Learning-Strategien und kombinierten Hybridmethoden. Diese erfordern jedoch in der Regel große Datenmengen und Rechenressourcen und lassen sich nicht ohne Weiteres an neue Aufgaben anpassen. Alternative Ansätze wie IL und TPIL können neue Fähigkeiten mit begrenzten Datenmengen schnell trainieren, haben jedoch Schwierigkeiten, sich ohne zusätzliche Informationen an dynamische Umgebungen anzupassen. Diese Arbeit stellt ein Framework vor, das TP-KMPs mit vortrainierten VLMs integriert, um eine auf natürlicher Sprache basierende Robotersteuerung mit minimalen Demonstrationsanforderungen zu ermöglichen. Die modulare Architektur trennt Wahrnehmung, Schlussfolgerung und Ausführung und nutzt dabei klassische Robotikmethoden und moderne Grundlagenmodelle. Fähigkeiten werden durch kinästhetische Demonstrationen (kinesthetic demonstration) (3–6 Beispiele pro Fähigkeit) erworben und über Befehle in natürlicher Sprache ausgeführt. Das Framework nutzt dynamische Werkzeuggenerierung, um eine nahtlose Integration zu ermöglichen und das fine-tuning des Grundmodells zu eliminieren. Als TPIL-Ansatz generieren TP-KMPs glatte Trajektorien, die von den automatisch vom VLM festgelegten Aufgabenparametern abhängen, und ermöglichen so die Ausführung in verschiedenen Umgebungskonfigurationen. Zu den wichtigsten Beiträgen gehören die Implementierung des vollständigen Frameworks zusammen mit einem Kamerasystem für die 6D-Objektposenschätzung. Zusätzlich nutzt ein probabilistischer Mechanismus zur Kombination von Fähigkeiten die Kovarianzstruktur von TP-KMPs, um aus bestehenden Fähigkeiten ohne zusätzliche Demonstrationen neue Verhaltensweisen zu synthetisieren. Eine Strategie zur Manipulation der Kovarianz berücksichtigt Kompatibilitätsbeschränkungen, wenn einzelne KMPs in den Demonstrationsdaten keine ausreichende Variation aufweisen. Die experimentelle Validierung an einem drehmomentgesteuerten 7-DoF DLR Roboter SARA zeigt eine robuste Ausführung der Fähigkeiten, eine Generalisierung auf neue Konfigurationen und eine erfolgreiche Zusammensetzung der Fähigkeiten für industrielle und häusliche Manipulationsaufgaben. Das Framework erreicht eine mit traditionellen IL-Ansätzen vergleichbare Dateneffizienz und bietet gleichzeitig eine intuitive Interaktion in natürlicher Sprache. Die Ergebnisse bestätigen, dass vortrainierte VLMs als Schicht für Denkprozesse dienen können, wenn sie mit entsprechend strukturierten Schnittstellen versehen sind, wobei die zero-shot-Fähigkeiten erhalten bleiben und gleichzeitig die Einschränkungen der räumlichen Argumentation berücksichtigt werden.

Contents

1. Introduction

The deployment of robots in dynamic, human-centric environments requires programming methods that are both intuitive and adaptive. In modern manufacturing, logistics, and service industries, robots increasingly operate alongside human workers in collaborative tasks [1, 2, 3]. These environments are characterized by frequent task variations, diverse product types, and the need for rapid reconfiguration to meet changing production demands. Traditional robot programming approaches, such as direct coding or Point-to-Point (PTP) programming, define robot behavior manually [4]. While effective for repetitive, high-volume tasks, these methods demand extensive technical expertise and explicit specification of motion primitives [5]. This creates a significant barrier in scenarios where non-expert operators must interact with robotic systems or where tasks change frequently [6]. For widespread adoption of collaborative robotics in industry, accessible and flexible programming paradigms have become essential, enabling workers without specialized programming knowledge to effectively deploy and adapt robotic capabilities.

Natural language interfaces offer a promising solution to this accessibility challenge. Consider a manufacturing scenario where a worker assembles components while a collaborative robot assists with material handling and positioning tasks. In such settings, human workers communicate naturally through speech and gesture, coordinating actions fluidly without interrupting their workflow. If the robot requires manual reprogramming or joystick control for each task variation, the collaborative nature of the work is fundamentally undermined, reducing the robot to a mere tool rather than an adaptive partner. Natural language commands enable workers to modify robot behavior dynamically, requesting adjustments such as "place the part closer to me" or "hand me the wrench" without interrupting ongoing assembly tasks. This interaction paradigm mirrors natural human collaboration and provides an intuitive interface that requires no specialized training, thereby unlocking the full potential of human-robot collaboration in industrial settings.

One of the key advantages of natural language interfaces is their support for open-ended and compositional task descriptions. This allows for generalization beyond narrowly defined action spaces, as these models can infer implicit task constraints, decompose multi-step instructions, and adapt behaviors to context-specific requirements [7, 8, 9]. Natural language also facilitates contextual grounding, enabling the interpretation of spatial relations, temporal sequences, object attributes, and qualitative constraints [10, 11]. A single natural language interface is capable of mediating interactions across a broad spectrum of applications, ranging from household assistance to industrial collaboration, without necessitating redesign of the underlying control policies [12, 13, 14]. In summary, the integration of natural language fundamentally expands the range and complexity of tasks that robots can perform, while simultaneously improving accessibility, adaptability, and generalization across real-world

scenarios.

However, enabling robots to understand and execute tasks specified in natural language presents significant technical challenges. The robot must interpret natural language commands, map them to appropriate actions, and adapt execution to the current environmental context. This requires sophisticated reasoning about language semantics, spatial relationships, and task constraints. Recent advances in foundation models, particularly the development of Large Language Models (LLMs) and VLMs, offer unprecedented capabilities for addressing these challenges. Foundation models are large-scale neural networks trained on vast amounts of internet-scale data, enabling them to learn rich representations of language, vision, and their interrelations [15].

Foundation models have revolutionized natural language understanding in robotics [12]. Models trained exclusively on text data, known as LLMs [16], demonstrate remarkable capabilities in language understanding, reasoning, and generation. VLMs [17] extend these capabilities by integrating visual perception with language understanding, enabling robots to interpret commands that reference objects and spatial relationships in their environment. These pretrained models exhibit strong zero-shot reasoning capabilities, allowing them to interpret complex task descriptions without task-specific training [12]. Building on these capabilities, VLAs [18] represent an end-to-end approach that learns direct mappings from visual and natural language inputs to robot actions. By training on large datasets of robot demonstrations, VLAs can generate control commands directly from natural language instructions and camera images, offering an integrated solution to the language-to-action problem.

Despite their impressive capabilities, pure VLA approaches face critical limitations for practical industrial deployment. Training or adapting a VLA to a specific robot platform and task domain requires hundreds of thousands of demonstrations and substantial computational resources [19], making them impractical for small-scale or specialized applications where collecting such extensive datasets is infeasible. Additionally, the end-to-end nature of VLAs provides limited interpretability and lacks explicit geometric reasoning, raising concerns for safety-critical industrial applications. Alternative approaches using foundation models for high-level task planning while delegating low-level control to specialized modules offer greater flexibility but often rely on discrete motion primitives or proprietary components [20], limiting their general applicability. In-Context Learning (ICL) techniques provide another direction, enabling task adaptation by including demonstrations directly in the model's system prompt [21]. However, this approach is constrained by limited context windows and cannot accumulate skills beyond what fits in the prompt, preventing long-term skill library development.

In parallel to advances in natural language understanding, imitation learning has emerged as a data-efficient paradigm for robot skill acquisition. Unlike reinforcement learning, which requires extensive trial-and-error exploration, imitation learning directly leverages expert demonstrations to learn effective behaviors [22]. Early approaches based on Gaussian Mixture Models (GMMs) [23] and Gaussian Processes (GPs) [24] demonstrated that smooth, probabilistic trajectory representations could be learned from a small number of demonstrations. Building on these foundations, KMPs [25] extended these methods using kernel techniques to

capture complex nonlinear patterns. Task-parameterized extensions, particularly TP-KMPs [26, 27], further enhanced generalization by encoding movements relative to task-relevant coordinate frames, enabling learned skills to adapt to different object configurations dynamically. These methods require only a handful of demonstrations per skill and provide inherent uncertainty quantification, making them well-suited for safe human-robot interaction [28]. Detailed coverage of these approaches is provided in chapter 2 and chapter 3. However, these imitation learning methods lack natural language interfaces, requiring users to manually specify which skill to execute and provide task parameters through technical interfaces.

Critically, no existing framework combines the intuitive natural language interfaces enabled by foundation models with the data-efficient, geometrically aware trajectory generation of TPIL approaches such as TP-KMPs. End-to-end VLAs provide natural language control but require impractical amounts of training data. Foundation model-based planning systems offer language understanding but often lack smooth, adaptive motion generation or rely on proprietary components. Imitation learning methods achieve data-efficient skill acquisition but lack intuitive natural language interfaces. A framework that bridges these paradigms, combining the language understanding of VLMs with the geometric reasoning and data efficiency of TP-KMPs, while supporting permanent skill acquisition and composition, remains an open challenge.

This thesis presents a framework that integrates TP-KMPs with pretrained VLMs to achieve natural language-based robot control with minimal demonstration requirements. The system employs a modular architecture where the VLM handles task interpretation and skill selection, while TP-KMPs generate execution trajectories conditioned on environmental task parameters. Skills are acquired through kinesthetic demonstration, requiring only 3–6 examples per skill, and are executed through natural language commands interpreted by the VLM. The framework supports dynamic skill composition through probabilistic fusion and runtime extension through automatic schema generation, enabling continuous capability expansion without system reconfiguration.

The key contributions of this work are as follows:

- Development of a modular architecture that combines pretrained VLMs with TP-KMPs for data-efficient skill learning, requiring only 3–6 kinesthetic demonstrations per skill while maintaining the flexibility of natural language interaction.
- Implementation of dynamic tool generation and automatic schema creation mechanisms that enable seamless integration between natural language interfaces and skill libraries, allowing the VLM to reason about skill semantics without fine-tuning.
- Introduction of a probabilistic skill combination mechanism that leverages the covariance structure of TP-KMPs to synthesize novel behaviors from existing skills without additional demonstrations, including strategies for handling compatibility constraints.
- Experimental validation demonstrating robust skill execution, generalization to novel object configurations, and successful skill composition in real-world manipulation tasks.

This thesis is structured as follows. The theoretical foundations are established in chapter 2 by reviewing imitation learning approaches, foundation models in robotics, and related work combining these paradigms. The mathematical background for KMP and TP-KMP is provided in chapter 3. The system architecture, including object perception, VLM integration, and the learning and execution pipelines, is presented in chapter 4. The experimental setup and results demonstrating skill acquisition, execution performance, and skill composition capabilities are described in chapter 5. The findings are analyzed and limitations are discussed in chapter 6. Finally, chapter 7 summarizes contributions and chapter 8 outlines directions for future research.

2. Related Work

2.1. Imitation Learning

Imitation Learning (IL), also referred to as Learning from Demonstration (LfD), represents a paradigm shift from traditional robot programming by enabling the transfer of new skills through demonstration rather than explicit coding [29]. In this approach, a human operator performs a task while the model observes or records the demonstration. Subsequently, the model infers the underlying behavior by generalizing from these examples, ultimately learning to perform the task autonomously [29, 30].

Unlike conventional programming, where each step of a task must be described in detail, IL allows the robot to capture the intent and dynamics of the task directly from examples [29, 30]. This makes it particularly attractive for complex or frequently changing environments, where formalizing rules and writing explicit instructions would be time-consuming, error-prone, or even infeasible. Some movements are significantly easier to demonstrate than to explicitly program [4]. The demonstrations can vary in nature and may involve kinesthetic teaching (physically guiding the robot), teleoperation (remote control of the robot), or video-based observation [29, 4]. One of the key advantages of IL is its accessibility to non-experts, as tasks can be demonstrated in a natural and intuitive manner without requiring programming experience or understanding of robot kinematics and dynamics [31, 32]. Furthermore, IL tends to be data-efficient, as it relies only on examples of successful task execution rather than the iterative trial-and-error required in reinforcement learning [32, 30], making it well-suited for real-world robotic systems where failed executions may result in equipment damage or safety hazards [32].

In contrast to IL, reinforcement learning approaches learn through trial-and-error interaction with the environment, where the agent receives rewards or penalties based on action outcomes. While reinforcement learning has achieved remarkable success in simulated environments, its application to real-world robotic manipulation faces significant challenges [33, 34]. The exploratory nature of reinforcement learning requires extensive interaction with the physical environment, which can lead to equipment damage, safety hazards, and substantial downtime. Although simulation-based training approaches have been developed to mitigate these risks by learning policies in virtual environments before deployment [35], their performance on real robots suffers from the sim-to-real gap, which remains an open challenge in the field [35]. Moreover, the sample complexity of reinforcement learning often necessitates a large number of interactions (frequently on the order of thousands of episodes or more) to converge to acceptable policies [33, 35, 34], whereas newer approaches leverage transfer learning and fine-tuning techniques with approximately 50-100 steps [12]. Nevertheless, IL remains faster to adapt to new situations, making it more practical for many industrial and service robotics

applications.

Despite these advantages, early IL approaches face limitations in generalization and adaptability. The development of probabilistic methods progressively addresses these challenges, leading to increasingly capable imitation learning techniques.

2.1.1. Gaussian Mixture Models (GMMs) and Regression (GMR)

A Gaussian Mixture Model (GMM) models a probability distribution as a weighted sum of Gaussian components. In the context of IL, GMMs are widely used to capture the variability and correlation structure of demonstrated trajectories [23, 28]. Given such a joint distribution (e.g., over time and robot states), Gaussian Mixture Regression (GMR) is employed at reproduction time to condition on a query variable (such as time or phase) and retrieve a smooth reference trajectory as the conditional mean together with an associated covariance [28].

While GMM/GMR approaches have been successful in various robotic applications, they exhibit limitations that restrict their practical applicability. One key drawback is their parametric nature, which requires the number of Gaussian components to be specified a priori. An inappropriate choice can lead to underfitting or overfitting if the selected number of components does not align well with the underlying data distribution [28]. Moreover, in high-dimensional state-action spaces, the number of parameters per component grows quickly, so practical implementations often rely on simplified covariance structures or feature engineering, which can limit expressiveness.

Furthermore, standard GMM/GMR formulations are typically estimated via maximum likelihood and thus primarily capture aleatoric uncertainty (the variability present in the demonstrations) through the component covariances, while ignoring epistemic uncertainty about the model parameters themselves. As a result, they may become overconfident in regions with sparse data, which is problematic for safe robot operation in dynamic environments [36]. Lastly, vanilla GMM/GMR encodes trajectories in absolute coordinates and is not inherently task-parameterized, making it difficult to generalize learned skills to new goals or frames without additional structure, such as task-parameterized or parametric GMM extensions [26, 28].

2.1.2. Gaussian Processes (GPs)

A Gaussian Process (GP) is a non-parametric Bayesian model that defines a prior distribution over functions [24]. Similar to GMMs, GPs can be employed in IL to model the relationship between input states (or contextual variables) and output actions based on demonstrated trajectories [28, 37]. Using, for example, Gaussian Process Regression (GPR), one obtains for each query state a Gaussian predictive distribution whose mean provides a smooth reference action or trajectory point and whose variance reflects the model's predictive uncertainty [28].

Compared to GMM/GMR, GP-based approaches do not require specifying a fixed number of components in advance and naturally provide estimates of epistemic uncertainty that typically grow away from the training data. However, they also exhibit limitations that

restrict their practical applicability in IL. A prominent drawback is their computational complexity. Standard GPR has cubic training time in the number of demonstrations and quadratic memory requirements, which makes it less suitable for large datasets or strict real-time constraints without additional approximations or model structure. Additionally, achieving good performance on complex, high-dimensional tasks often requires careful kernel design and task-specific feature representations, a challenge that has motivated deep imitation learning methods that learn features and policies jointly [38]. Similar to GMM/GMR, a plain GP prior is not inherently task-parameterized. Generalizing learned skills to new contexts typically requires augmenting the inputs with task parameters or adopting explicit task-parameterized GP formulations [28, 25].

2.1.3. Kernelized Movement Primitives (KMPs)

The Kernelized Movement Primitive (KMP) is a kernel-based, non-parametric approach for encoding and reproducing demonstrated trajectories. It leverages kernel methods to model the relationship between input variables (e.g., time or joint configurations) and output variables (e.g., end-effector poses or speed), allowing for flexible and adaptive trajectory generation [25]. Starting from a probabilistic reference trajectory extracted from demonstrations, KMPs derive a movement primitive by minimizing the information loss (*Kullback-Leibler* Divergence (KL-Divergence)) between the primitive and the reference distribution. During reproduction, the KMP yields, for each query input, a Gaussian distribution over the output, whose mean defines a smooth reference trajectory and whose covariance reflects the variability in the demonstrations and the imposed constraints [25]. A detailed explanation of how KMPs work is provided in section 3.1.

Compared to earlier movement primitive formulations, the KMP offers several advantages in the context of IL. First, due to its kernel treatment, it avoids the explicit design of basis functions and can naturally handle trajectories driven by high-dimensional inputs while keeping the number of open parameters small. Second, the KMP supports modulation of trajectories via additional probabilistic constraints (e.g., via-points and end-points). However, KMP also inherits typical limitations of kernel methods. Its computational cost grows with the size of the reference database, and its performance depends on the choice of kernel function and associated hyperparameters. Moreover, it shares the same limitations as other IL methods regarding generalization to varying task contexts, which has been addressed by the Task-Parameterized Kernelized Movement Primitive (TP-KMP) [27] as described in the next section.

A summary comparison of the discussed probabilistic IL methods is provided in Table 2.1.

2.1.4. Task-Parametrized Imitation Learning (TPIL)

The IL approaches mentioned above share a common limitation in practical applicability. These methods are task-specific, replicating the exact sequence demonstrated and failing to generalize to similar but modified tasks. This means that typically only a single skill or operation can be demonstrated at a time [38]. If the position of objects changes or if the

Table 2.1.: Qualitative comparison of probabilistic imitation learning methods.

Property	GMM+GMR	GP+GPR	KMP
Non-parametric	✗	✓	✓
Aleatoric uncertainty	✓	✓	✓
Epistemic uncertainty	✗	✓	✓
Scalability to High-Dimensional Spaces	✗	✗	✓
Adaptive Skill Modulation (e.g., via-points)	✗	✗	✓

task itself is modified, the operator often needs to re-demonstrate the entire operation [31]. Although the number of required demonstrations is usually in the single-digit range, this process can still be inconvenient, time-consuming, and impractical in dynamic or real-world settings. While approaches exist to adapt learned trajectories based on user interaction [39, 40], they require additional effort from the user during task execution.

Task-Parameterized Imitation Learning (TPIL) is a variant of IL developed to address the challenge of varying object positions in the environment [25, 26, 28, 41, 42]. Key examples include Task-Parameterized Gaussian Mixture Model (TP-GMM) [23, 28] and variations of it [42], as well as TP-KMP [25, 27]. In these approaches, movements are encoded relative to the coordinate frames of relevant objects, allowing the model to explicitly account for changes in task geometry. By conditioning learned behaviors on task parameters that capture key environmental features, these methods enable robots to adapt flexibly to novel task configurations.

To support generalization across diverse object arrangements, task-parameterized models typically combine movement representations from multiple reference frames using a product of probability distributions. This probabilistic fusion allows the generation of trajectories that remain consistent even when the spatial layout of the task environment varies. Consequently, skills acquired through TPIL can generalize to unseen object configurations, enabling robust and adaptive execution through real-time updating of task parameters [28, 41]. Therefore, it is not necessary to reprogram or relearn a skill solely because object positions have changed, which is particularly advantageous in shared human-robot environments. While standard IL may produce unpredictable or unsafe motions when faced with scenarios outside the training data, TPIL demonstrates more predictable and reliable robot behavior in novel contexts [43].

Despite these advantages, TPIL faces several challenges. The complexity and computational cost of TPIL models, such as TP-GMMs, increase rapidly with the number of task parameters and the dimensionality of the data. This makes TPIL challenging to scale to high-dimensional tasks or long-horizon trajectories, which are common in realistic robotic applications [44, 27]. However, models such as TP-KMP are able to handle high-dimensional input and output domains more effectively [25, 27].

The selection and definition of task parameters (e.g., reference frames) are often performed manually or require additional components such as object tracking systems, typically depending on accurate perception of task-relevant objects. This reliance on external sensing components can restrict applicability and robustness in unstructured environments [44].

Furthermore, while TPIL is effective for generalizing over spatial transformations in position and orientation, it cannot handle more abstract variations in task logic or high-level reasoning.

2.2. Foundation Models in Robotics

Foundation models are pretrained on internet-scale datasets and exhibit enhanced generalization capabilities. Notably, these models have demonstrated emergent properties, including the ability to perform zero-shot inference on tasks not explicitly represented in their training data [15, 12, 13].

Foundation models in robotics can be categorized into three primary classes: LLMs, VLMs, and VLAs. Each class builds upon the capabilities of its predecessor, progressively integrating additional modalities to enhance robotic control and interaction.

2.2.1. Large Language Models (LLMs)

Foundation models trained on internet-scale text data are called LLMs and are designed to understand and generate human language with remarkable fluency [45, 15]. These models can autonomously generate executable code or perform commonsense reasoning [13, 12].

In robotics, LLMs are primarily employed for high-level task planning, reasoning, and code generation [12, 13, 46, 47]. Their capacity for semantic understanding and logical inference allows them to decompose complex task descriptions into executable action sequences. Utilizing the reasoning capabilities of pretrained LLMs is more effective than training small models from scratch [47]. Language-conditioned approaches have been explored in various studies [8, 7, 48, 49, 9, 50].

However, LLMs lack the ability to directly process visual information or generate robot control actions. Without access to sensory input, they cannot perform spatial reasoning or ground language commands in the physical environment, limiting their utility to abstract reasoning tasks that must be paired with separate perception and execution modules [12, 13, 9].

2.2.2. Vision-Language Models (VLMs)

Building upon the language understanding capabilities of LLMs, VLMs extend these capabilities by integrating visual perception with language understanding [17]. These models can facilitate open-vocabulary visual recognition and overcome the sensory limitations of LLMs by providing visual understanding alongside natural language reasoning.

In robotics, VLMs are employed for tasks requiring visual grounding [12, 13, 46, 10, 51, 52]. Perception modules frequently employ pretrained vision models, possibly in conjunction with a user-provided prompt [53, 54, 55, 56, 20]. These models can interpret spatial relations, temporal sequences, object attributes, and qualitative constraints [11]. This capacity for contextual grounding is particularly useful in unstructured environments where symbolic representations or fixed taxonomies are insufficient.

Despite these advantages, current VLMs are generally not well suited for generating precise robot trajectories and are mainly used for perception or high-level guidance, rather than low-level control [12, 13, 46]. Utilizing pretrained VLMs to directly predict robot motion requires providing extensive contextual information, which contradicts the purpose of intuitiveness and automation. Consequently, they require integration with separate execution modules to translate high-level understanding into physical actions.

2.2.3. Vision-Language-Action Models (VLAs)

The most recent evolution in foundation models for robotics are VLAs, which extend VLMs to directly act on the robot. These models are tailored specifically to robotics applications and integrate perception, reasoning, and action generation within a unified architecture.

To directly produce motor commands or motion trajectories, VLAs receive a combination of visual input and textual prompt. Models such as OpenVLA [19] exemplify this approach. These models operate in an end-to-end fashion, mapping directly from sensory input to control outputs without requiring intermediate processing stages.

Characterized by their broad applicability and capacity for generalization to previously unseen tasks, VLAs often achieve zero-shot performance. In some instances, these models are robot-agnostic, enabling deployment across varying robotic platforms by simply transferring the predicted trajectory or action parameters [57]. The unified architecture eliminates the need for multiple specialized modules and their associated integration complexity.

However, VLAs require substantial amounts of task-specific data for training or fine-tuning, yet robot-relevant training data remains scarce and expensive to acquire [13, 53, 58]. Moreover, most pretrained VLAs need to be fine-tuned on the specific robot they should act on, making it impossible to find suitable training data online depending on the robot embodiment [12, 13, 58]. Many existing models also operate at low control frequencies [46], rendering them unsuitable for tasks demanding fine-grained force modulation or responsive human-robot collaboration. Given the varying capabilities and limitations of LLMs, VLMs, and VLAs, researchers have developed two primary architectural paradigms for deploying these models in robotic systems.

2.2.4. End-to-End and Layered Architectures

In the context of language-conditioned robotic control, a wide range of studies have explored the application of foundation models [12, 13, 46]. These approaches typically leverage foundation models either in an end-to-end fashion or within a modular, layered architecture.

End-to-end paradigms utilize a single foundation model, commonly a VLA, that receives a combination of visual input and textual prompt to directly produce motor commands or motion trajectories, as discussed above. Alternatively, layered approaches decompose the robotic control pipeline into discrete functional modules such as perception, reasoning, and execution. Each module can be implemented using distinct foundation models, which are often smaller and may not require additional fine-tuning [19]. Execution may be handled by traditional motion primitives rather than foundation models [59], depending on task

complexity. These architectures often rely on inter-model communication and verification, whereby the output of one model is assessed or refined by another to ensure feasibility and consistency in achieving the robotic task objective [14].

Both approaches, end-to-end and layered, are inherently capable of processing natural language, which significantly enhances their versatility and applicability across diverse human-centric domains. Natural language serves as a high-level interface between humans and robots, enabling task specification in an intuitive and accessible manner without the need for specialized programming skills, as discussed in chapter 1.

Despite recent advancements with foundation models in robotics, several limitations persist that hinder their deployment in real-world applications. Current approaches frequently lack robust mechanisms for uncertainty quantification and safety assurance [60], both of which are essential for reliable operation in dynamic or human-centric environments. Real-time adaptability also remains a challenge across all model types.

2.3. Research Gap and Proposed Solution

Recent advances in foundation models for robotics have produced a diverse set of frameworks that address natural language comprehension, environmental adaptability, and skill learning. Approaches such as SayCan [61], Retrieval-Augmented Planning (RAP) [62], and Closed-Loop Embodied Agents (CLEA) [63] demonstrate effective integration of LLMs and VLMs with robotic systems for task planning and execution. However, a gap remains when considering the combination of data efficiency, accessibility, and practical deployability. Existing frameworks typically require either (i) extensive demonstration data and computational resources for training (e.g., SayCan with 68,000+ demonstrations, OpenVLA with 970k trajectories), (ii) proprietary foundation models that limit reproducibility and increase operational costs, or (iii) in-context learning approaches that are constrained by prompt length and cannot acquire genuinely new skills at runtime. Furthermore, few existing approaches support the flexible combination of learned skills into novel task sequences while maintaining smooth, predictable trajectory generation with inherent uncertainty quantification.

The proposed integration of TP-KMP with pretrained VLMs addresses these limitations through a layered architecture that separates high-level reasoning from low-level control. The model's responsibilities are intentionally constrained to high-level reasoning tasks, which has been demonstrated to enhance overall system performance. The use of TP-KMP as the foundational execution component eliminates the need for extensive demonstration data or model fine-tuning, as off-the-shelf general-purpose VLMs can be directly employed. The mathematical and probabilistic foundations of TP-KMP enable the generation of smooth and predictable trajectories with inherent uncertainty quantification. Furthermore, new skills can be acquired rapidly through the training of additional TP-KMP instances from only a handful of demonstrations, circumventing both the extensive data requirements of end-to-end approaches and the prompt length constraints of in-context learning methods. The framework's flexibility in controller selection, including impedance control systems, contributes to enhanced safety in human-robot interaction scenarios.

Compared to traditional LfD approaches, the proposed framework offers notable improvements in usability and scalability. The integration of a VLM eliminates the manual specification of task parameters, enabling operators to keep both hands free while commanding the robot through natural language instructions. This interface remains intuitive and accessible for workers without specialized training, while avoiding dependence on proprietary foundation models. Moreover, the integration of TP-KMPs with a VLM makes the teaching process more autonomous and scalable by eliminating the need to manually define task parameters for each demonstration.

2.4. Comparison with Existing Works

This section reviews seven recent works that are conceptually related to the approach presented in this thesis. Each work leverages LLMs or VLMs to interface with robotic systems but employs different architectural paradigms and learning strategies. The works are first described independently, followed by a comparative analysis that highlights the key distinctions with respect to the framework developed in this thesis.

2.4.1. OpenVLA

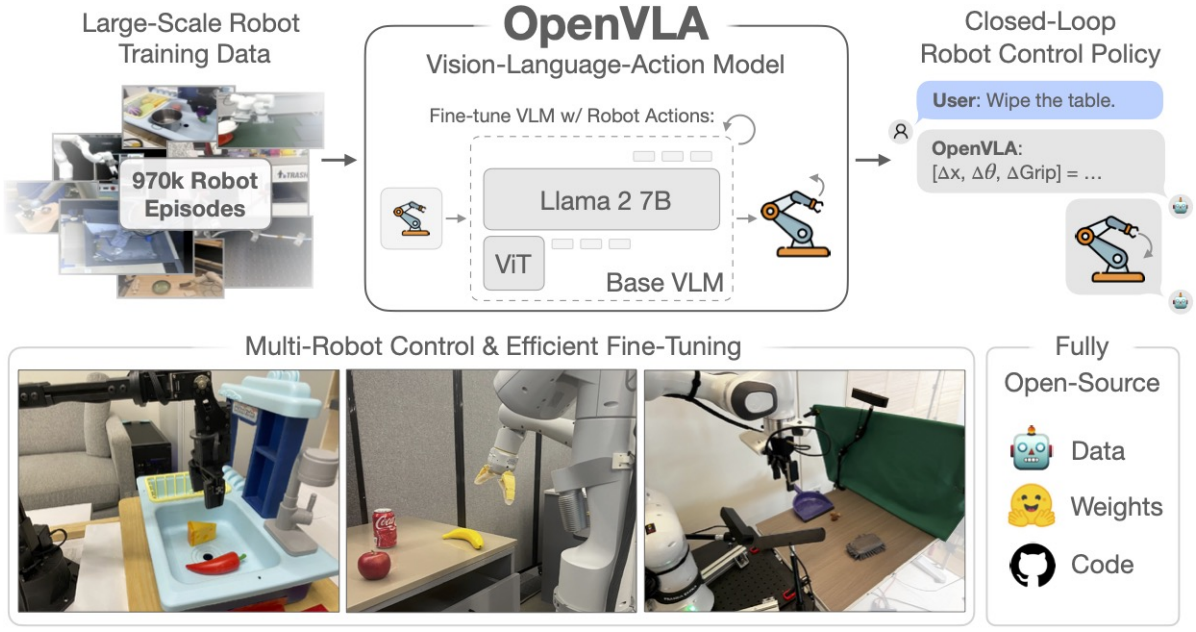

Figure 2.1.: Overview of the OpenVLA architecture [19]. The model integrates a vision-language backbone with a tokenization scheme for robot actions, enabling end-to-end prediction of control commands from visual and language inputs. Image credit: Kim et al. [19].

OpenVLA [19] is a 7B-parameter vision-language-action model designed to generate robot

control actions directly from visual observations and natural language instructions. The model is built upon a pretrained vision-language model (`Prismatic VLM` [64]) that incorporates a fused `SigLIP` [55] and `DINOv2` [54] vision encoder along with a `Llama 2` [45] language backbone. To enable action prediction, continuous robot actions are discretized into 256 bins per dimension and mapped to tokens in the language model's vocabulary.

Training is performed on 970k robot demonstrations from the Open X-Embodiment dataset using a next-token prediction objective, where action tokens are predicted conditioned on visual and language inputs. The approach demonstrates generalization across multiple robot embodiments and can be adapted to new tasks through fine-tuning on 10–150 demonstrations, either via full parameter updates or parameter-efficient methods such as LoRA [65]. At inference time, the model operates in a closed-loop fashion, continuously predicting single-step actions at approximately 6 Hz on consumer-grade Graphical Processing Units (GPUs).

The modular design of the framework presented in this thesis separates high-level reasoning from low-level control, using a pretrained VLM for task interpretation without fine-tuning, while TP-KMPs handle trajectory generation. It provides explicit uncertainty quantification, and enables geometric generalization through task parameterization with a significantly smaller datasets than OpenVLA, which learns implicitly from large datasets.

2.4.2. SayCan: Grounding Language in Robotic Affordances

Figure 2.2.: Overview of the SayCan framework [61]. The system combines LLM probabilities for task usefulness with value function probabilities for execution feasibility to select skills that are both semantically appropriate and physically achievable. Image credit: Ahn et al. [61].

SayCan [61] addresses the challenge of grounding LLMs in real-world robotic capabilities by combining high-level semantic knowledge from LLMs with learned affordance functions that represent what is physically feasible in the current state. The key insight is that while LLMs can decompose complex natural language instructions into subtask sequences, they lack awareness of what a specific robot can actually accomplish in its current environment. SayCan factorizes the problem into two components: the LLM provides task-grounding, representing the probability that a skill's language description makes progress toward completing the

instruction, while value functions provide world-grounding, representing the probability that the skill will succeed from the current state.

The system operates by scoring potential next skills, selecting those that maximize the product of semantic relevance and physical feasibility. Skills are represented through language descriptions and come with associated policies and value functions trained either via behavioral cloning (BC-Z [66]) or reinforcement learning (MT-Opt [67]). The value functions are trained using temporal-difference methods with sparse binary rewards (1.0 for successful completion, 0.0 otherwise) in simulation. The LLM component uses prompt engineering with dialog-structured examples (similar to ICL) to constrain outputs to the robot's skill repertoire, with the `PaLM-540B` [68] model serving as the primary language model.

SayCan was evaluated on 101 real-world tasks across seven instruction families using a mobile manipulator in office kitchen environments, with skills including object manipulation (pick, place) and navigation (go to locations). The framework demonstrated zero-shot execution of temporally extended instructions and showed that robot performance scales with LLM capability, observed by improved performance through upgrading from `FLAN-137B` [69] to `PaLM-540B` [68]. The approach required 68,000 teleoperated demonstrations collected over 11 months plus 12,000 filtered autonomous episodes for training the underlying policies and value functions, with skills extensible through adding new prompt examples and affordance functions.

Compared to SayCan, the framework presented in this thesis achieves dramatically lower data requirements, avoids dependence on proprietary models such as `PaLM`, and provides inherent uncertainty quantification through the probabilistic TP-KMP formulation. However, SayCan's affordance-based feasibility scoring offers implicit closed-loop adaptation that the current open-loop execution in this thesis does not provide.

2.4.3. RAP: Retrieval-Augmented Planning

RAP [62] is a framework designed to enhance the planning capabilities of LLMs by leveraging past successful experiences stored in memory. The approach draws inspiration from the human ability to recall and apply relevant past experiences when making decisions in new situations. RAP consists of four core components: Memory (storing successful task execution logs), Reasoner (generating plans and retrieval keys using LLMs), Retriever (finding relevant past experiences based on similarity), and Executor (generating actions via ICL from retrieved experiences).

Memory is constructed by collecting logs from successful task executions, including task information, overall plans, and trajectories (action-observation sequences). For textual environments, observations are text descriptions, while for multimodal environments, observations include visual representations from a fixed camera. Retrieval is performed by calculating weighted similarity scores between the current state and stored logs. The retriever adaptively switches between action-based and observation-based similarity depending on the context, using sentence transformers [16] for text and CLIP-based [17] models for images.

The framework demonstrates effectiveness across both text-only environments (ALFWorld [70], WebShop [71]) and multimodal embodied tasks (Franka Kitchen [72], Meta-

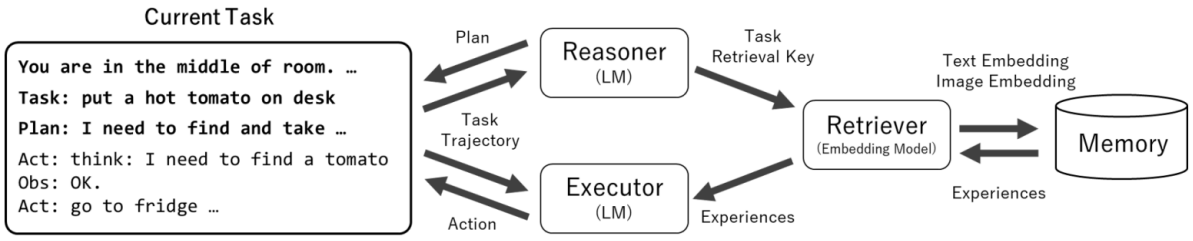

Figure 2.3.: Overview of the RAP framework [62]. The system stores past successful trajectories and retrieves relevant experiences based on the current situation to guide action planning. Image credit: Kagaya et al. [62].

World [73]). On ALFWorld with `GPT-3.5` [74], RAP uses memory built from 1000 training trajectories. For multimodal tasks, RAP enhances `LLaVA` [75] agents on Franka Kitchen and Meta-World, requiring 25 demonstrations per task to train low-level policy networks (Multi-Layer Perceptrons (MLPs)) that translate high-level plans into executable actions. How these executable actions are generated and stored is not detailed in the paper.

Compared to RAP, the approach presented in this thesis requires fewer demonstrations per skill, while eliminating the need for simulation-based learning. However, RAP's memory-based retrieval mechanism for leveraging past experiences represents a promising direction for future extensions to the proposed framework.

2.4.4. CLEA: Closed-Loop Embodied Agent

CLEA [63] is a closed-loop planning framework that enables adaptive robotic task execution in dynamic environments through continuous environmental feedback. Unlike open-loop approaches that generate static action sequences, CLEA employs four functionally decoupled modules: an observer (VLM) that converts visual inputs to text, a summarizer (LLM) that maintains belief states from interaction history, a planner (LLM) that generates hierarchical sub-goals and action sequences, and a critic (VLM) that performs real-time feasibility assessments and triggers replanning when needed.

The framework models tasks as Partially Observable Markov Decision Processes (POMDPs), maintaining a belief state over environmental states. The memory module stores observation-action-feedback tuples in a First In, First Out (FIFO) history buffer, which the summarizer distills into structured representations for planning. The planner uses chain of thought prompting to generate action sequences from a predefined skill pool, while the critic evaluates each action before execution, detecting issues such as outdated assumptions, redundant operations, or infeasible plans.

CLEA was evaluated using two heterogeneous robots (a dual-arm mobile manipulator and a fixed-base manipulator) on object search, manipulation, and integrated tasks in a real kitchen environment. `Qwen2.5-72B` models [76] were used for both LLM and VLM components.

While the framework presented in this thesis currently executes skills in an open-loop

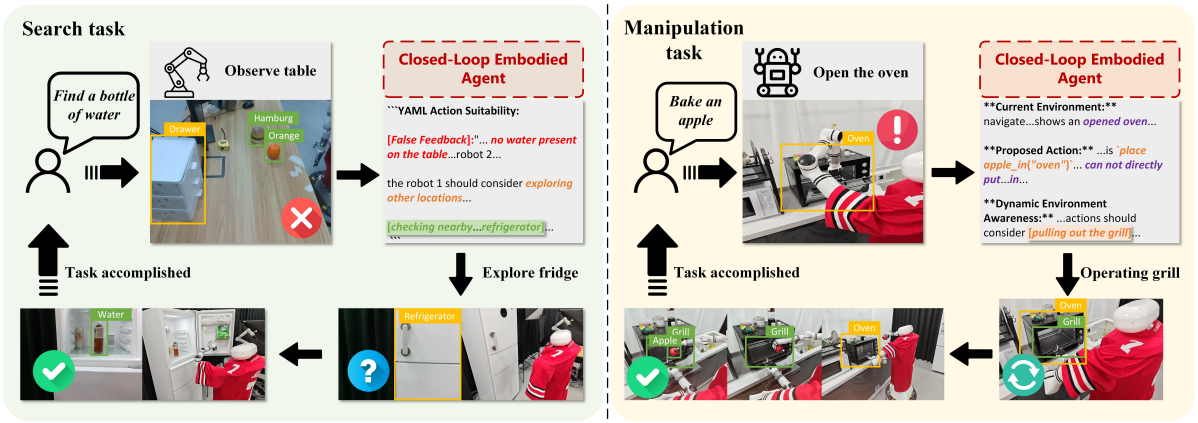

Figure 2.4.: Overview of the CLEA framework [63]. The system integrates observer, memory, planner, and critic modules to enable adaptive decision-making through continuous perception-reasoning-execution cycles. Image credit: Lei et al. [63].

fashion, CLEA's modular critic architecture demonstrates a promising direction for future extensions. Integrating a similar feedback mechanism could enable the proposed system to detect execution failures and trigger replanning, combining the advantages of data efficiency, skill learning, and uncertainty quantification with closed-loop adaptability.

2.4.5. RoboPrompt: In-Context Learning Enables Robot Action Prediction in LLMs

The method presented by Yin et al. [21] combines LfD with LLM-based action prediction through in-context learning. Demonstrations are segmented into keyframes by identifying time points where the robot's velocity approaches zero. At each keyframe, both the robot's and the object's 6D poses are recorded and subsequently translated into natural language using array-like structures. Object poses are extracted using a visual backbone such as `Grounding DINO` [77], while robot actions are described as simple movements between stored positions.

The input-output pairs derived from these demonstrations are formatted into an ICL prompt including 10 examples and passed to an off-the-shelf, pretrained LLM. At inference time, the model generates suitable action descriptions given a new initial configuration. The approach demonstrates the pattern recognition and basic reasoning capabilities of LLMs in the robotics domain, though it remains restricted to in-context learning and exhibits limited generalization across domains.

The framework presented in this thesis, on the other hand, stores learned skills persistently, enabling unlimited skill library growth, with even fewer demonstrations or through combinations of skills.

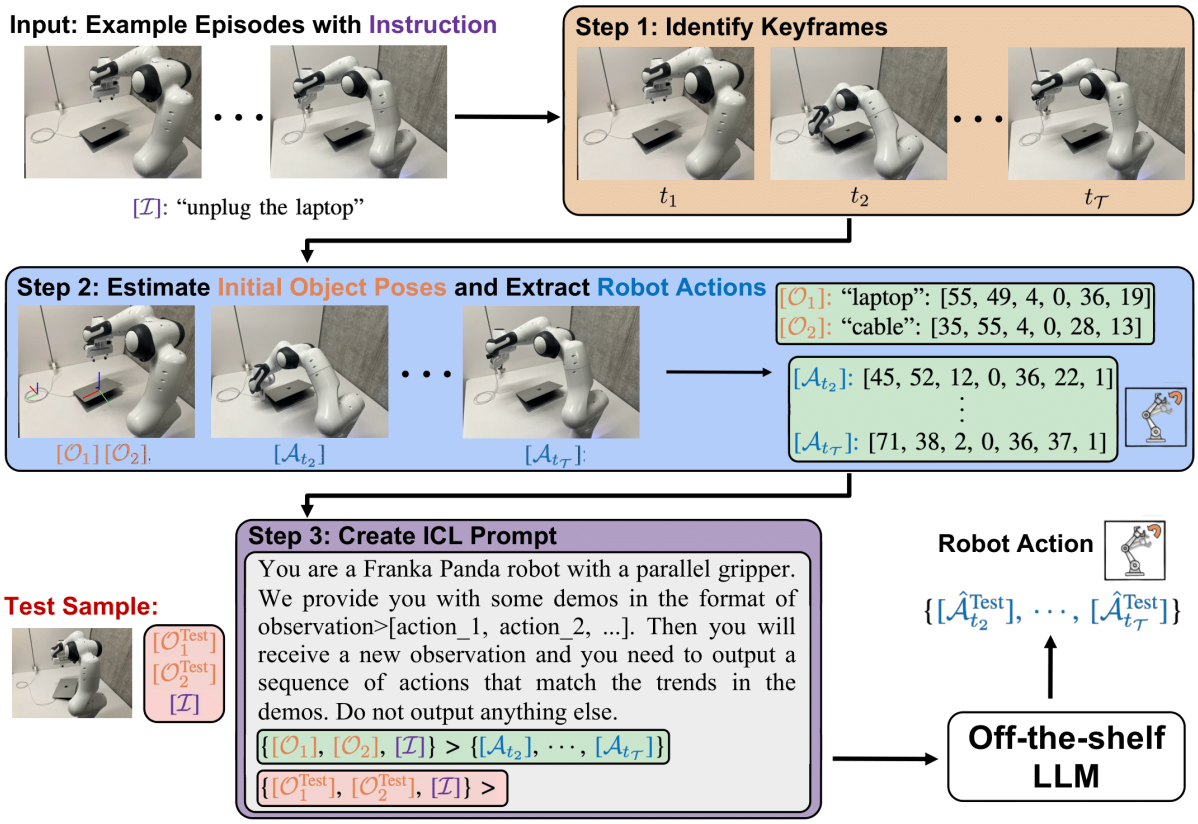

Figure 2.5.: Overview of the framework by [21]. Demonstrations are segmented into keyframes, and input-output pairs are formatted into an in-context learning prompt for a pretrained large language model. Image credit: Yin et al. [21].

2.4.6. VocalSandbox: Continual Learning and Adaptation

The framework proposed by Grannen et al. [78] employs a LLM to map natural language instructions to a structured Application Programming Interface (API) of executable robot functions. Functionality is organized into a hierarchy of low- and high-level actions. Low-level actions, such as `goto(..)` and `grasp()`, are implemented as pre-programmed policies. In the described implementation, these policies are realized using discrete Dynamic Movement Primitives (DMPs) constructed by extracting waypoints from kinesthetic demonstrations and interpolating between them. This design enables motion generalization to novel start and goal configurations. Object poses are inferred using a perception system that employs a SAM [79] model and an RGB-D camera.

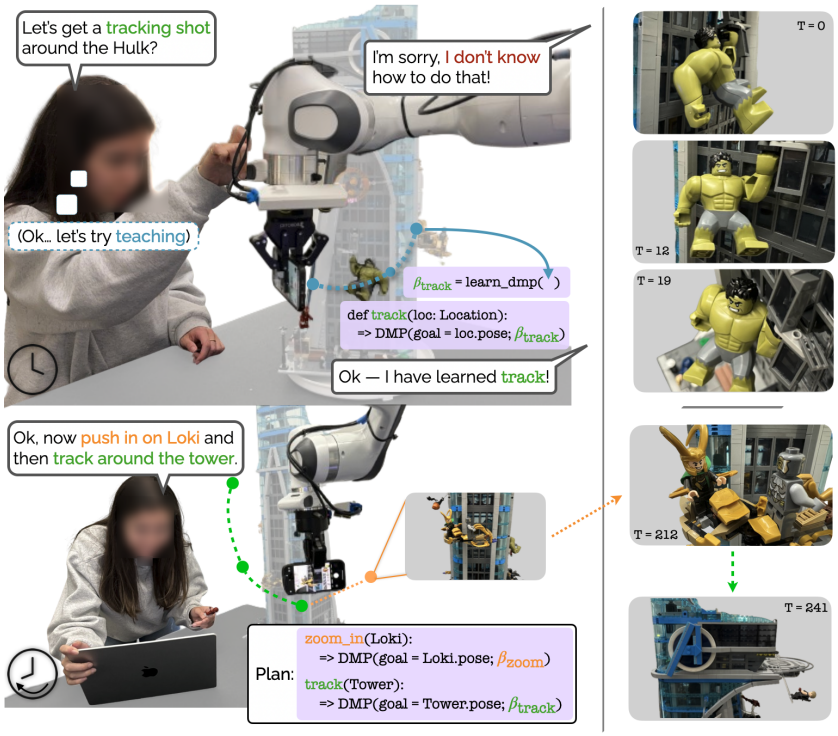

Figure 2.6.: Overview of the VocalSandbox framework [78]. A large language model maps natural language instructions to a structured API of robot functions, enabling learning and adaptation through user interaction. Image credit: Grannen et al. [78].

High-level functions are composed of sequences of low-level actions. As a case in point, `pickup(..)` is implemented as a `goto(..)` call followed by `grasp()`. The API accepts arguments specifying object identities or spatial locations, such as `HOME = [1, 2, ..]` and `TOY_CAR = "A green toy car"`. Both high-level functions and arguments can be dynamically extended through interaction with the LLM. `GPT-3.5 Turbo` [74] is employed via the function-calling API, with each function exposed as a callable tool. The system relies on ICL with a

proprietary model, and the prompting strategy is fixed and context-dependent, with system prompts predefining high-level task objectives.

The framework presented in this thesis employs TP-KMPs instead of DMPs, providing probabilistic uncertainty quantification. It uses open-source VLMs rather than proprietary models and allows for persistent skill acquisition beyond the a given dataset of low-level actions.

2.4.7. A Robotic Skill Learning System Built Upon Diffusion Policies and Foundation Models

Ingelhag et al. present a Robotic Skill Learning System (RSLS) that combines visuomotor diffusion policies [80] with pretrained LLMs and VLMs for skill selection and precondition checking [81]. The system maintains a library of skills, each represented by a learned diffusion policy together with a textual description and a set of preconditions. Given a user instruction and an image of the current scene, an LLM selects a suitable skill from the library. If a candidate skill is found, a VLM verifies whether its preconditions are satisfied in the current workspace before execution. If no existing skill matches the request, the system prompts the user to provide demonstrations for a new skill.

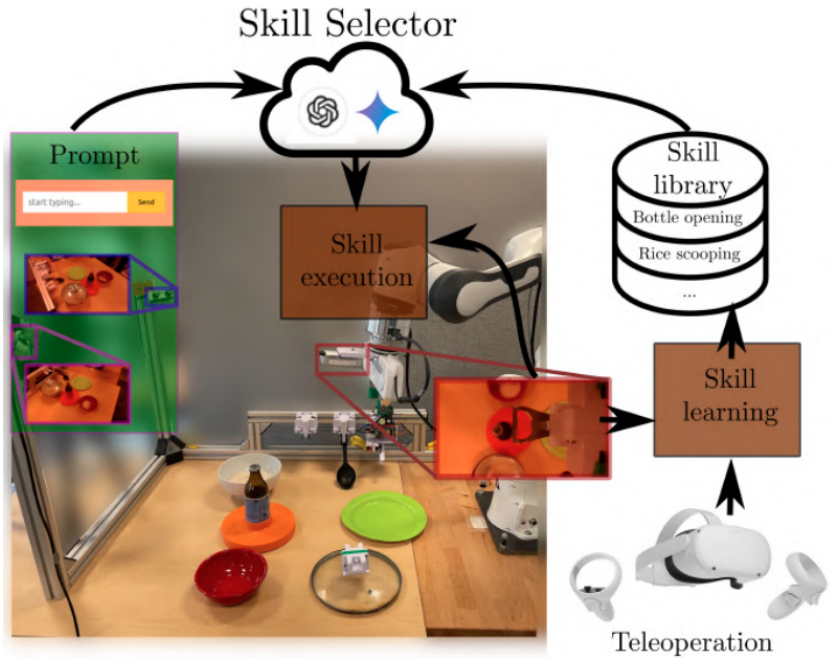

Figure 2.7.: Overview of the Robotics Skill Learning System (RSLS) [81]. A large language model selects skills from a library based on user instructions, while a vision-language model verifies skill preconditions before execution. Image credit: Ingelhag et al. [81].

New skills are acquired from teleoperated demonstrations using an Oculus Quest controller

in both simulation and the real world. The collected trajectories (typically 50–150 demonstrations per skill) are used to train visuomotor diffusion policies that map image observations and end-effector states to continuous actions. New skills are trained for 600 epochs taking approximately 10 hours on consumer-grade GPUs.

The authors evaluate the framework both in Isaac Sim [82] and on a real Franka Panda robot on a set of food-serving tasks, including bottle opening, lid removal, rice scooping, and sausage placing. They compare different foundation models (`GPT-4` [83] and `Gemini` [84]) for the skill selection and precondition-checking components [81].

While Robotic Skill Learning System (RSLS) and the framework presented in this thesis both offer active skill acquisition through user-initiated demonstrations, several key differences exist. The framework presented here requires fewer demonstrations, supports skill combination through probabilistic fusion, provides uncertainty quantification, and uses open-source models exclusively.

2.4.8. Comparative Analysis

The approach developed in this thesis differs from the reviewed works along several dimensions concerning architecture, learning paradigm, skill representation, and runtime extensibility. An overview of the key differences is provided in Table 2.2.

Table 2.2.: Qualitative comparison of the presented framework with related methods. A checkmark (✓) indicates the property is supported, a cross (✗) indicates it is not supported, a checkmark in parentheses indicates partial support, and a dash (–) indicates the property is not applicable (e.g., CLEA uses a predefined skill pool without per-skill training). Data efficiency is considered achieved if new skills can be learned from 10 or fewer demonstrations.

Property	This work	OpenVLA	SayCan	RAP	CLEA	RoboPrompt	VocalSandbox	RSLS
Natural Language Interface	✓	✓	✓	✓	✓	✓	✓	✓
No Largescale Training	✓	✗	✗	✗	✓	✓	✓	✗
Model Accessibility	✓	✓	✗	✓	✓	✓	✗	✗
Data Efficiency	✓	✗	✗	✗	–	✓	–	✗
Runtime Skill Acquisition	✓	✗	✗	✗	✗	✗	(✓)	✓
Persistent Skill Representation	✓	✓	✓	✓	(✓)	✗	(✓)	✓
Combination of Skills	✓	✗	✗	✗	✗	✗	(✓)	✗
Closed-Loop Adaptation	✗	✓	(✓)	(✓)	✓	✗	✗	✗

End-to-end versus modular architectures. All approaches provide a natural language interface, but differ in their architecture. OpenVLA employs a single end-to-end VLA that directly predicts actions from visual and natural language inputs, trading strong generality for substantial data and compute requirements. In contrast, VocalSandbox, RSLS, SayCan, CLEA, and RAP adopt modular designs that separate high-level reasoning from low-level control. These approaches maintain an explicit skill library or retrieval mechanism and use foundation models for task decomposition and skill selection, while low-level control is delegated to separate execution modules. The framework proposed in this thesis follows a similar decomposition but instantiates the low-level layer with TP-KMPs learned from

demonstrations, and uses a pretrained VLM solely for natural language understanding and skill selection. RoboPrompt also employs a modular architecture but uses the LLM to directly predict low-level actions and a distinct model for perception.

Largescale Training. OpenVLA and SayCan require training or fine-tuning on substantial robot-relevant datasets. OpenVLA is trained end-to-end on 970k demonstrations, while SayCan necessitate training value functions. RSLS trains diffusion policies on consumer-grade GPUs for 10h, which is also considered largescale in comparison. In the case of RAP, low-level policy networks require training for each task and there is the need to build a memory from successful trajectories, which is also considered largescale training. In contrast, RoboPrompt, VocalSandbox, CLEA, and the present work avoid largescale training or fine-tuning, instead relying on off-the-shelf pretrained LLMs or VLMs for high-level reasoning and skill selection.

Model Accessibility. SayCan depends on the proprietary 540B `PaLM` model, while VocalSandbox uses `GPT-3.5 Turbo` and RSLS employs `GPT-4` and `Gemini` for skill selection. In contrast, the other approaches use non commercial or non proprietary models. OpenVLA is trained on a `Llama` model, CLEA uses `Qwen` models, RAP can operate with open-source models such as `LLaVA`, RoboPrompt and the framework represented in this thesis are designed to work with open-source VLMs.

Data Efficiency. The reviewed frameworks exhibit substantial variation in their data requirements. SayCan represents the most data-intensive approach, requiring 68,000 teleoperated demonstrations plus 12,000 autonomous episodes to train its underlying policies and value functions. OpenVLA requires 970k demonstrations for initial training, though task adaptation can be achieved with 10–150 demonstrations. RSLS acquires each new skill via 50–150 teleoperated demonstrations. In addition to memory constructed from successful trajectories, RAP requires 25 demonstrations per task to train a policy network to select the correct low-level action, plus what is needed to build the low-level action. In contrast, CLEA and VocalSandbox operate with predefined skill pools, avoiding per-skill training but limiting flexibility. The ICL approach RoboPrompt circumvents fine-tuning entirely, while employing 10 examples per prompt. The presented framework achieves skill acquisition with only 3–6 kinesthetic demonstrations per skill without requiring foundation model fine-tuning or access to large GPU clusters.

Skill Representation. The reviewed frameworks employ diverse skill representations. VocalSandbox uses DMPs for low-level skills, while RSLS trains visuomotor diffusion policies that implicitly encode spatial reasoning but remain bound to specific objects. SayCan relies on learned policies paired with value functions that estimate execution feasibility. RAP combines retrieval-based action selection with separately trained MLPs to map the high-level plan to low-level executable actions, but it is not stated how these low-level actions are represented. CLEA draws from a predefined skill pool without learned motion primitives. The presented framework utilizes TP-KMPs, which provide generalization through coordinate frame transformations. OpenVLA must implicitly learn spatial reasoning through extensive training data. RoboPrompt represents skills as sequences of keyframe poses described in natural language.

Runtime Skill Acquisition. The only approaches supporting runtime skill acquisition in

a narrow sense are the presented framework and RSLS. In a broader sense, VocalSandbox allows users to define new high-level functions by composing existing low-level actions, though this does not extend to acquiring new low-level skills. The other frameworks do not support runtime skill acquisition, relying instead on predefined skill libraries, extensive offline training or ICL information contained within the prompt.

Persistent Skill Representation. Most of the approaches support skill representations, where learned skills are persistently stored and reusable across different contexts. The presented framework, OpenVLA, RSLS, SayCan, and RAP all maintain libraries of learned skills or memory that can be invoked as needed. Notably, there is no information how the low-level action in RAP are stored. In contrast, VocalSandbox does not store newly defined high-level functions persistently and they exist only within the current session, while the low-level function remain. CLEA operates with a fixed skill pool without per-skill training, thus it is to assume that these skills are reusable across tasks. RoboPrompt does not maintain a persistent skill library, as movements are generated on-the-fly through ICL without storage.

Combination of Skills. The presented framework uniquely supports the combination of existing skills into new ones through probabilistic fusion. VocalSandbox allows high-level function composition but is limited to predefined low-level actions. RSLS does not support skill combination, as each skill is represented by a separate diffusion policy. SayCan, CLEA, and RAP also rely on discrete skill selection, which can be sequenced differently to achieve varied tasks but do not support the fusion of multiple skills into a single new skill. OpenVLA and RoboPrompt do not provide explicit mechanisms for skill combination.

Closed-loop Adaptation. The frameworks differ in their capacity for runtime adaptation and error recovery. CLEA employs a dedicated critic module that continuously assesses action feasibility and triggers replanning when execution conditions change or failures are detected. SayCan's affordance functions provide implicit adaptivity by scoring skill feasibility based on the current state, though without explicit replanning mechanisms. RAP adapts through retrieval, selecting relevant past experiences based on the current observation. OpenVLA operates in a closed-loop fashion at the action level, continuously predicting single-step actions. The remaining frameworks, including VocalSandbox, RSLS, and RoboPrompt, primarily execute skills open-loop once selected. The presented framework similarly executes skills open-loop, relying on the robustness of the underlying TP-KMP rather than active replanning.

These architectural choices reflect different trade-offs between generality, data efficiency, and deployment complexity. The evaluation presented in chapter 5 demonstrates how these design decisions manifest in practical performance.

3. Background

3.1. KMP

The KMP [25] is a probabilistic framework for learning and generalizing movement patterns in robotic systems. It aims to combine the strengths of multiple existing approaches, such as DMPs [85], GMMs [23], and Probabilistic Movement Primitives (ProMPs) [86]. By using kernel functions, KMPs can handle high-dimensional input spaces while preserving the probabilistic properties of the demonstrated trajectories. Furthermore, they allow the incorporation of additional constraints, such as via-points, to adapt learned movements to new situations. Moreover, KMPs can extrapolate movements beyond the range of the training data.

The KMP framework consists of a learning phase and a prediction phase. In the learning phase, the model receives a set of demonstrations containing input-output pairs. The KMP fits a probabilistic reference trajectory to these demonstrations, encapsulating the variability between them. The prediction phase begins by defining a parametric trajectory distribution, which is aligned with the reference trajectory by minimizing the KL-Divergence [87, 24]. Given a new input query, this optimization problem yields predictions for both the mean and covariance of the trajectory.

As mentioned above, KMPs allow for the incorporation of constraints such as via-points, but for this work, only the learning and prediction phases are of interest and will be explained in the following. An overview of the process is given in Algorithm 1.

3.1.1. Preliminaries

Let $\mathbf{s}_{n,h} \in \mathbb{R}^{\mathcal{I}}$ be the input and $\boldsymbol{\xi}_{n,h} \in \mathbb{R}^{\mathcal{O}}$ be the output, such that $\{\{\mathbf{s}_{n,h}, \boldsymbol{\xi}_{n,h}\}_{n=1}^{N}\}_{h=1}^{H}$. The superscripts H and N denote the number of demonstrations and the length of each demonstration, respectively. The KMP uses these demonstrations to estimate the relation between s and ξ. Importantly, the probabilistic encoding of the demonstrations allows s and ξ to represent different types of variables. For example, the input could represent the robot's End-Effector (EE) position, while the output corresponds to its velocity. Alternatively, the input and output could denote time and the robot's EE pose, respectively.

The KMP leverages a GMM to encode the training data, i.e., the demonstrations. To this end, the GMM is used to estimate the joint probability distribution $\mathcal{P}(s, \xi)$ from the demonstrations, resulting in

$$\begin{bmatrix} \mathbf{s} \\ \boldsymbol{\xi} \end{bmatrix} \sim \sum_{c=1}^{C} \pi_c \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c). \tag{3.1}$$

Here, π_c, $\boldsymbol{\mu}_c$, and $\boldsymbol{\Sigma}_c$ denote the prior probability, mean, and covariance of the c-th Gaussian

component, with C representing the total number of Gaussians. Using GMR [28], a probabilistic reference trajectory can be computed as $\{\hat{\boldsymbol{\xi}}_n\}_{n=1}^{N}$. Each point $\hat{\boldsymbol{\xi}}_{\boldsymbol{n}}$ in the trajectory can be described with the conditional probability distribution

$$\hat{\boldsymbol{\xi}}_{\boldsymbol{n}}|\boldsymbol{s_n} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n), \tag{3.2}$$

where $\hat{\boldsymbol{\mu}}_n$ is the mean and $\hat{\boldsymbol{\Sigma}}_n$ is the covariance.

The goal of the KMP is to minimize the KL-Divergence between the probabilistic reference trajectory $\mathcal{P}_R(\xi|s_n)$ and the probability distribution of the parametric trajectory $\mathcal{P}_P(\xi|s_n)$, where

$$\mathcal{P}_R(\boldsymbol{\xi}|\mathbf{s}_n) = \mathcal{N}(\boldsymbol{\xi}|\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n) \tag{3.3}$$

is derived from (3.2), and

$$\mathcal{P}_P(\boldsymbol{\xi}|\mathbf{s}_n) = \mathcal{N}\big(\boldsymbol{\xi}|\boldsymbol{\Theta}(\mathbf{s}_n)^T\boldsymbol{\mu}_w, \boldsymbol{\Theta}(\mathbf{s}_n)^T\boldsymbol{\Sigma}_w\boldsymbol{\Theta}(\mathbf{s}_n)\big) \tag{3.4}$$

will be derived in the following.

Denote the parametric trajectory

$$\boldsymbol{\xi}(\mathbf{s}) = \boldsymbol{\Theta}(\mathbf{s})^T\mathbf{w}, \tag{3.5}$$

with $\boldsymbol{\Theta}(s) \in \mathbb{R}^{\mathcal{B}\mathcal{O}\times\mathcal{O}}$ being the matrix

$$\boldsymbol{\Theta}(\mathbf{s}) = \begin{bmatrix} \mathbf{b}(\mathbf{s}) & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{b}(\mathbf{s}) & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{b}(\mathbf{s}) \end{bmatrix}, \tag{3.6}$$

where $\mathbf{w} \in \mathbb{R}^{\mathcal{B}\mathcal{O}}$ is the weight vector and $\mathbf{b}(s) \in \mathbb{R}^{\mathcal{B}}$ denotes the B-dimensional basis functions. Consider the weight vector to be normally distributed $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$, with unknown mean $\boldsymbol{\mu}_w$ and covariance $\boldsymbol{\Sigma}_w$, such that

$$\boldsymbol{\xi}(\mathbf{s}) \sim \mathcal{N}\big(\boldsymbol{\Theta}(\mathbf{s})^T\boldsymbol{\mu}_w, \boldsymbol{\Theta}(\mathbf{s})^T\boldsymbol{\Sigma}_w\boldsymbol{\Theta}(\mathbf{s})\big) \tag{3.7}$$

holds and leads to (3.4).

Thus, the objective function of the minimization problem is formulated as

$$J_{ini}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w) = \sum_{n=1}^{N} D_{KL}\big(\mathcal{P}_P(\boldsymbol{\xi}|\mathbf{s}_n), \mathcal{P}_R(\boldsymbol{\xi}|\mathbf{s}_n)\big), \tag{3.8}$$

where $D_{KL}(\cdot,\cdot)$ denotes the KL-Divergence and is defined as

$$D_{KL}(\mathcal{P}_P(\boldsymbol{\xi}|\mathbf{s}_n), \mathcal{P}_R(\boldsymbol{\xi}|\mathbf{s}_n)) = \int \mathcal{P}_P(\boldsymbol{\xi}|\mathbf{s}_n) \log\left(\frac{\mathcal{P}_P(\boldsymbol{\xi}|\mathbf{s}_n)}{\mathcal{P}_R(\boldsymbol{\xi}|\mathbf{s}_n)}\right) d\boldsymbol{\xi}. \tag{3.9}$$

The objective function can be rewritten using the trace $\mathrm{Tr}(\cdot)$ and the determinant $|\cdot|$ of a matrix as

$$J_{ini}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w) = \sum_{n=1}^{N} \frac{1}{2} \Big(\log\left(|\hat{\boldsymbol{\Sigma}}_n|\right) - \log\left(|\boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\Sigma}_w \boldsymbol{\Theta}(\mathbf{s}_n)|\right) - \mathcal{O} + \mathrm{Tr}\left(\hat{\boldsymbol{\Sigma}}_n^{-1} \boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\Sigma}_w \boldsymbol{\Theta}(\mathbf{s}_n)\right) + \left(\boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n\right)^T \hat{\boldsymbol{\Sigma}}_n^{-1} \left(\boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n\right) \Big). \quad (3.10)$$

By removing the constant parts $\frac{1}{2}$, $\log(|\hat{\boldsymbol{\Sigma}}_n|)$, and $\mathcal{O}$, (3.10) can be decomposed into two subproblems regarding the mean and the covariance, respectively. This results in

$$J_{ini}(\boldsymbol{\mu}_w) = \sum_{n=1}^{N} \left(\boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n\right)^T \hat{\boldsymbol{\Sigma}}_n^{-1} \left(\boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n\right) \quad (3.11)$$

for the mean and

$$J_{ini}(\boldsymbol{\Sigma}_w) = \sum_{n=1}^{N} \Big(- \log\left(|\boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\Sigma}_w \boldsymbol{\Theta}(\mathbf{s}_n)|\right) + \mathrm{Tr}\left(\hat{\boldsymbol{\Sigma}}_n^{-1} \boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\Sigma}_w \boldsymbol{\Theta}(\mathbf{s}_n)\right) \Big) \quad (3.12)$$

for the covariance term. The two resulting problems will be solved separately.

3.1.2. Prediction of Mean

To prevent over-fitting, the regularization term $||\boldsymbol{\mu}_w||^2$ is introduced in the mean minimization subproblem (3.11), resulting in the new mean minimization subproblem

$$J(\boldsymbol{\mu}_w) = \sum_{n=1}^{N} \left(\boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n\right)^T \hat{\boldsymbol{\Sigma}}_n^{-1} \left(\boldsymbol{\Theta}(\mathbf{s}_n)^T \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n\right) + \lambda \boldsymbol{\mu}_w^T \boldsymbol{\mu}_w, \quad (3.13)$$

with $\lambda > 0$. The cost function (3.13) has the form of a weighted least squares optimization problem if the regularization term $||\boldsymbol{\mu}_w||^2$ is ignored. Furthermore, the information contained in $\hat{\boldsymbol{\Sigma}}_n$ allows for significant variations from the reference trajectory points when their associated uncertainties are high, whereas low covariances enforce proximity to the reference trajectory. Thus, $\hat{\boldsymbol{\Sigma}}_n$ can be interpreted as a relevance metric for each data point in the trajectory, effectively adjusting the optimization pressure for each point. Deriving the optimal solution $\boldsymbol{\mu}_w^*$ (see [25] in section *2.2 Kernelized Movement Primitive (KMP)*) leads to

$$\boldsymbol{\mu}_w^* = \boldsymbol{\Phi}(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{\Sigma})^{-1} \boldsymbol{\mu}, \quad (3.14)$$

with

$$\begin{aligned} \boldsymbol{\Phi} &= \left[\boldsymbol{\Theta}(\mathbf{s}_1), \boldsymbol{\Theta}(\mathbf{s}_2), \ldots, \boldsymbol{\Theta}(\mathbf{s}_N)\right], \\ \boldsymbol{\Sigma} &= \mathrm{blockdiag}\left(\hat{\boldsymbol{\Sigma}}_1, \hat{\boldsymbol{\Sigma}}_2, \ldots, \hat{\boldsymbol{\Sigma}}_N\right), \\ \boldsymbol{\mu} &= \left[\hat{\boldsymbol{\mu}}_1^T, \hat{\boldsymbol{\mu}}_2^T, \ldots, \hat{\boldsymbol{\mu}}_N^T\right]^T. \end{aligned} \quad (3.15)$$

Consequently, the expected value, namely the output for a new input $\mathbf{s}^*$, is computed as

$$\begin{aligned}\mathbb{E}\big(\boldsymbol{\xi}(\mathbf{s}^*)\big) &= \boldsymbol{\Theta}(\mathbf{s}^*)^T \boldsymbol{\mu}_w^* \\ &= \boldsymbol{\Theta}(\mathbf{s}^*)^T \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}.\end{aligned} \tag{3.16}$$

The KMP leverages kernel functions to handle high-dimensional input, which is now applied to rewrite (3.16). The inner product of basis functions forms a kernel function $k(\cdot,\cdot)$ as

$$\mathbf{b}(\mathbf{s}_i)^T\mathbf{b}(\mathbf{s}_j) = k(\mathbf{s}_i,\mathbf{s}_j). \tag{3.17}$$

Applying the kernel function to (3.6) yields the kernel matrix $\mathbf{k}(\mathbf{s}_i,\mathbf{s}_j)$

$$\begin{aligned}\mathbf{k}(\mathbf{s}_i,\mathbf{s}_j) &= \boldsymbol{\Theta}(\mathbf{s}_i)^T\boldsymbol{\Theta}(\mathbf{s}_j) \\ &= \begin{bmatrix} k(\mathbf{s}_i,\mathbf{s}_j) & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & k(\mathbf{s}_i,\mathbf{s}_j) & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & k(\mathbf{s}_i,\mathbf{s}_j) \end{bmatrix} \\ &= k(\mathbf{s}_i,\mathbf{s}_j)\mathbf{I}_{\mathcal{O}},\end{aligned} \tag{3.18}$$

where $\mathbf{I}_{\mathcal{O}}$ is the $\mathcal{O}$-dimensional identity matrix. Using this result, the matrices

$$\mathbf{K} = \begin{bmatrix} \mathbf{k}(\mathbf{s}_1,\mathbf{s}_1) & \mathbf{k}(\mathbf{s}_1,\mathbf{s}_2) & \cdots & \mathbf{k}(\mathbf{s}_1,\mathbf{s}_N) \\ \mathbf{k}(\mathbf{s}_2,\mathbf{s}_1) & \mathbf{k}(\mathbf{s}_2,\mathbf{s}_2) & \cdots & \mathbf{k}(\mathbf{s}_2,\mathbf{s}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{k}(\mathbf{s}_N,\mathbf{s}_1) & \mathbf{k}(\mathbf{s}_N,\mathbf{s}_2) & \cdots & \mathbf{k}(\mathbf{s}_N,\mathbf{s}_N) \end{bmatrix} \tag{3.19}$$

and

$$\mathbf{k}^* = [\mathbf{k}(\mathbf{s}^*,\mathbf{s}_1)\ \mathbf{k}(\mathbf{s}^*,\mathbf{s}_2)\ \cdots\ \mathbf{k}(\mathbf{s}^*,\mathbf{s}_N)] \tag{3.20}$$

are defined to rewrite (3.16) as

$$\begin{aligned}\mathbb{E}(\boldsymbol{\xi}(\mathbf{s}^*)) &= \tilde{\boldsymbol{\mu}} \\ &= \mathbf{k}^*(\mathbf{K} + \lambda\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}.\end{aligned} \tag{3.21}$$

It is noteworthy that (3.21) equals the mean of the GPR [24] if $\hat{\boldsymbol{\Sigma}}_n = \mathbf{I}_{\mathcal{O}}$. Furthermore, the optimal solution $\boldsymbol{\mu}_w^*$ of the initial mean minimization subproblem (3.11) represents the best estimate given the demonstrations from the reference trajectory distribution (3.3). This is because (3.11) is equivalent to maximizing the posterior $\prod_{n=1}^{N}\mathcal{P}(\boldsymbol{\Theta}(\mathbf{s}_n)^T\boldsymbol{\mu}_w|\hat{\boldsymbol{\mu}}_n,\hat{\boldsymbol{\Sigma}}_n)$ (see [25] in Appendix B).

3.1.3. Prediction of Covariance

For the same reasons as in subsection 3.1.2 a regularization term gets introduced in the initial minimization subproblem. Thus, the term $\mathrm{Tr}(\boldsymbol{\Sigma}_w)$ gets inserted in (3.12) resulting in (3.22).

Algorithm 1 Prediction using KMPs

1: **Initialization**
- Denote $k(\cdot,\cdot)$ and choose λ.

2: **Learning from demonstrations**
- Collect demonstrations $\{\{\mathbf{s}_{n,h}, \boldsymbol{\xi}_{n,h}\}_{n=1}^{N}\}_{h=1}^{H}$.
- Extract the reference database $\{\mathbf{s}_n, \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n\}_{n=1}^{N}$.

3: **Computing Prediction**
- Receive new input query $\mathbf{s}^*$
- Update $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, $\mathbf{K}$ and $\mathbf{k}^*$
- Compute the output via
  $\mathbb{E}(\boldsymbol{\xi}(\mathbf{s}^*)) = \mathbf{k}^*(\mathbf{K}+\lambda\mathbf{I})^{-1}\boldsymbol{\mu}$ and
  $\mathbb{D}(\boldsymbol{\xi}(\mathbf{s}^*)) = \frac{N}{\lambda}\left(\mathbf{k}(\mathbf{s}^*,\mathbf{s}^*) - \mathbf{k}^*(\mathbf{K}+\lambda\boldsymbol{\Sigma})^{-1}\mathbf{k}^{*T}\right)$

Note that the largest eigenvalue of $\boldsymbol{\Sigma}_w$ is smaller than regularization term, because $\boldsymbol{\Sigma}_w$ is positive definite.

$$J(\boldsymbol{\Sigma}_w) = \sum_{n=1}^{N}\left(-\log\left(|\boldsymbol{\Theta}(\mathbf{s}_n)^T\boldsymbol{\Sigma}_w\boldsymbol{\Theta}(\mathbf{s}_n)|\right) + \operatorname{Tr}\left(\boldsymbol{\Sigma}_w^{-1}\boldsymbol{\Theta}(\mathbf{s}_n)^T\hat{\boldsymbol{\Sigma}}_n\boldsymbol{\Theta}(\mathbf{s}_n)\right)\right) + \lambda\operatorname{Tr}(\boldsymbol{\Sigma}_w) \tag{3.22}$$

In the next step the derivative of (3.22) with respect to $\boldsymbol{\Sigma}_w$ gets computed (see [25] chapter *Covariance Prediction of KMP*) and it is set to 0, resulting in

$$\sum_{n=1}^{N}\left(-\boldsymbol{\Sigma}_w^{-1} + \boldsymbol{\Theta}(\mathbf{s}_n)\hat{\boldsymbol{\Sigma}}_n^{-1}\boldsymbol{\Theta}(\mathbf{s}_n)^T\right) + \lambda\mathbf{I} = \mathbf{0}. \tag{3.23}$$

This can be reformulated using the definitions from (3.15) to obtain the optimal soulution $\boldsymbol{\Sigma}_w^*$ as

$$\boldsymbol{\Sigma}_w^* = N(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^T + \lambda\mathbf{I})^{-1}. \tag{3.24}$$

Note that, the structure of this solution mirrors that of the covariance in weighted least squares estimation, with the distinction of including a multiplicative factor N and a regularization term $\lambda\mathbf{I}$. Using the *Woodbury identity*, which is defined as

$$(\mathbf{A}+\mathbf{C}\mathbf{B}\mathbf{C}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{C}(\mathbf{B}^{-1}+\mathbf{C}^T\mathbf{A}^{-1}\mathbf{C})^{-1}\mathbf{C}^T\mathbf{A}^{-1}, \tag{3.25}$$

the covariance corresponding to $\xi(\mathbf{s}^*)$ for a given query $\mathbf{s}^*$ is obtained as

$$\begin{aligned}\mathbb{D}(\boldsymbol{\xi}(\mathbf{s}^*)) &= \boldsymbol{\Theta}(\mathbf{s}^*)^T\boldsymbol{\Sigma}_w^*\boldsymbol{\Theta}(\mathbf{s}^*)\\ &= N\boldsymbol{\Theta}(\mathbf{s}^*)^T(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^T+\lambda\mathbf{I})^{-1}\boldsymbol{\Theta}(\mathbf{s}^*)\\ &= \frac{N}{\lambda}\boldsymbol{\Theta}(\mathbf{s}^*)^T\left(\mathbf{I}-\boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\lambda\boldsymbol{\Sigma})^{-1}\boldsymbol{\Phi}^T\right)\boldsymbol{\Theta}(\mathbf{s}^*).\end{aligned} \tag{3.26}$$

Through application of the kernel matrix defined in (3.18) and (3.19) the covariance can be defined as

$$\begin{aligned}\mathbb{D}(\boldsymbol{\xi}(\mathbf{s}^*)) &= \tilde{\boldsymbol{\Sigma}}\\ &= \frac{N}{\lambda}\left(\mathbf{k}(\mathbf{s}^*,\mathbf{s}^*) - \mathbf{k}^*(\mathbf{K}+\lambda\boldsymbol{\Sigma})^{-1}\mathbf{k}^{*T}\right).\end{aligned} \tag{3.27}$$

Apart from the scaling factor $\frac{N}{\lambda}$, the covariance formula in (3.27) has two notable properties. First, the demonstrated variability $\mathbf{\Sigma}$ extracted from training data (see (3.15)) appears in the term $(\mathbf{K} + \lambda\mathbf{\Sigma})^{-1}$, integrating task-specific uncertainty information. Second, KMP predicts a full covariance matrix, which enables modeling of correlations between output components rather than assuming independence.

3.2. TP-KMP

The Task-Parameterized Kernelized Movement Primitive (TP-KMP) extends the KMP framework to task-parameterized settings by associating multiple local KMPs with coordinate frames attached to task-relevant objects [25, 27, 26, 28]. As in the standard KMP, any choice of input and output variables can be modeled (e.g., time, joint states, Cartesian poses). In this work, time is used as input and the position and orientation of the robot's EE as output. For simplicity, TP-KMP is explained using only these input and output variables.

Conceptually, a single KMP encodes a global mapping from input to output that is tied to one nominal task configuration. In contrast, TP-KMP introduces a set of P local reference frames, each anchored to an object of interest, and learns one *local* KMP per frame from demonstrations transformed into that frame [27]. These frames are placed at the poses of objects of interest, and their positions and orientations relative to a *global* coordinate frame (the robot's base frame in this work) define where the *local* KMPs operate. At reproduction time, each *local* KMP performs its own prediction as derived in section 3.1, and all *local* predictions are subsequently fused into a single *global* distribution that defines the executed trajectory in the robot base frame. This task-parameterized formulation allows the same skill to generalize to new object poses by updating the frame parameters without retraining the underlying KMPs, thereby enabling the TP-KMP to predict the robot's trajectory in changing environments. This approach differs from parametric methods such as TP-GMM by retaining the non-parametric, kernel-based representation of KMP [26, 28, 27].

Each frame $p = 1, \ldots, P$ of the TP-KMP is defined through the task parameters $\mathbf{v}^{(p)}, \mathbf{A}^{(p)}$, which represent an object's position and orientation with respect to the robot's base frame. The demonstrations to train the TP-KMP are recorded in this global frame. These demonstrations are mapped into each local frame using

$$\boldsymbol{\xi}^{(p)} = {\mathbf{A}^{(p)}}^{-1}\left(\boldsymbol{\xi} - \mathbf{v}^{(p)}\right) \tag{3.28}$$

Since the input is time, it does not need to be transformed for each frame.

After all local KMPs have made their predictions using (3.21) and (3.27), these are transformed back into the global frame. Since the task parameters can change over time t during prediction and may differ from those used for the demonstrations, the transformations are given by

$$\tilde{\boldsymbol{\mu}}_t^{(p)} = \mathbf{A}_t^{(p)}\tilde{\boldsymbol{\mu}}^{(p)} + \mathbf{v}_t^{(p)} \tag{3.29}$$

and

$$\tilde{\mathbf{\Sigma}}_t^{(p)} = \mathbf{A}_t^{(p)}\tilde{\mathbf{\Sigma}}^{(p)}{\mathbf{A}_t^{(p)}}^T. \tag{3.30}$$

Algorithm 2 Prediction using TP-KMPs

1: **Initialization**
- Denote $k(\cdot,\cdot)$ and choose λ.
- Define (p) local frames $\{\mathbf{A}^{(p)}, \mathbf{v}^{(p)}\}_{p=1}^{(p)}$.

2: **Learning from demonstrations**
- Collect demonstrations $\{\{\mathbf{s}_{n,h}, \boldsymbol{\xi}_{n,h}\}_{n=1}^{N}\}_{h=1}^{H}$ in the *global* frame
- Transform demonstrations into *local* frames using (3.28)
- Extract the reference databases for each frame $\{\{\mathbf{s}_n, \boldsymbol{\mu}_n^{(p)}, \boldsymbol{\Sigma}_n^{(p)}\}_{n=1}^{N}\}_{p=1}^{(p)}$.

3: **Computing global prediction**
- Receive new input query $\mathbf{s}^*$
- Each frame makes a *local* prediction with
  $\mathbb{E}(\boldsymbol{\xi}(\mathbf{s}^*))^{(p)} = \tilde{\boldsymbol{\mu}}^{(p)} = \mathbf{k}^{*(p)}(\mathbf{K}^{(p)} + \lambda\mathbf{I})^{-1}\boldsymbol{\mu}^{(p)}$ and
  $\mathbb{D}(\boldsymbol{\xi}(\mathbf{s}^*))^{(p)} = \tilde{\boldsymbol{\Sigma}}^{(p)} = \frac{N}{\lambda}\left(k(\mathbf{s}^*,\mathbf{s}^*) - \mathbf{k}^{*(p)}(\mathbf{K}^{(p)} + \lambda\boldsymbol{\Sigma}^{(p)})^{-1}\mathbf{k}^{*(p)^T}\right)$
- Transform the *local* predictions into the *global* frame using (3.29) and (3.30).
- Compute the *global* output by fusing the predictions together with
  $\bar{\boldsymbol{\mu}}_t = \bar{\boldsymbol{\Sigma}}_t \sum_{p=1}^{(p)} \tilde{\boldsymbol{\Sigma}}_t^{(p)^{-1}} \tilde{\boldsymbol{\mu}}_t^{(p)}$ and
  $\bar{\boldsymbol{\Sigma}}_t = \left(\sum_{p=1}^{(p)} \tilde{\boldsymbol{\Sigma}}_t^{(p)^{-1}}\right)^{-1}$

Using (3.29) and (3.30), each frame predicts the mean $\tilde{\boldsymbol{\mu}}$ and covariance $\tilde{\boldsymbol{\Sigma}}$ from its own perspective. Those are fused together in the global frame using a product of Gaussians

$$\bar{\boldsymbol{\mu}}_t = \bar{\boldsymbol{\Sigma}}_t \sum_{p=1}^{P} \tilde{\boldsymbol{\Sigma}}_t^{(p)^{-1}} \tilde{\boldsymbol{\mu}}_t^{(p)}, \tag{3.31}$$

$$\bar{\boldsymbol{\Sigma}}_t = \left(\sum_{p=1}^{P} \tilde{\boldsymbol{\Sigma}}_t^{(p)^{-1}}\right)^{-1}, \tag{3.32}$$

which results again in a Gaussian distribution $\mathcal{N}(\bar{\boldsymbol{\mu}}_t, \bar{\boldsymbol{\Sigma}}_t)$.

Through (3.31) and (3.32), the predictions are fused together, giving models with lower covariance a higher influence on the global prediction. This mechanism enables the TP-KMP to extract features that can be reliably observed by a local KMP throughout the demonstrations and reflect them in the global prediction. The procedure of using the TP-KMP is outlined in Algorithm 2.

Recently, building on the original KMP formulation, which allows trajectory modulation through probabilistic via-point and end-point constraints in a single coordinate frame [25], Knauer et al. [40] introduced an interactive incremental learning framework that combines TP-KMPs with kinesthetic human feedback. Their approach enables real-time skill refinement through physical human-robot interaction, where users can guide the robot's end-effector to demonstrate desired trajectory corrections. A key contribution is that, instead of inserting via-points only in a global frame as in the standard KMP formulation, they introduce via-points

locally in the different object frames of the TP-KMP, so that corrections remain attached to the corresponding objects and naturally follow them when task conditions or object poses change.

4. Methodology

This work combines TP-KMPs with VLMs. The framework consists of a learning phase and an execution phase.

In the learning phase, the user provides physical demonstrations on the robot, which are recorded and used to train a TP-KMP. Once trained, the TP-KMP is recognized as a *skill* that can be loaded and executed.

During the execution phase, the framework takes a user prompt l as input. The VLM is leveraged to process the query and check if there is a matching skill. If a fitting skill is available, the VLM loads it accordingly. If no suitable skill exists, the VLM has the option to combine two existing skills into a new one to satisfy the user's demand, as described in section 4.4. After a skill is loaded, the VLM determines which objects are of interest and parameterizes the skill's frames.

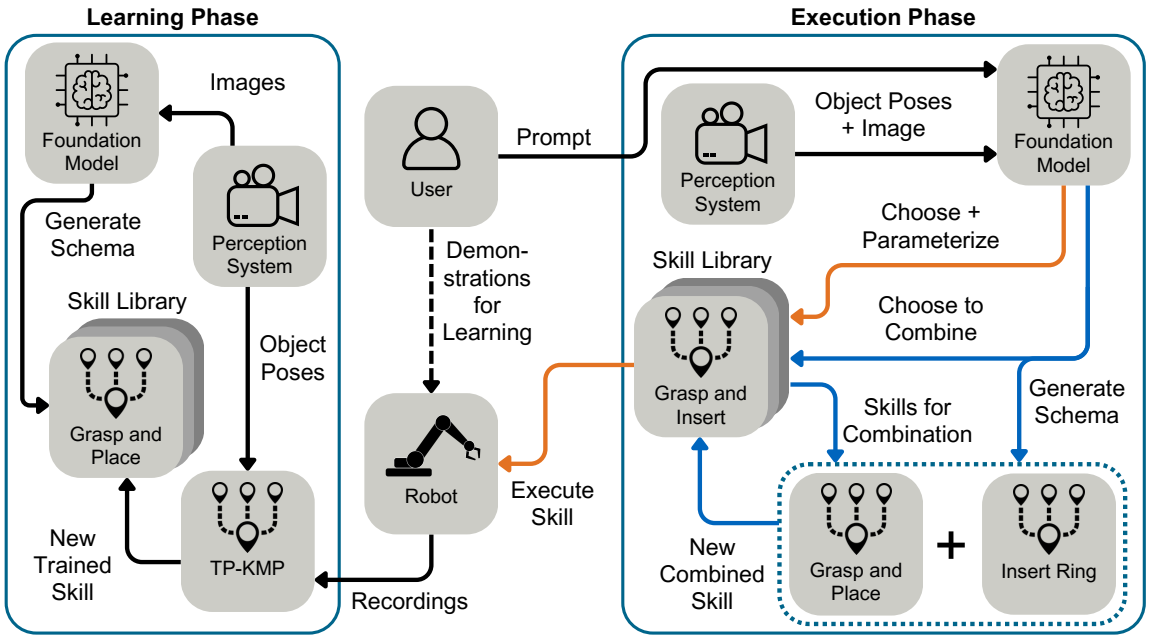

Figure 4.1.: Flowchart illustrating the learning phase and execution phase of the framework in a high-level overview. More detailed descriptions and illustrations of each component can be found in section 4.2 and section 4.3 respectively.

Subsequently, the predictions of the local KMPs are fused together to form the global prediction. The mean of the global prediction is used as the robot's trajectory.

The TP-KMP is used to generate the robot's trajectory with changing object positions. To achieve this, the TP-KMP requires the positions of the local KMPs with respect to a global base frame. The local KMP frames are placed at the object frames, while the global frame corresponds to the robot's base frame. Thus, the positions of the objects with respect to the robot define the local KMP frames of the TP-KMP. When a skill (i.e., a trained TP-KMP) is loaded, only the frames need to be parameterized to generate a prediction for a new input query (see Algorithm 2). In this work, the input query $\mathbf{s}^*$ of the TP-KMP is always the discrete and normalized time t, which is defined through a finite number of small timesteps with $\Delta t > 0$. Consequently, the only variable that can change between two predictions using the same skill is the position of the frames.

The VLM is used to determine which objects are relevant to the user's query. Since VLMs struggle to generate accurate 6D poses of objects from images, and since a key aspect of this work is avoiding fine-tuning of foundation models, an *Object Pose Estimator* (see subsection 4.1.1) is introduced. The sole purpose of this Object Pose Estimator is to extract the poses of visible objects, enabling the VLM to parameterize skills using object names. The framework handles the translation between object names and 6D poses before parameterizing the TP-KMP.

4.1. Preliminaries

4.1.1. Object Pose Estimation

The framework requires knowledge of object poses to parameterize the frames of the TP-KMP. It is possible to provide the framework with a set of known objects and their positions manually, but this would contradict the goal of automation. To address this, a perception pipeline is implemented to determine the 6D poses of objects from a 2D image using a fixed camera. The pose estimation module is not part of the core framework and can be implemented using various methods, including methods that use non-fixed cameras.

The implemented example pipeline consists of two machine learning models. First, the image is passed to a model based on `YOLOv7` [88]. The YOLO model detects objects in the image and returns bounding boxes around the detected objects. Next, a dense pose estimator [89], which utilizes 3D mesh models of the objects, is used to estimate the 6D poses of the objects detected and outlined by the bounding boxes. An overview of this pipeline is shown in Figure 4.2.

The camera is placed on the side of the robot's workspace and is fixed in position and orientation. To improve the accuracy of the intrinsic and extrinsic camera parameter estimation, the image is captured at high resolution (4K). The image is passed to the YOLO model, which is trained on the objects included in the *Yale-CMU-Berkeley* Object and Model Set (YCB Dataset) [90, 91]. Thus, the objects that can be detected and used by the pose estimator are limited to this dataset. The YCB Dataset includes many different everyday items that can be found in common US supermarkets. These objects are categorized by type, which includes 'Food item', 'Kitchen item', 'Tool item', 'Shape item', and 'Task item'. For each item, the

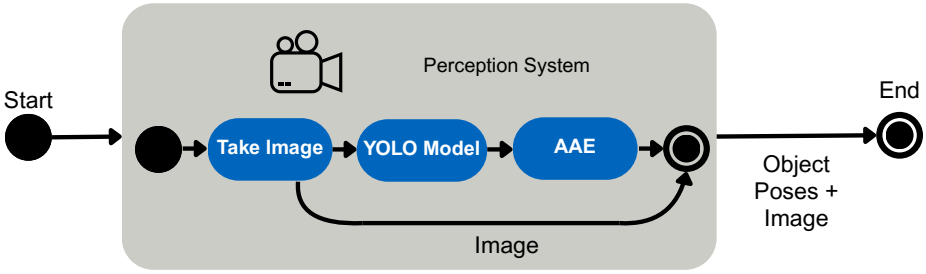

Figure 4.2.: Flowchart illustrating the object pose estimation pipeline: from image capture through object detection to 6D pose estimation.

dataset provides 600 RGB-D images, 600 high-resolution RGB images, segmentation masks for each image, calibration information for each image, and texture-mapped 3D mesh models. The output of the YOLO model consists of bounding boxes around the detected objects. For detection, a confidence score of $\kappa \geq 0.7$ is used.

The image and bounding box information are forwarded to the dense pose estimator. The key element of the dense pose estimator is the Augmented Autoencoder (AAE). The AAE is an extension of the Denoising Autoencoder (DAE) [92], which is trained with a modified strategy. Various augmentations are applied to the input images, while the reconstruction targets remain clean. The augmentations include lighting changes, color and texture variations, occlusions, background clutter, and geometric transformations. A synthetic data generation pipeline is used to generate the training images. This domain randomization forces the network to encode only the intrinsic 3D orientation of the object while becoming invariant to the augmented variations. A key advantage of this approach is that it requires no real pose-annotated training data and inherently handles symmetric objects by learning appearance-based representations. At test time, to estimate the 3D orientation, the scene crop is encoded and matched against a codebook of pre-computed latent codes from synthetic views using nearest neighbor search. Subsequently, the 3D translation is estimated by comparing the bounding box size ratio between the test image and the matched synthetic reference view. Finally, depth data is leveraged to perform a standard Iterative Closest Point (ICP) [93] refinement step to improve accuracy. An example of the estimated poses is shown in Figure 4.3, where the estimated object frames are overlaid on the captured image.

Figure 4.3.: Example output of the object pose estimation pipeline. The estimated 6D poses of detected objects are visualized by overlaying their coordinate frames on the captured image. The red, green, and blue axes represent the x-, y-, and z-axes of each object frame, respectively.

The dense pose estimator returns the 6D poses of the detected objects in the camera frame. These poses need to be transformed into the robot's base frame before they can be used to parameterize the TP-KMP. Since the camera is fixed in place, the transformation applied to the estimated 6D poses remains constant.

The transformation matrix is obtained via a hand-eye calibration between the camera and the robot. For the calibration, a board with checkerboard patterns is attached to the robot's EE. The robot moves to different positions, where an image is captured at each. The dimensions of the board, the checkerboard pattern, and the robot joint states when the images are recorded are known. Using this information, the intrinsic camera parameters are derived. Subsequently, a 4×4 transformation matrix representing the extrinsic parameters, i.e., the translational and rotational difference between the robot's base frame and the camera frame, is computed. This matrix can be used to transform the 6D poses returned by the dense pose estimator into the robot's base frame.

4.1.2. VLM Tool Calling

In this framework, the VLM has two major purposes: the selection of skills and their parameterization, and the generation of schemas that describe a skill. The latter is used for training new skills and for the combination of skills. To make the translation between user input and desired output easier for the VLM, thereby making the framework more robust, tools are leveraged. To achieve this, VLMs that are capable of handling tools are used, specifically models that are trained to use tools [94, 95, 96].

For all VLM functionality in this framework, the input is a prompt, which may include an

image depending on the task, and the output is a tool call. When the VLM calls a tool, its output must fulfill certain constraints. Fundamentally, the output is a string with a specific format. This format, typically JavaScript Object Notation (JSON), can be interpreted by the programming language at runtime, enabling the code to execute specific functionality.

Tool definitions allow to specify the expected output format of the VLM. It is possible to define the tool's attributes and whether they are required. Additionally, one can specify the type of each attribute, increasing the system's robustness. Furthermore, the docstring of a tool provides the VLM with an explanation of the tool's function. Finally, each attribute can include a docstring describing its role in the tool, making it helping the VLM to determine which value to pass.

Tools allow to define a method to validate the input provided by the VLM. This ensures that only valid values are passed to the code, preventing runtime errors and potential program crashes. If an invalid value is supplied, the VLM is given up to three attempts to correct it. This feature helps to improve robustness in tool calling, not only for data types, but also semantically For example, the validation method can be used to verify whether the objects the VLM intends to manipulate are actually detected by the perception pipeline.

In accordance with this principle, a set of tools is defined, including, for example, the tool `TellUserMessage`, which can be used by the VLM to send a message to the user. The complete set of tools, their attributes, and their functions is listed in section B.1.

Initially, a tool named `LoadSkill` was defined to enable the execution of skills. The VLM had to select the name of the skill it wanted to load from a dynamically generated list containing available skills. Moreover, it needed to set the names of the objects that are used to parameterize the skill as strings. Importantly, the order of the object names fundamentally determined the robot's behavior. Consider a skill that provides a *grasp and place* motion. The order of objects, which refer to the TP-KMP's frames, defines which object to grasp and where to place it, profoundly changing the skill's execution. It was observed that the different VLMs that were tested struggled to supply the correct values to the tool's attributes. The problem was that the VLM, based solely on the skill name, had to infer the motion the robot would execute, the number of objects used by the skill, and the order in which the objects are used in the skill. Even humans would struggle to provide the correct values to the tool with this lack of information. This is why dynamic tools were introduced to describe skills. Dynamic tools solve this problem by embedding all necessary information about a skill directly into the tool definition itself, eliminating the need for the VLM to infer behavior from skill names alone.

Dynamic Tools

Dynamic tools are not defined through code, but are generated at runtime using a schema. A schema is associated with a skill and contains information about the skill's name and function. A schema is written in Yet Another Markup Language (YAML) format and describes a tool by its name, docstring, and its attributes with their types and descriptions. The schema's attributes refer to the objects that can be used in the skill, inherently defining how many objects are involved. Moreover, the schema holds information about the order in which the

objects are interacted with. These schemas are generated either manually by the user during skill creation or automatically by the VLM when prompted to describe a demonstrated skill (see section 4.2) or while skills are combined (see section 4.4). Once created, schemas are stored alongside the trained TP-KMP as YAML files and loaded whenever the skill is made available to the system.

Leveraging the principle of dynamic tools, the *grasp and place* example is defined through a skill named `SkillGraspAndPlace`. The skill's docstring explains what the execution of the skill causes, and the two attributes are `object_to_grasp` and `target_location`. Each attribute is clarified through its own docstring, and the only information that the VLM has to provide are object names for these attributes. The correct order of objects for skill parameterization is handled through the `object_order` field of the schema. The `object_order` field contains an ordered list of attribute names, ensuring that objects are assigned to the TP-KMP's frames in the correct sequence, regardless of the order in which the VLM provides them. In Listing 4.1, the schema generated by the model `Qwen2.5-VL-72B-Instruct` [97] for the *grasp and place* motion can be seen. The complete input for the generation of the schema is listed in subsection B.2.1.

Listing 4.1: Generated schema for the *grasp and place* motion

```
action_name: "SkillStackObjects"

explanation: "This skill allows the robot to stack one object on top of another. The robot first identifies and
      grasps the object_to_stack, then carefully places it on top of the base_object. This ensures a stable stack of
      the two objects."

possible_fields:
  object_to_stack:
    type: "str"
    description: "The name of the object that should be stacked on top. Choose an object from the list of
          detected objects."
  base_object:
    type: "str"
    description: "The name of the object that serves as the base for stacking. Choose an object from the list of
          detected objects."

object_order:
- "self.object_to_stack"
- "self.base_object"
```

4.1.3. Safe, Autononomous Robotic Assistant

The robot used to embody the framework is SARA, a 7-Degree of Freedom (DoF) collaborative robotic arm. The robot weighs 22.6 kg, has a maximum reach of 1250 mm when fully extended, and supports a nominal payload of 12 kg. Its workspace is defined by a spherical shell with an inner radius of 297 mm and outer radius of 1024 mm at the wrist.

SARA is equipped with comprehensive sensory capabilities, including torque sensors in all seven joints (force resolution better than 0.1 N), force-torque sensors in the base and wrist,

and two Inertial Measurement Units (IMUs). These sensors enable fast and reliable collision detection [98] with limited collision torques, ensuring safe human-robot collaboration. The robot's joints can achieve angular velocities of up to 400 °/s.

A key feature for this work is SARA's ability to perform kinesthetic teaching through direct physical interaction. The integrated torque sensors allow users to physically guide the robot during the demonstration phase (see section 4.2), which is essential for training the TP-KMP. SARA communicates via the Links and Nodes middleware over Ethernet and supports multiple control modes. The 7-DoF configuration provides kinematic redundancy for avoiding singular configurations and obstacles while maintaining the desired EE pose.

SARA supports various EE tools that can be attached to the flange, including screwdrivers, grinders, and pinching grippers. A sensor port (Peripheral Component Interface Express (PCIe)) enables integration of optical sensors. In this work, a pinching gripper is assumed to be mounted at all times, and tool changes between tasks are not considered.

On the link between the sixth and seventh joint, a display, Light-Emitting Diode (LEDs), and buttons are incorporated. The buttons, LEDs, and display are programmable and can be used for user interaction.

More information about SARA can be found here.

4.2. Learning Phase

The learning phase comprises two main steps: demonstration collection and TP-KMP training. Demonstrations are necessary because the TP-KMP is a task-parameterized version of the KMP, which is an IL model. The TP-KMP learns the relationship between task parameters and robot motions from these demonstrations. During execution, the TP-KMP generates the robot's trajectory based on the learned model and the current task parameters.

4.2.1. Demonstration Collection

To collect demonstrations, the robot switches to *zero-gravity mode*, which compensates for gravitational forces on each link. Specifically, the commanded torque vector $\boldsymbol{\tau} \in \mathbb{R}^n$ applied to the robot joints is

$$\boldsymbol{\tau} = \mathbf{g}(\mathbf{q}), \tag{4.1}$$

where $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^n$ represents the gravity torque vector, which depends on the robot's joint configuration $\mathbf{q} \in \mathbb{R}^n$, with n being the number of joints. Under this control mode, the robot can be easily moved by the user, enabling kinesthetic teaching through direct physical guidance.

Once the user initiates the demonstration recording process, an image of the scene is captured. Using the perception pipeline (see subsection 4.1.1), objects are detected and their poses are estimated. These are saved together with the demonstration trajectory data.

With *zero-gravity mode* enabled, the user can use the buttons on the last link to start, pause, resume, stop, and cancel the process of recording a demonstration. The LEDs provide feedback to the user about the process state. Additionally, the buttons can be used to close

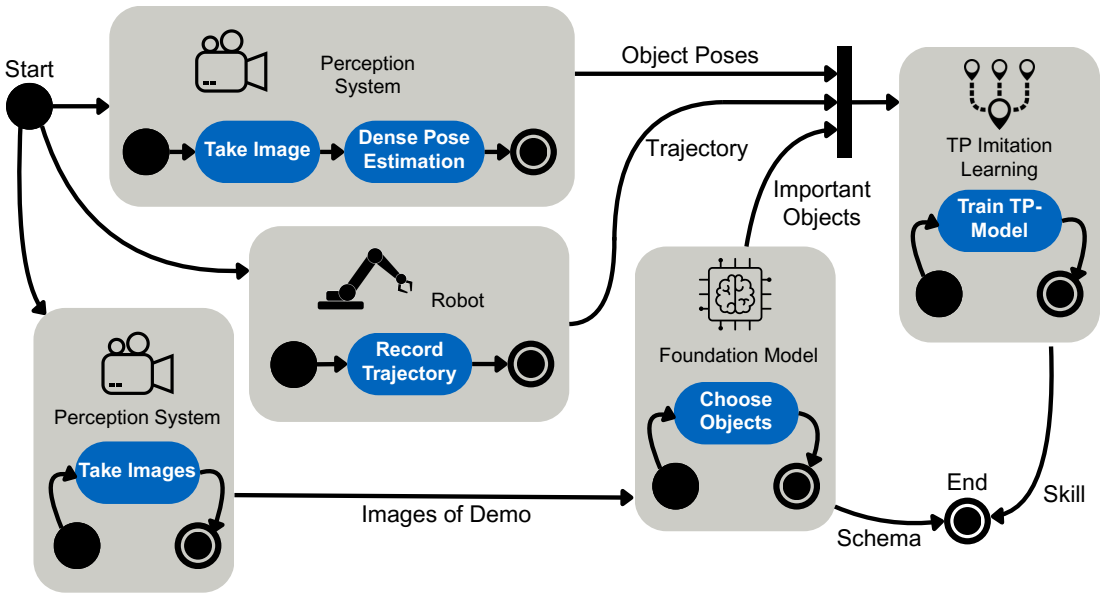

Figure 4.4.: Flowchart illustrating the learning phase workflow: from kinesthetic demonstration and object pose estimation through TP-KMP training to schema generation and skill storage.

and open the pinch gripper attached to the flange. After the user completes a demonstration and stops the recording, the user can rearrange the objects for the next demonstration and trigger the perception pipeline using the buttons.

During kinesthetic teaching, the 6D pose of the robot's EE is recorded at a frequency of 100 Hz. The gripper state is also recorded simultaneously. This demonstration data is used to train the TP-KMP.

Typically, a small number of demonstrations, e.g., between three and five, are sufficient per skill. Importantly, the demonstrations should include variations in all learned variables. In this work, the TP-KMP predicts a full 6D pose for all points of the trajectory. Thus, the variations between demonstrations should include both position and orientation. The importance of this fact is emphasised in section 4.4.

4.2.2. TP-KMP Training

After all demonstrations for a skill are completed, the demonstration data is saved. Each demonstration is associated with the 6D poses of the objects detected by the perception pipeline, where all detected objects become frames of the TP-KMP. The TP-KMP is initialized with the object frames and the corresponding trajectory data of the robot's EE, and is then trained with these values. The training procedure follows the methodology described in section 3.2.

Simultaneously, the VLM is prompted to generate the skill's schema containing the declarative knowledge (see subsubsection 4.1.2 and subsection B.2.1). The declarative knowledge includes the order in which the objects were interacted with. Technically, the TP-KMP is invariant to the order of the frames since the input is time, and the prediction determines where to go first. However, the information about which object is interacted with first, which corresponds to the frame order, is important for parameterization at execution time and for the combination of skills (see subsection 4.1.2 and section 4.4). The complete learning phase workflow is illustrated in Figure 4.4.

The generated declarative knowledge is validated to ensure it conforms to the YAML format, and the VLM is given up to three attempts to generate a valid schema. However, the semantic content of the declarative knowledge is not verified automatically. The user can modify the declarative knowledge by directly editing the file's content. Alternatively, it is possible to generate the schema file manually. Defining the declarative knowledge manually reduces the framework's scalability (i.e., the degree of automation), but ensures the correctness of the file's content.

4.3. Execution Phase

The execution phase consists of an ongoing conversation between the user and the framework. The framework waits for user input and responds using tool calls (see subsection 4.1.2 for more information on tool calls). Each user query initiates a perception-prediction-execution cycle, after which the framework awaits the next user input. Once the framework is initialized,

the conversation can begin.

4.3.1. Initialization

First, the VLM is initialized with a system prompt after the framework connects to the model. In the system prompt, the model receives instructions to interact with the user from the perspective of SARA and to respond exclusively through tool calls. Additionally, information about the expected tasks is provided to the VLM. The complete system prompt can be found in Listing A.1.

In the next initialization step, a connection to SARA is established. Thereafter, the framework checks if the correct gripper is attached to the robot's flange and moves the arm to a predefined position. The robot is controlled using an *Impedance Controller* [99, 100] in the task space coordinate frame. The impedance controller receives a 6D pose in Cartesian space as input and outputs force commands. The controller behaves as if the robot's Tool Center Point (TCP) were connected to the input pose via a mass-spring-damper system. The impedance controller is passivity-based and leverages SARA's torque sensors to enable collision detection, making it well suited for collaborative work with humans.

Subsequently, the provided path is checked for skills and their corresponding YAML files. Using the schemas contained in the YAML files, all dynamic tools are generated (see subsubsection 4.1.2 for more information).

4.3.2. Skill Selection and Parameterization

After the framework has been initialized, the user can begin interacting with the system. This can be done by typing into the chat field on the right side of the frontend or by using a microphone connected to the machine running the framework. Commands given via the microphone are displayed in the chat window. If the framework sends a message to the user, it is displayed on the monitor and can also be output through speakers. An illustration of the frontend is shown in Figure 5.4.

Querying the model with a prompt triggers the perception pipeline to capture an image, detect objects, and infer their positions (see subsection 4.1.1). The image and the detected object names are appended to the user's prompt, enabling the VLM to use this information.

Based on the received prompt, the VLM decides which tool to use and what values to pass. If the VLM selects a skill, the corresponding TP-KMP is loaded. As defined by the skill's schema, the VLM provides the object names to be used with the skill. The provided object names are validated by the skill's validation method to ensure that the perception pipeline successfully determined the poses of the selected objects. According to the schema's object order, the TP-KMP is then parameterized with the objects' 6D poses as its frames. Refer to subsubsection 4.1.2 for more details on dynamic tool calling.

Thereafter, the TP-KMP performs a prediction for the local frames, which are subsequently fused in the global (robot base) frame using the covariance-weighted approach described earlier. For a detailed explanation of this process, see section 3.2. The mean of the prediction is used as the robot's trajectory. The TP-KMP's predicted mean and covariance can be visualized

in the frontend. The steps executed from user prompt to global prediction are summarized in Figure 4.5.

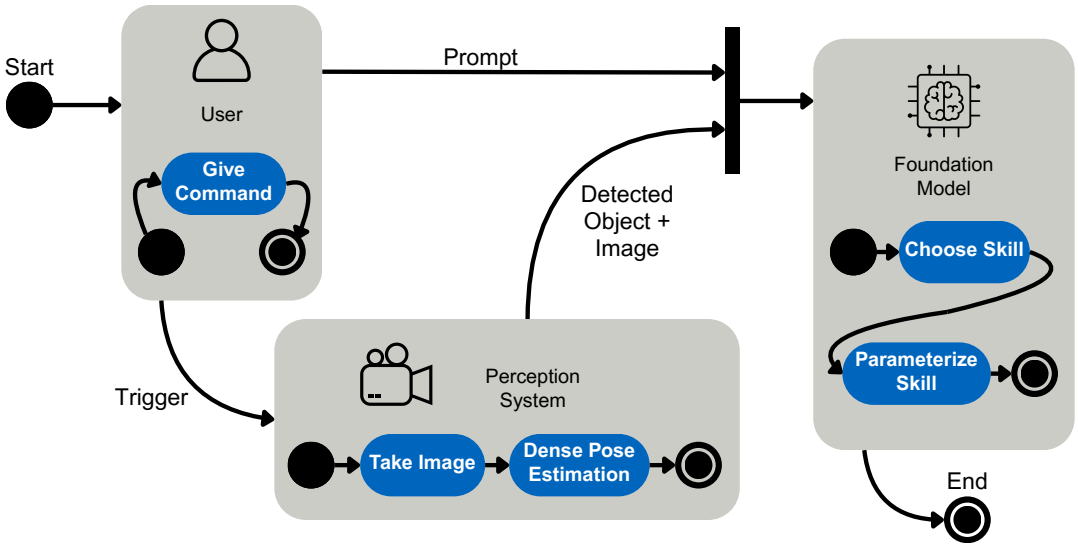

Figure 4.5.: Flowchart illustrating the execution phase cycle: from user prompt and perception through skill selection and parameterization to TP-KMP prediction and trajectory generation.

It might occur that no skill matches the users demand. In this case the framework allows to combine two existing skills into a new one, that can fulfill the task. If the VLM chooses to merge two skills together instead of executing an existing one, the new skill is made public for use immediatly after creation, leveraging dynamic tools. After that, the skill can be used by the VLM in the next iteration of the conversation cycle. The full procedure of skill combnination is explained in section 4.4.

After the trajectory is computed, it is transmitted to the impedance controller for physical execution, after which the framework returns to the listening state, awaiting further user input.

4.4. Combining Skills

Skill combination enables the merging of existing skills into a new one, thereby creating functionality that was not previously demonstrated. A new TP-KMP is created using frames (KMPs) from different skills. Since the TP-KMP fuses the local predictions of its constituent KMPs, this results in a new global prediction that was never explicitly demonstrated by the user during data collection.

The newly synthesized skill is intended to be used as a dynamic tool, which requires the creation of a corresponding schema. This schema must describe the new motion and specify how the objects are used and in what sequence.

4.4.1. Combination of TP-KMPs

Combining TP-KMPs involves joining frames (KMPs) from different TP-KMPs to form a new TP-KMP. Once the newly created TP-KMP is parameterized, the prediction process proceeds as described in Algorithm 2.

In principle, an unlimited number of KMPs can be combined to form a new TP-KMP. However, in practice, the KMPs must satisfy certain constraints to ensure meaningful global predictions. In the final step of Algorithm 2, the *local* predictions of the frames are fused into a *global* prediction by leveraging the covariance of the frame predictions. The covariance acts as an indicator of the influence each frame's prediction has on the global prediction. If multiple frames are confident (i.e., have low covariance) about their output for the same input values, the TP-KMP prediction will not match any individual frame's prediction exactly. Instead, it will produce an unintended blend of the conflicting predictions. Therefore, it is crucial that only *compatible* frames are combined.

Compatibility Constraints

To illustrate the importance of compatibility, consider the following example of a newly combined TP-KMP with two frames, where time is the input and 6D pose is the output. One KMP exhibits high covariance during the first half of the input domain, while the other exhibits low covariance. Both frames estimate low covariance during the second half. According to Algorithm 2, the local predictions are blended based on their covariances. As a result, in the first half, the global prediction resembles the KMP prediction with lower covariance. In the second half, since both KMPs estimate low covariance, the global prediction becomes a mixture of both frame predictions. Consequently, the TP-KMP's prediction for the second half does not correspond to either individual KMP's intended motion.

This example motivates the following formal definition: *compatible* KMPs are those that are not confident about their output over the same region of the input domain. This means that at any given input, only one KMP should be confident in its prediction. More formally, frames $1, 2, \ldots, P$ are compatible if the input domain can be partitioned into J non-overlapping continuous regions $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_J$, with $J = P$, and there exists a bijective mapping $\pi : \{1, \ldots, J\} \to \{1, \ldots, P\}$ assigning each region to its most confident frame, such that for each region $\mathcal{G}_j$ and all $t_g \in \mathcal{G}_j$:

$$\forall\, q \in \{1, \ldots, P\} \setminus \{\pi(j)\},\ \forall\, o \in \mathcal{O}: \quad \tilde{\sigma}^{(q)}_{o,t_g} - \tilde{\sigma}^{(\pi(j))}_{o,t_g} > \tau,$$

where $\tilde{\sigma}^{(p)}_{o,t_g}$ is the diagonal element of the covariance matrix $\tilde{\Sigma}^{(p)}$ of frame p at time t_g for output dimension $o \in \mathcal{O}$. The covariance difference must exceed a threshold τ, such that the global prediction will predominantly follow the KMP assigned to region $\mathcal{G}_j$. This ensures that

each frame is the most confident predictor within exactly one region of the input domain for all output dimensions.

Furthermore, the critical portion of each frame's motion must lie within the region $\mathcal{G}_j$ where it has the lowest covariance. When these conditions are satisfied, any number of KMPs can be combined into a new TP-KMP.

Challenges in Achieving Compatibility

While the compatibility constraint is theoretically straightforward, satisfying it in practice can be challenging. High uncertainty in a KMP typically results from strong variation in the demonstrations. However, it is not always feasible to introduce such variation, as it must span the entire output domain. In this work, the output is 6D poses, which means that the demonstrations need to vary both in position and orientation. Furthermore, the demonstrations must differ significantly to produce sufficiently high covariance in the appropriate regions.

To illustrate these challenges, consider the example of a *grasp and place* motion as introduced in subsection 4.1.2. This TP-KMP includes two frames (KMPs): the first frame represents the grasp motion, and the second frame represents the placing motion. Ideally, the first frame should be confident during the first half of the input domain across all output variables and uncertain during the second half, while the second frame should behave in the opposite manner.

Introducing variation in the x- and y-dimensions of the robot's frame is relatively straightforward, requiring only the placement of objects at different locations on a surface. The primary limitation is the robot's workspace dimensions. Varying the z-dimension already presents challenges because human reach is limited, especially when a table is placed in front of the robot.

Orientation is the most difficult aspect. In order to generate high variation, the objects must be rotated significantly. However, physical constraints limit how much an object can be tilted before it becomes unstable and falls over. This makes it impossible to demonstrate consistent pick motions for all desired orientations.

Note that the variation must be considered from the perspective of the frames. For example, if both objects are tilted toward each other by 45 degrees and both are approached from above during the grasp and place motions, then the relative deviation from each frame's perspective is 90 degrees. This perspective slightly reduces the required variation in each individual frame. However, depending on the task or desired motion, it can still be impossible to generate sufficient variation to meet the compatibility constraint.

Covariance Manipulation

To address the issue of limited variation in demonstrations due to physical constraints, the covariance can be artificially manipulated during combination. To ensure that skills and their frames remain compatible despite insufficient deviation, the covariance of a KMP can be adjusted to modify its influence on the global prediction. This adjustment involves both reducing covariance in the region where the frame should dominate and increasing it in all

other regions to minimize unintended influence. The entire covariance matrix is manipulated, with different scaling applied to different regions of the input domain.

Consider again the example with two KMPs. The input domain is divided into two regions $\mathcal{G}_1$ and $\mathcal{G}_2$ corresponding to the two frames. The covariance of the first frame is reduced in the first region $\mathcal{G}_1$ and increased in the second region $\mathcal{G}_2$, while the covariance of the second frame is reduced in the second region $\mathcal{G}_2$ and increased in the first region $\mathcal{G}_1$. When combining additional KMPs, the number of regions is matched to the number of frames P, with each region spanning approximately $1/P$ of the input domain. Each frame undergoes manipulation in all P regions, with covariance reduction applied to only one designated region $\check{\mathcal{G}}_j$ and covariance amplification applied to all others. It remains essential that the critical motion of each frame lies within its respective reduced covariance region.

To prevent abrupt transitions between consecutive frames and mitigate the risk of high-speed robot motions, buffer zones are introduced at the boundaries of each region $\mathcal{G}_j$. These buffer zones span five percent of each region's extent at both the beginning and end, resulting in manipulated regions $\check{\mathcal{G}}_j$ that are smaller than the original regions $\mathcal{G}_j$ and a total unmanipulated buffer of ten percent between consecutive manipulated regions. Within these buffer zones, no manipulation is applied, allowing for gradual transitions in the global prediction. However, no buffer zones are applied at the very beginning of the first manipulated region $\check{\mathcal{G}}_0$ or at the very end of the last manipulated region $\check{\mathcal{G}}_{P-1}$, ensuring full coverage of the trajectory endpoints.

The manipulation is applied using a cosine-based scaling function that generates smooth transitions within each manipulated region. For a manipulated region $\check{\mathcal{G}}_j$ consisting of N_g prediction points with a maximum scaling weight $\rho_{\max}$ (e.g., $\rho_{\max} = 30$), a one-dimensional profile is generated as

$$\gamma(i) = \frac{1}{2}\left(1 + \cos\left(\pi \frac{2i - N_g}{N_g}\right)\right), \quad i = 0, 1, \ldots, N_g - 1, \tag{4.2}$$

where i denotes the index of the prediction point within the manipulated region. This profile achieves its maximum value of 1 at the center and smoothly decays to 0 at the boundaries. A two-dimensional mask is constructed as the outer product $\mathbf{\Gamma} = \gamma \otimes \gamma$, resulting in a matrix of shape $N_g \times N_g$. The actual scaling factor matrix is computed as

$$\boldsymbol{\rho} = 1 + (\rho_{\max} - 1)\mathbf{\Gamma}, \tag{4.3}$$

where the addition of 1 ensures that the scaling ranges from 1 (no manipulation) at the boundaries to $\rho_{\max}$ at the center. Within the designated region $\check{\mathcal{G}}_j$ for frame p, the corresponding block of the covariance matrix $\mathbf{\Sigma}_n^{(p)}$ is divided element-wise by $\boldsymbol{\rho}$ to increase the frame's influence. In all other manipulated regions, the covariance blocks are multiplied element-wise by $\boldsymbol{\rho}$ to decrease the frame's influence. The manipulated covariance matrix $\tilde{\mathbf{\Sigma}}_n^{(p)}$ for frame p is thus constructed by applying the appropriate operation to each regional block.

For the first and last manipulated regions $\check{\mathcal{G}}_0$ and $\check{\mathcal{G}}_{P-1}$, the cosine-based profile is adjusted to maintain its maximum value throughout the endpoint regions, ensuring no decay at the trajectory's beginning or end. Figure 4.6 illustrates an example of a combined skill with two

KMPs, showing how the global prediction follows each frame in its respective region with smooth transitions facilitated by the buffer zones.

This approach highlights the necessity of knowing the order of the KMPs within the TP-KMP, which is incorporated in the skill's schema. For instance, consider two TP-KMPs, each composed of two frames (KMPs). If the first frame of both TP-KMPs has it's critical motion during the first region $\mathcal{G}_1$ of the input domain, these two frames cannot be meaningfully combined, even with covariance manipulation. While the first half of the combined TP-KMP may match one KMP's prediction, the second region $\mathcal{G}_2$ will not produce meaningful motion because neither frame has it's critical motion in that region to provide the necessary guidance.

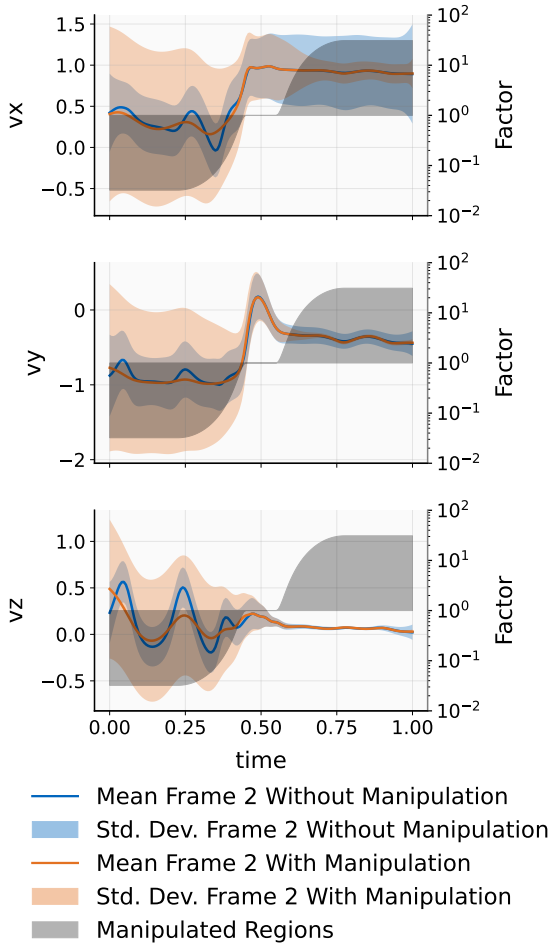

Figure 4.6.: Prediction of the second frame of a combined skill with two KMPs with covariance manipulation and the corresponding frame without covariance manipulation.
The factor for manipulation is shown as well.

Discussion

The combination of TP-KMPs proves to be a powerful method for generating new motions that have not been demonstrated during data collection. This enables the system to satisfy new user demands on the fly without requiring additional demonstrations. However, successful combination imposes certain constraints, particularly on the covariance distribution of the constituent frames. While methods such as covariance manipulation can help satisfy these constraints, they cannot resolve all compatibility issues.

Careful consideration is required to determine which skills can be combined and how they should be ordered. To address this challenge, the VLM's reasoning capabilities are exploited to autonomously select appropriate skills and determine the optimal combination strategy, as discussed in the following section.

4.4.2. Tool Call and Schema Generation

For a successful combination of skills such that the resulting skill has a meaningful prediction, the frames used need to be selected with care. Furthermore, it involves generating a corresponding schema that describes the skill's motion. Both tasks require a fair amount of reasoning to be executed correctly. Hence, the VLM is leveraged to provide the necessary reasoning capabilities.

To simplify the task for the VLM, the selection of frames (KMPs) and schema generation are split into two separate tasks. The choice of frames and their order in the new TP-KMP via calling the `CombineSkills` tool and filling its attributes is performed by the VLM that leads the conversation with the user. For schema generation, a new connection to the VLM is established specifically for that task. The same model is used for both operations, but the distinct connections maintain separate histories and system prompts. For the sake of readability, the connection used for user interaction is referred to as the *main* VLM, and the one for schema generation is named the *schema* VLM.

Frame Selection

To specify which frames to use from which TP-KMP, the main VLM calls the `CombineSkills` tool and sets the value for the `combination_dict` attribute. The full tool description can be found in section B.1. Through the `combination_dict`, the main VLM defines the TP-KMPs to use as the keys, with the values being the frame identifiers defined by the attribute names used in the TP-KMPs' schemas. With respect to the example of the *grasp and place* motion established in subsection 4.1.2, the first frame of the skill would be defined in the dictionary as `{"SkillGraspAndPlace": "self.object_to_grasp"}`. The VLM adds further frames in the same manner. The order of the skills and their frames corresponds to the sequence in which they are merged together.

The validation method of the `CombineSkills` tool verifies that the given skills exist. If this is not the case, the main VLM is informed and has two additional attempts to pass correct values.

Schema Generation

Generating a schema for the combined skill is performed by utilizing the schema VLM. This approach leverages the information contained in the schemas corresponding to the skills being combined. The schema VLM receives all schemas of the skills being combined as well as the frame selection information specified in the `combination_dict`.

According to the schema VLM's system prompt, the content for the new skill is generated. The content must be in YAML format, which is validated after generation. Moreover, it is ensured that the keywords and attributes have the correct type and format. The function used to check the schema's content can be found in Listing B.4.

5. Evaluation

The proposed framework is validated through a series of qualitative demonstrations on SARA. The goal of these tests is to verify that the system can successfully acquire, represent, and reproduce manipulation skills through demonstration, adapt these skills to changing object positions, and deploy them in response to user demands. Further, the ability to combine existing skills to create new motions is evaluated.

5.1. Experimental Setup

The framework is tested on SARA (see subsection 4.1.3), equipped with a pinch gripper and a static RGB-D camera positioned at the side of the robot's workspace. The camera captures the working table in front of SARA, enabling object detection and pose estimation through the perception pipeline described in subsection 4.1.1. Several objects from the YCB Dataset dataset are used, including the *CHEEZ-IT* cracker box, the *Master Chef* ground coffee can, and the *SPAM* canned meat. Example outputs of the pose estimator for the first two objects are shown in Figure 4.3.

Different foundation models were used throughout the development of the framework. These models include `Pixtral 12B` [101], `Qwen2.5-VL-72B-Instruct` [97], `Qwen2.5-32B` [76], and `DeepSeek-R1-32B` [102]. Notably, the latter two are not VLMs but LLMs. They were only used for the execution phase (see section 4.3), but did not receive image input. The VLM used for the experiments is the `Qwen2.5-VL-72B-Instruct`, as it empirically proved to be the best performing among the tested models.

5.2. Skill Teaching and Demonstration

Various skills are taught interactively, including *grasp and place*, *grasp and pour*, and *insert ring*. For each skill, multiple demonstrations are recorded to cover a range of motion directions, spatial, and orientational variations. The demonstrations are given directly on SARA following the procedure described in section 4.2. Each KMP is trained using $k_{gmm} = 26$ Gaussian components and $\lambda = 0.1$, with a *Matérn* kernel with $\nu = 5/2$ [24] and a length of $k_l = 0.1$ to predict $N = 150$ points.

5.2.1. Grasp and Place

The first skill taught is the *grasp and place* motion, which also serves as a running example in chapter 4. This skill involves grasping an object from above and placing it at a designated

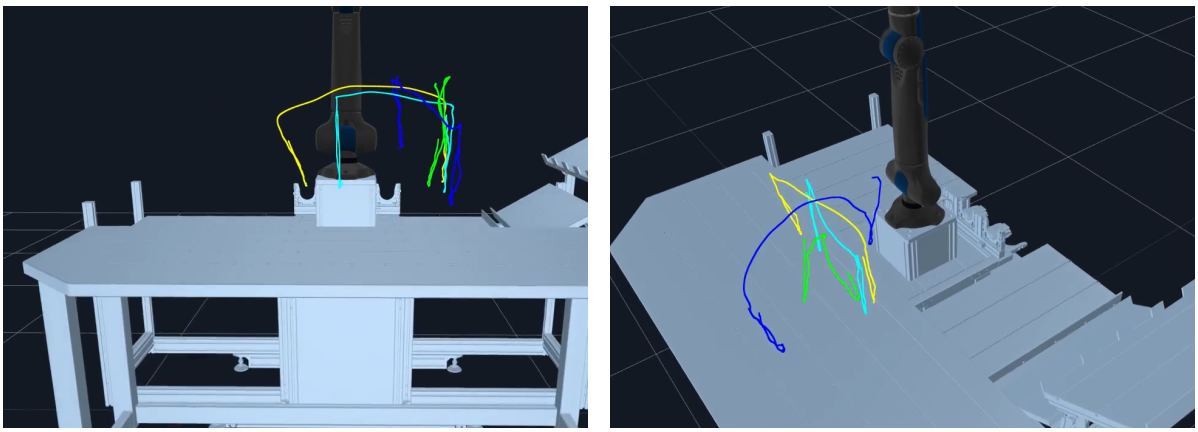

Figure 5.1.: Visualization of the demonstrations used to teach the *grasp and place* motion in the frontend. Each colored line represents the positional component of a single demonstration in the global (robot's base) frame.

location or on top of another object, hence consisting of two frames. Four demonstrations are recorded, each intentionally varied in position, orientation, and direction of motion to introduce diversity in the dataset. As already described, the incorporation of a significant amount of diversity in orientation proves difficult. The objects cannot be positioned with arbitrary orientation, since they fall over if tilted too much, making it impossible to , for example, grasp them from above. As can be observed from the yellow line in the left image of Figure 5.1, it was possible to introduce a tilt of approximately 25 degrees on one object.

After training the TP-KMP and generating a schema, which can be seen in Listing 4.1, the skill is added to the framework.

5.2.2. Grasp and Pour

The *grasp and pour* motion is similar to the *grasp and place* motion in the sense that it also consists of two frames and is also taught using the *CHEEZ-IT* cracker box and the *Master Chef* ground coffee can. The motion describes the process of grasping an object from the side and then moving above a second object. Once the first object is above the second one, the first object is tilted such that its content can be released. For teaching, the cracker box is grasped such that the gripper touches the box on the cover and back side. The ground coffee can serves as a container. Since the *CHEEZ-IT* box was not opened, no crackers are released. Images showcasing the motion can be seen in Figure 5.2.

5.2.3. Insert Ring

The final skill demonstrates a motion not involving objects from the YCB Dataset. This task is motivated by the intended use of the framework in industrial settings and involves objects and tasks provided by industry partners of the DLR. The *insert ring* motion places a bearing ring into a measurement unit that conducts quality checks. Images of the measurement unit

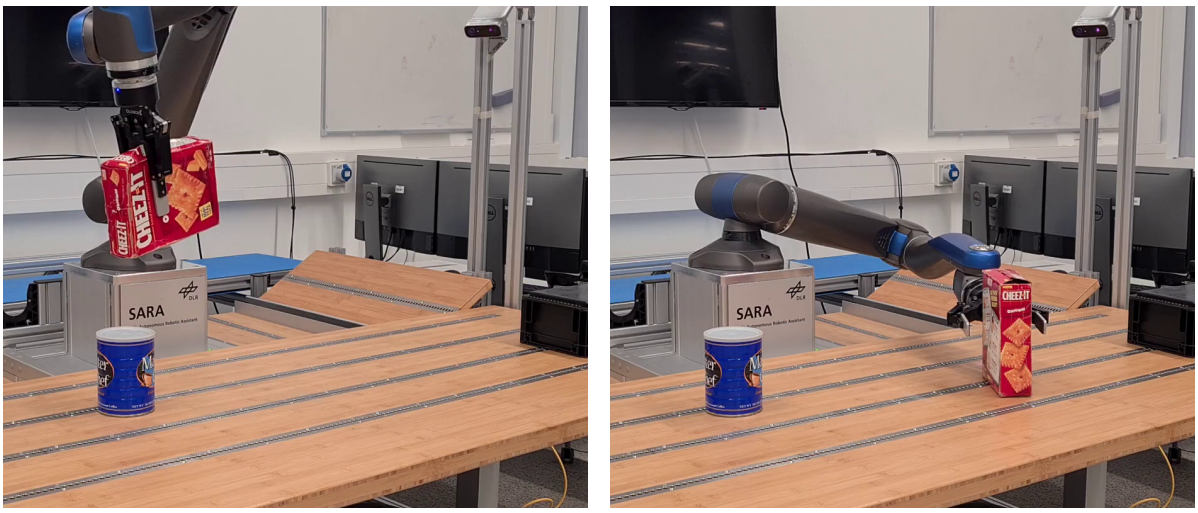

Figure 5.2.: Images of SARA executing the *grasp and pour* motion. The motion consists of grasping an object from the side (right) and then pouring its content into a second object (left).

are shown in Figure A.1. The measurement unit is secured in position using a click system integrated into the working table.

This task cannot be achieved with the *grasp and place* motion, as the insertion requires a particular and well-defined trajectory. The measurement pin, which conducts the measurement, is positioned at the bottom of the measurement surface. This pin can move vertically along a straight line and is attached to a spring pressing it to the outside of the measurement surface. Together with two additional contact points, these three points define a circle. The first additional contact point is positioned opposite the measurement pin, and the second is positioned approximately 15 degrees to the side of the first. The bearing ring must be placed such that it touches all three contact points simultaneously.

For successful insertion, the ring must first engage the measurement pin and press it upward until it can slip over the other two contact points. Once the ring passes these points, it can be released and is pulled into its final position by the spring force that holds the measurement pin in place.

This skill proves significantly more challenging to teach than the *grasp and place* motion due to the tight tolerances required for successful insertion. While the *grasp and place* motion allows for some spatial variation, the *insert ring* task demands precise alignment and a carefully controlled trajectory to avoid collisions with the measurement unit.

Six demonstrations are recorded to capture the precise insertion trajectory. During teaching, particular attention is paid to the sequential nature of the insertion process. The critical aspect is to first push the measurement pin upward with sufficient force and only then slip the ring over the remaining two contact points. This two-stage motion ensures that the ring does not collide with the other contact points during insertion. The demonstrations along with the resulting prediction for the position are shown in Figure 5.3.

Another challenge concerns the measurement unit itself. The measurement unit cannot be detected by the perception pipeline, as it is not part of the YCB Dataset. Fine-tuning a YOLO [88] model and training an AAE for the unit to integrate it into the perception pipeline proves difficult, as both require a 3D mesh of the desired object, which is not available. Consequently, the position and orientation of the measurement unit are specified manually. The skill assumes that the ring is already grasped by the gripper, eliminating the need to specify the pose of the ring. Furthermore, teaching the skill is not straightforward, making it difficult to apply different rotations during training.

In contrast to the other two skills, this skill's TP-KMP consists of only one KMP. This is due to the absence of a picking motion before insertion. To achieve compatibility with other skills for potential combination, the motion is divided into two parts. The first part exhibits no motion at all, in fact the robot's EE holds an arbitrary pose. The second half showcases the actual insertion. This approach enables significant variation between demonstrations, such that the KMP has high covariance in the first half and very low covariance in the second half.

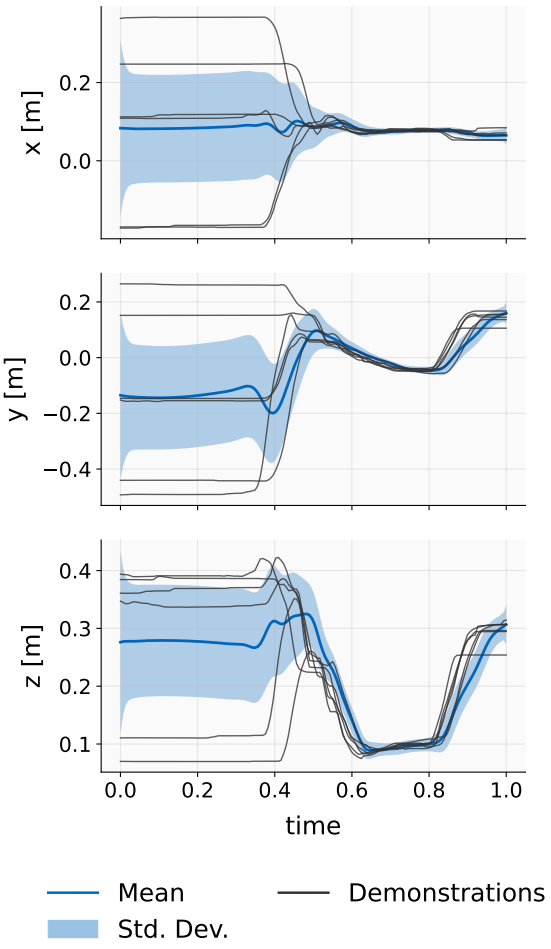

Figure 5.3.: Visualization of the six demonstrations and the resulting TP-KMP prediction for the *insert ring* motion. The plots show the position coordinates over the normalized time in the local frame of the KMP.

Although the TP-KMP possesses only one KMP, it is still possible to move the measurement unit and execute a successful insertion. However, the new position and orientation must be specified manually. The click system integrated into the working table proved very helpful in this regard, allowing for precise localization without using measurement tools. Leveraging this, it is confirmed that the skill can be executed at different positions with successful insertion.

5.3. Skill Execution

For the framework's verification, the `Qwen2.5-VL-72B-Instruct` [97] is used. Different prompts are employed to trigger the desired motions, varying in their level of abstrac-

tion and specificity. This section demonstrates how the VLM interprets natural language commands and selects appropriate skills based on the visual scene and user intent.

5.3.1. Prompt Variations and Skill Selection

Throughout the testing phase, the framework is interacted with using various natural language prompts. These prompts are categorized into three levels of abstraction: direct skill references, task-level commands, and high-level goal descriptions. Table 5.1 provides examples of each category and their corresponding outcomes.

Prompts that explicitly mention the skill name or closely describe the taught motion consistently trigger the correct skill. As expected, such direct references work reliably, as they match the skill descriptions provided to the VLM. More interesting are task-level commands that describe the desired outcome without explicitly naming the skill. These prompts generally work well, with the VLM successfully interpreting the user's intent and selecting the appropriate skill based on the visible objects. For instance, when given the prompt *"I want cracker on the can"* with the cracker box and both coffee and canned meat cans visible, the model correctly selects the *grasp and place* skill to pick up the cracker box and then randomly chooses one of the cans.

The framework is also tested with abstract, goal-oriented prompts such as the lowest two in Table 5.1. These very vague prompts result in the VLM requesting clarification or stating that no matching skill exists, as multiple interpretations are possible.

Table 5.1.: Examples of prompt variations and their desired outcomes. A darker background color represents a more abstract prompt, categorizing them into Direct Skill References, Task-Level Commands, and High-Level Goal Descriptions respectively. The object names refer to the YCB Dataset objects *CHEEZ-IT* cracker box, *Master Chef* ground coffee, and *SPAM* canned meat.

Prompt	**Detected Objects**	**Desired Outcome**
"Grasp and place the box on the can"	cracker box, ground coffee	*grasp and place* motion
"Pour the crackers into the can"	cracker box, ground coffee, canned meat	*grasp and pour* motion
"I want cracker on the coffee"	cracker box, ground coffee	*grasp and place* motion
"I want some cracker in the can"	cracker box, ground coffee, canned meat	*grasp and pour* motion
"Prepare breakfast"	cracker box, ground coffee, canned meat	No matching skill, clarification requested
"Organize the workspace"	cracker box, ground coffee, canned meat	No matching skill, clarification requested

5.3.2. Natural Language Interface Evaluation

Robustness to Language Variations

During testing, the VLM demonstrates robustness to variations in phrasing. Synonymous commands such as *"Put the box on the can"* and *"Place the box on top of the can"* both trigger the same skill. Minor grammatical variations or casual language do not impede the model's ability to interpret the command correctly. Even misspellings or filler words used during natural speech do not affect the outcome.

Ambiguity Handling

In cases of ambiguity regarding the skill, the VLM generally responds with clarification requests rather than making arbitrary decisions. For example, when given an underspecified prompt with multiple possible interpretations, the model asks the user to specify which action is desired. This behavior is important for ensuring that the robot performs the intended task. If the ambiguity concerns which object to use, on the other hand, the model tends to choose one that matches the description without requesting clarification.

Visual Context Integration

The natural language interface proves flexible when objects are repositioned within the workspace. The same prompt can be used regardless of where the objects are located, as the VLM interprets the command based on the perceived object configuration. For instance, the prompt *"Stack the box on the can"* works correctly whether the objects are positioned on the left, right, or center of the workspace, as long as they are visible to the camera.

5.3.3. Execution Results

Once the VLM has chosen a skill and parameterized it with the detected object poses, the skill is executed. For all three skills presented, execution succeeded with different object positions. The tested object poses varied significantly from those used during training, demonstrating the generalization capability of the TP-KMP. Moreover, for the *grasp and place* and the *grasp and pour* motions, different object orientations were tested, and the framework proved to execute the skills reliably. Success is qualitatively assessed by observing whether the motion is executed as expected and achieves the intended outcome.

However, it was observed that the EE does not always reach the same position relative to the object. This happens especially often when objects are placed at the edges of the working table. Two factors can cause this effect. First, the accuracy of the perception pipeline's pose estimation is reduced at the edges of the camera's field of view or when objects appear small in the image. Second, the object might be at the limit of SARA's workspace. The skill's trajectory prediction does not explicitly account for the robot's reachability constraints. If the predicted trajectory extends beyond the reachable workspace, SARA attempts to follow it but

cannot fully execute the motion. This can result in incomplete or inaccurate execution of the predicted skill trajectory.

5.3.4. Limitations and Failure Cases

While the VLM-based interface generally performs well, some limitations are observed. Very abstract or ambiguous prompts sometimes confuse the model, particularly when multiple skills could plausibly satisfy the command. Additionally, if the prompt references objects that are not visible in the camera feed, the VLM cannot fulfill the request, and sometimes chooses a random object.

The model's performance also depends on the quality of the skill descriptions contained in the provided schemas. Clearer, more detailed skill descriptions generally lead to better skill selection accuracy. This suggests that careful prompt engineering at the system level for schema generation is important for robust performance.

The framework's performance depends on the accuracy of the perception pipeline. If object detection or pose estimation fails, the VLM cannot correctly interpret the scene, leading to either incorrect skill selection or requests for clarification. Furthermore, the pose estimation's precision is reduced when the objects are placed at the edge of the camera's vision. This can lead to inexact motion, even with correct skill selection. Inaccuracy of object pose estimation can and most likely will happen if objects are partly occluded. Since the perception pipeline is not part of the framework, it technically does not matter for performance. Still it is noteworthy that the object poses provided to the framework require correctness.

5.4. Combining Skills

To evaluate the framework's ability to combine skills, a new motion is generated by merging two existing skills. In an industrial setup example, the *grasp and place* skill is combined with the *insert ring* skill to produce a motion that involves grasping an object and subsequently inserting it into the ring measurement unit. Logically, the grasped object should be a bearing ring that matches the ring measurement unit.

5.4.1. VLM Decision-Making for Skill Composition

The chain of thought that leads to the conclusion that a combination of skills is required incorporates multiple steps that the VLM must execute. First, it must recognize that none of the existing skills fully satisfies the user's request. Next, the model needs to understand that the user's intended motion can be achieved by combining two existing skills. Finally, the appropriate and compatible frames of the involved skills need to be selected and ordered correctly. This is no trivial task and requires substantial reasoning capabilities from the VLM.

Hence, it is not surprising that the model struggles to arrive at the correct solution with more abstract user prompts. In Table 5.2, three example prompts of varying complexity are presented. The first prompt is categorized as a direct skill reference, explicitly telling the model which skills and objects to use. This prompt consistently results in successful skill

combination, as the model receives clear instructions on which skills to merge and in what order, explicitly referencing the names of the skill schemas.

In contrast, the second prompt represents a task-level command that does not specifically state the needed skills and frames but hints that a combination of skills is required. The VLM receives the information that the task requires picking the ring up and then conducting a quality check, without explicitly naming the involved skills. The model needs to understand that none of the existing skills satisfies this motion. However, the VLM often fails to reach that conclusion and instead decides to load the *grasp and place* motion.

The third prompt is also categorized as a task-level command but provides less information on how to arrive at the correct solution. This description requires that the VLM infers both the need for combination and which specific skills to compose. This abstract formulation proves significantly more challenging. The model must understand that measuring the ring requires first grasping it and then inserting it into the measurement unit, and that this sequence cannot be accomplished by any single existing skill. Consequently, it is not surprising that the model nearly always fails to arrive at the correct solution.

The success rate for skill combination varies considerably based on prompt specificity. Direct references to skills and explicit combination requests achieve near-perfect accuracy, while task-level prompts result in incorrect solutions most of the time. In cases where the model fails to identify the correct combination, it typically selects a single skill that partially addresses the prompt but does not fully satisfy the intended outcome.

The performance is highly influenced by the VLM's system prompt and the information provided in the skill schemas. With improved prompt engineering, the framework is expected to perform better.

Table 5.2.: Examples of prompts tested to trigger skill combinations. Darker shading indicates more abstract combination requests.

Prompt	**Detected Objects**	**Desired Outcome**
"Combine the SkillStackObjects and SkillInsertRing, using the 'object to grasp' and the 'measurement unit'."	bearing ring, measurement unit	Combination of motions *grasp and place* and *insert ring*
"Pick up the ring and do a quality check. If no matching skill exists combine fitting ones."	bearing ring, measurement unit	Combination of motions *grasp and place* and *insert ring*
"Measure the ring"	bearing ring, measurement unit	Combination of motions *grasp and place* and *insert ring*

5.4.2. Skill Selection and Sequencing

In this example, the correct combination is to take the frame containing the grasp from the *grasp and place* motion together with the only frame of the *insert ring* motion, in this order.

When the VLM decides to combine skills to satisfy the prompt, it selects the correct skills and frames. Additionally, the sequence of the frames in the new TP-KMP is correct. Even though the skill representing the *grasp and pour* motion is published to the VLM, it never chooses to combine that skill with the *insert ring* motion. An image of the conversation leading to a successful combination and the resulting prediction is shown in Figure 5.4.

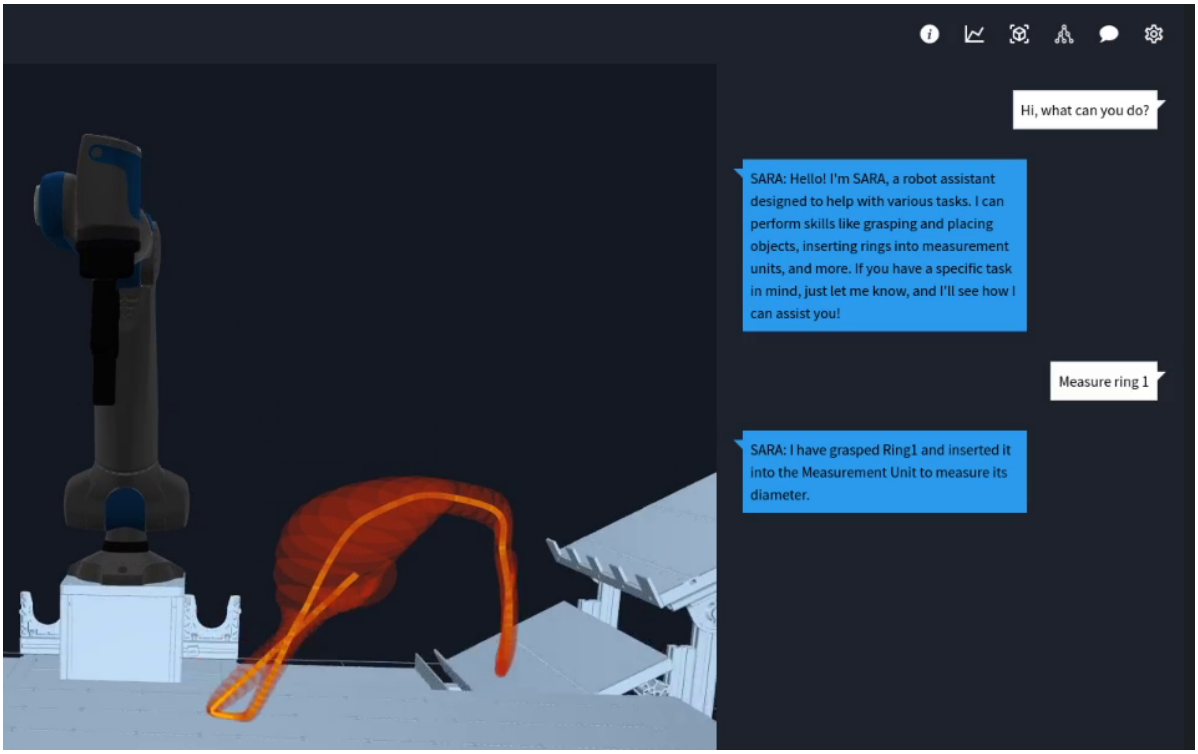

Figure 5.4.: Screenshot showing the frontend with the visualization of the combined ring insertion TP-KMP's prediction. The orange line indicates the position part of the trajectory, while the red circles show the covariance of these values. The object positions were defined manually, since they cannot be detected by the perception pipeline.

5.4.3. Execution Performance

Combining the two skills containing the *grasp and place* and the *insert ring* motions yields a composite skill that picks up the ring and subsequently inserts it into the measurement unit. After creation, the skill is made available to the VLM such that it can be selected for execution. In Figure 5.4, the prediction of the combined skill in the frontend is visualized. The execution of the skill produces the desired robot motion, successfully picking up the ring and inserting it into the measurement unit. This behavior works reliably across different object positions. Consequently, there is no need to utilize the covariance manipulation technique introduced in subsubsection 4.4.1 for this example.

In a second example, the skills *grasp and pour* and *grasp and place* are merged to create a motion that grasps an object from the side and then places it with a 90-degree rotation. This combined skill fails to execute the desired motion without covariance manipulation. To analyze the orientation behavior, the 3D rotation matrices are represented as 6D vectors using the Gram-Schmidt orthogonalization process [103, 104], which allows for continuous and differentiable orientation representation. In Figure 5.5, the local predictions of the original skill frames used for combination and the predictions of the combined skill with and without covariance manipulation in the global (robot base) frame are shown. The combined skill should first follow the first frame of the *grasp and pour* motion to pick up the object from the side and then follow the second frame of the *grasp and place* motion to place the object with the gripper facing downward, thus applying a rotation.

Analyzing the plots in Figure 5.5, it is evident that this is not the case for the combined skill without covariance manipulation. Examining the *vx* component as an example, this deviation is caused by a considerably large covariance in the second half of the second frame of the *grasp and place* motion, which corresponds to the placement phase. In contrast, the covariance of the first frame of the *grasp and pour* motion in that region is small, resulting in a higher influence on the global prediction.

The shape of these covariances reflects the task-specific constraints during demonstration. For the *grasp and pour* motion, while positional variation was present across demonstrations, the orientation remained consistent to prevent tilting of objects during pouring. Consequently, the learned model exhibits low orientation uncertainty despite high spatial variability. When this frame is combined with the *grasp and place* motion, which has higher orientation uncertainty in the placement phase, the product of Gaussians is dominated by the low-uncertainty pouring orientation. This results in the global prediction inappropriately following the orientation trajectory of the *grasp and pour* motion throughout the entire combined skill. Similar behavior can be observed for the *uy* component.

The manipulated version of the combined skill maintains the intended path and the skill can be executed successfully. It is noteworthy that due to the combination, the orientation values change much more rapidly over a short time span than in the two original skills. This results in relatively fast movements of the robot's EE.

5.4.4. Generalization to Novel Objects

During execution, the skill describing the *grasp and place* motion proves to work reliably with different object positions. Interestingly, the skill also works with objects that are not used in the learning phase. Hence, the robot is able to use the same skill to pick up the canned meat *SPAM* and place it on the ground coffee can. It is also possible to change the object that functions as the base. Notably, the trajectory might not fit the new object perfectly anymore, meaning that when the *SPAM* is placed on the *Master Chef* can, there is a gap between the two objects when the gripper releases the canned meat.

These effects are logically explained by the fact that the perception pipeline returns the center of the objects as their coordinate frame. These frames are used to parameterize the TP-KMP's KMPs. The KMPs predictions are transformed into the global (robot's base) frame

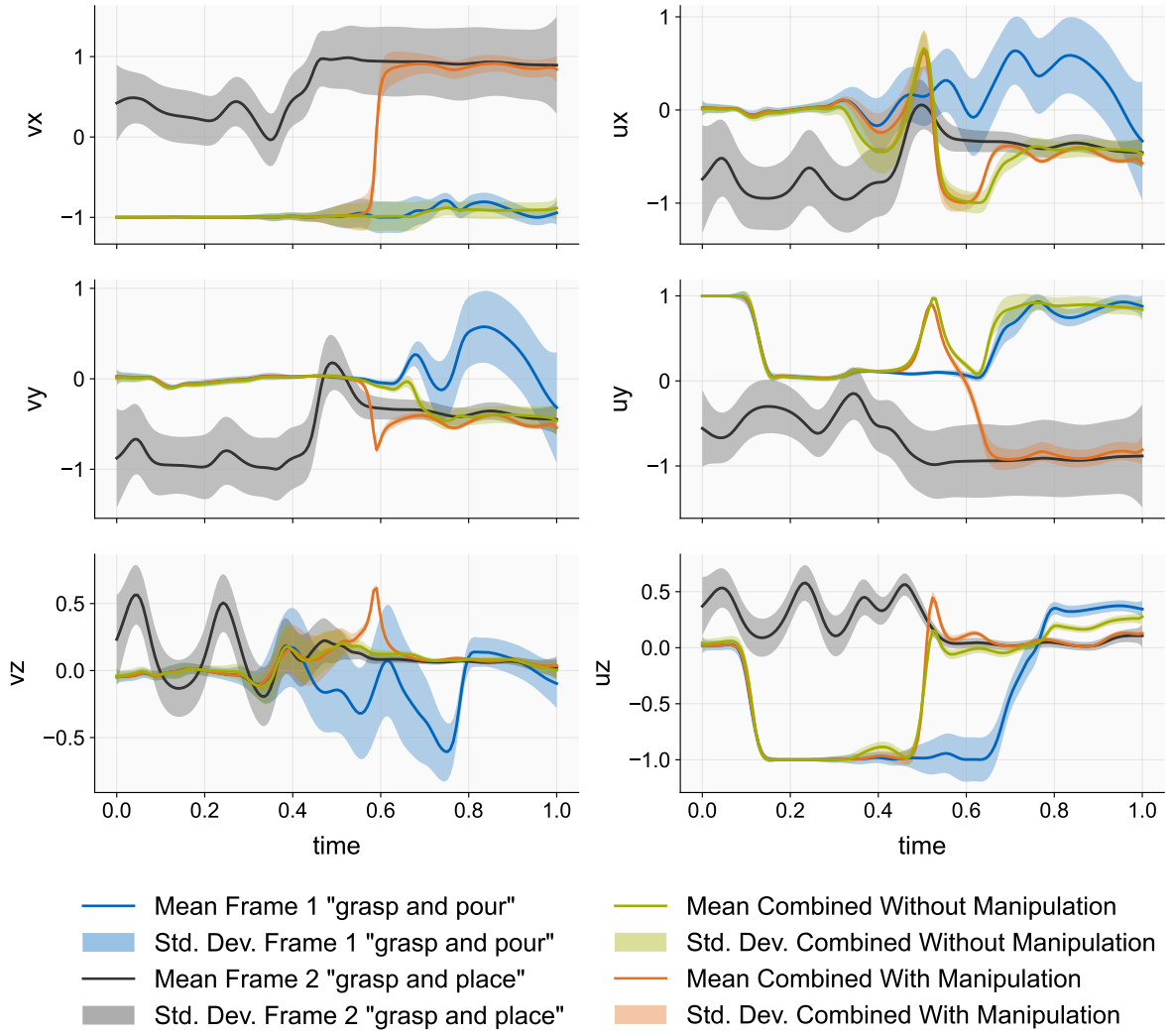

Figure 5.5.: Plot showing the orientation of the robot trajectories represented as 6D vectors calculated using the Gram-Schmidt process. The plot shows the global prediction of the combined skill with and without covariance manipulation as well as the local predictions of the frames used for combination in the global frame.

using these frames, as described in section 3.2. Thus, the learned trajectory of each KMP is relative to the object's size. The placing motion was trained with the cracker box and the ground coffee can, leading to a KMP's prediction that will stop the robot's EE at a certain distance above the *Master Chef*'s origin to open the gripper. If a significantly smaller object like the canned meat is used with the same prediction, the distance of the gripper to the center of the coffee can will be similar, but the canned meat does not fill this distance as the cracker box did.

The same principle can be applied to the grasping motion. The EE of the robot approaches the *CHEEZ-IT* cracker box at a certain distance to its origin from above. This distance will be the same for all other objects used with this skill. For the smaller *SPAM* can, this did not pose a problem, but imagine a bigger object. The distance from the large object's origin to the outline of its shape is greater than the learned distance between its center and the EE. Consequently, the robot will collide with the object. For soft objects, this might be irrelevant, but fragile objects will be damaged. Notably, the robot will not push against the object with infinite force. The force is defined by the distance between the object's outline and the learned destination of the EE and the hyperparameters of the impedance controller. Assuming that the robot approaches the object slowly, the mass and damping factor can be omitted. With the chosen stiffness of $K_f = 750$ N/m representing the virtual spring, the maximum exerted force $F_{\max}$ on the object can be calculated by

$$F_{\max} = K_f \|d\|, \tag{5.1}$$

where $\|d\|$ denotes the displacement magnitude between the desired and actual EE positions in meters.

The distance $\|d\|$ is not easily calculated, because the predicted trajectory changes with object positions and the way the motion was taught also matters. If demonstrations show that the grasp motion is performed by touching the object at the very edge, the distance is different from a motion that was taught by fully placing the object inside the gripper. Nevertheless, it can be approximated by the difference in size between the objects, assuming that the relative position to the object's origin of the EE will not change between different attempts. The only needed measure is the distance from the object origins to their outline where they are grasped, denoted as d_o. For two objects $o = 1$ and $o = 2$, this results in

$$\|d\| \approx \frac{|\|d_1\| - \|d_2\||}{2}. \tag{5.2}$$

This approximation of the distance can also be used in other contexts, for example, to calculate the gap between the *SPAM* can and the ground coffee can. Having this approximation in mind, it is possible to intuitively decide if the taught skill will work with another object by comparing the sizes and shapes of the objects. Obviously, this approximation is feasible only for this skill, but it provides good intuition on how object sizes matter for the learning and execution phases and how they affect other skills as well.

As another example, the *grasp and pour* motion fails when applied to the *SPAM* canned meat. The motion is demonstrated using the *CHEEZ-IT* cracker box, grasping it from the side

(see Figure 5.2). The KMP learns to lower the robot's EE to a certain height with respect to the object's origin before rotating it from an upside-down orientation to grasp from the side. When this motion is applied to the significantly smaller *SPAM* can, the prediction causes SARA's gripper to collide with the table before the rotation occurs.

5.5. Summary

The qualitative evaluation conducted in this chapter demonstrates several key capabilities and limitations of the proposed framework. The system successfully learns, executes, and combines manipulation skills while providing a natural language interface for deployment.

The framework demonstrates reliable skill execution across all three implemented skills. The *grasp and place* motion generalizes well to different object positions and moderate orientation changes, while the more complex *insert ring* skill maintains precision even with position variations. The *grasp and pour* motion successfully executes with consistent objects but shows sensitivity to significant size variations. This generalization capability confirms the effectiveness of the task-parameterized approach in adapting to new object configurations.

The natural language interface proves robust in handling various command formulations. Direct skill references and task-level commands consistently trigger appropriate skill selection, with the VLM correctly interpreting user intent even with grammatical variations or minor misspellings. The system appropriately handles ambiguity by requesting clarification when commands are too abstract or when multiple interpretations are possible, contributing to predictable and safe operation.

Skill combination capabilities show promise but with clear limitations. The framework successfully combines the *grasp and place* and *insert ring* skills when given explicit instructions, maintaining smooth transitions between component motions. However, the VLM's ability to autonomously recognize combination needs and select appropriate skills degrades significantly with more abstract commands, indicating current limitations in complex reasoning capabilities.

Several technical limitations emerge from the evaluation. The framework's performance depends heavily on accurate object pose estimation, with reduced reliability at workspace boundaries or when objects are partially occluded. Object size sensitivity affects skill transfer, particularly for motions with specific geometric requirements like the *grasp and pour* motion. These limitations, while not fundamental to the framework's design, represent practical constraints for deployment that inform future development priorities.

6. Discussion

This chapter interprets the results presented in chapter 5 and discusses implications, limitations, and the broader context of the proposed framework. The evaluation shows that integrating TP-KMP with pretrained VLMs enables intuitive skill teaching, natural language-based execution, and on-the-fly skill composition, but also exposes several design and practical constraints.

6.1. Architectural Design and System Assumptions

The modular architecture separates perception, reasoning, and execution into distinct components. This contrasts with end-to-end approaches such as OpenVLA [19], which integrate visual processing, language understanding, and action prediction into a single model. In industrial and service robotics, the chosen architecture trades some generality for data efficiency, requiring only 3–6 demonstrations per skill instead of hundreds or thousands for fine-tuning. The mathematical foundations of TP-KMP provide inherent uncertainty quantification and smooth trajectory generation, whereas end-to-end policies must implicitly acquire these properties through extensive training.

A key design goal was compatibility with different foundation models and TPIL methods. Multiple VLMs and LLMs were tested, including Pixtral 12B [101], Qwen2.5-VL-72B-Instruct [97], Qwen2.5-32B [76], and DeepSeek-R1-32B [102]. The modular structure enables straightforward model substitution and adaptation to emerging foundation models without architectural changes. This contrasts with approaches such as VocalSandbox [78], where the foundation model is prompted via the API of a specific provider, limiting flexibility.

While TP-KMP is used as the TPIL method, the framework also accommodates alternatives such as TP-GMM [28, 42], provided the TPIL implementation satisfies the framework interface. The skill combination mechanism in section 4.4 relies on covariance-based fusion and thus applies to probabilistic TPIL models that represent uncertainty via covariance matrices. Implementation details, such as the covariance manipulation strategy (subsubsection 4.4.1), may require adaptation for different uncertainty representations.

6.1.1. Perception and Pose Estimation

Although the framework assumes known object poses, an example perception pipeline was implemented for demonstration. The pipeline described in subsection 4.1.1 uses `YOLOv7` [88] for object detection and an AAE-based [92] dense pose estimator for 6D pose inference. This introduces additional complexity compared to potential VLM-based pose estimation, but offers greater accuracy and reliability. Recent studies indicate that current VLMs, while

exhibiting some spatial reasoning capabilities, remain primarily focused on semantic understanding and struggle with complex spatial relations and precise pixel-level affordances required for manipulation [11, 12]. Consequently, they are not yet reliable replacements for dedicated 6D pose estimators in tasks that demand high geometric precision, such as the ring insertion evaluated in chapter 5.

The fixed camera configuration simplifies calibration, as the extrinsic parameters between camera and robot base remain constant. However, this limits visibility and can cause occlusions when multiple objects are present or when the robot obstructs the view. Despite this, the configuration proved sufficient for the evaluated tasks and represents a pragmatic trade-off between system complexity and deployment effort.

The reliance on accurate object pose estimation constrains applicability to settings where the perception pipeline is reliable. The implemented pipeline is trained on the YCB Dataset, restricting object diversity and requiring either dataset extension or alternative perception approaches for broader deployment.

6.1.2. Use of VLMs as High-level Reasoners

The framework uses pretrained VLMs exclusively for high-level reasoning tasks such as skill selection, object identification, and schema generation, rather than for trajectory generation or low-level control. As discussed in section 2.2, current LLMs and VLMs lack sufficiently robust spatial reasoning for accurate motion prediction without extensive contextual information or fine-tuning [105, 106].

The tool-calling paradigm provides a structured interface between VLM reasoning and execution components. Dynamic tools (subsubsection 4.1.2) proved essential for reliable skill parameterization. Early attempts with a generic `LoadSkill` tool required the VLM to infer motion semantics from skill names alone, leading to frequent errors in object selection and ordering. Embedding skill-specific information directly into tool schemas reduces the cognitive load on the VLM and improves robustness.

The schema validation mechanism currently checks only structural correctness (YAML and JSON format, data types, required fields) and does not verify semantic validity. The system cannot automatically assess whether a schema accurately reflects the demonstrated motion or object order, leading to occasional user corrections, especially for complex skills. A multi-VLM architecture, with one model generating schemas and another verifying their semantics [107, 108], could mitigate this at the cost of increased complexity and computation.

6.1.3. Static Scene Assumption

A notable constraint is the static scene assumption. Once an image is captured and object poses are estimated, the TP-KMP predicts a trajectory based on these poses. Objects cannot be moved or rearranged during execution, as the trajectory is not updated. For many industrial and household manipulation tasks this is acceptable, but scenarios with dynamic objects or close human-robot collaboration would require real-time pose tracking and replanning.

6.2. Teaching Phase

Compared to ICL approaches [21], the framework stores skills in the model rather than in the system prompt, avoiding context window limitations and enabling long-term skill library accumulation. At the same time, demonstration collection exposed practical constraints of the current approach.

As discussed in section 4.4, sufficient variation in demonstrations is crucial for obtaining informative covariance structures in the learned TP-KMP, especially for skills intended for later combination. Variation in position is relatively easy to obtain within the workspace, but orientation variation is physically constrained. Objects cannot be arbitrarily tilted without losing stability. For the *grasp and place* motion, only about 25 degrees of tilt could be demonstrated reliably.

Insufficient orientation variation yields KMPs with low covariance in orientation dimensions across the input domain, which can cause undesired fusion behavior when frames are combined. The covariance manipulation method in subsubsection 4.4.1 alleviates this but cannot make arbitrary KMPs compatible. Demonstration strategies that explicitly teach atomic motions with compatibility padding may address this and are discussed in chapter 8.

6.2.1. User Interaction During Teaching

The current demonstration process uses physical buttons on the DLR robot's (SARA) last link to control recording (start, pause, resume, stop, cancel) and gripper operation. This interface is intuitive and effective but platform-specific. It is not set that any collaborative robot has similarly accessible programmable buttons or displays.

To increase platform independence and autonomy, the same functionality could be exposed through VLM-based static tools. Voice commands such as "start recording" or "stop demonstration" could be interpreted by the VLM and translated into control signals, removing the need for dedicated hardware while preserving hands-free kinesthetic teaching. This would, however, depend on robust speech recognition and may introduce latency in the control loop.

6.3. Execution Phase

The evaluation indicates that the VLM-based natural language interface interprets user commands across different abstraction levels. Direct skill references and task-level commands typically trigger correct skill selection, while very abstract prompts lead to clarification requests. This behavior aligns with the goal of an intuitive yet safe and predictable interface.

An observed asymmetry arises in handling ambiguity. When the skill choice is unclear, the model usually asks for clarification. When the object choice is ambiguous (e.g., multiple cans but the prompt mentions "the can"), it often selects one arbitrarily. For safety-critical applications, the system should request clarification in both cases. This could be implemented via validation logic that detects multiple matching objects and enforces a clarification step before execution.

6.3.1. User Feedback and Transparency

The frontend visualization (Figure 5.4) was valuable during development and evaluation, making VLM decisions and planned trajectories transparent before execution. However, requiring users to monitor a screen might conflict with the aim of intuitive interaction, requiring users to divide their attention. The framework can be operated purely by voice, but then users lose the ability to preview motions.

Alternative feedback mechanisms could reconcile this trade-off. Augmented reality projection could overlay predicted trajectories directly in the workspace, removing the need for a separate display. Alternatively, the VLM could verbally summarize the intended motion before execution via the `TellUserMessage` tool, allowing users to confirm or cancel the action.

6.4. Skill Combination: Capabilities and Limitations

The skill combination mechanism is a central contribution, allowing new behaviors without extra demonstrations. The evaluation demonstrated two successful combinations. Combining *grasp and place* with *insert ring* produced a composite motion that picks up a bearing ring and inserts it into the measurement unit. Combining *grasp and pour* with *grasp and place* produced a motion that picks up objects from the side, rotates them by 90 degrees, and places them. The latter required covariance manipulation. At the same time, the results revealed limitations in the VLM's ability to autonomously decide when combination is necessary and to select appropriate skills.

As shown in subsection 5.4.1, the success rate of skill combination strongly depends on prompt specificity. Direct references to skills and explicit combination requests work reliably, whereas task-level prompts often lead to incorrect single-skill selections. This suggests that current VLMs struggle with the multi-step inference needed to recognize that (1) no single skill satisfies the request, (2) the task can be decomposed into sub-tasks covered by existing skills, and (3) specific frames should be combined in a particular order.

The system prompt and schema content significantly influence this behavior. More explicit prompting, for example by providing combination examples or decision rules for when to combine versus when to select a single skill, may improve autonomous composition. However, there is a trade-off: very detailed prompts may boost performance on known task patterns but reduce generality for novel scenarios.

6.4.1. Covariance-based Fusion and Compatibility Constraints

When compatibility constraints are satisfied, covariance-based fusion produces smooth, continuous trajectories. The industrial ring insertion example executed reliably without covariance manipulation because the component skills naturally had compatible covariance structures. In contrast, the *grasp and pour* plus *grasp and place* combination required explicit manipulation to achieve the desired behavior.

The need for covariance manipulation reflects the physical constraints of demonstration. If orientation variation is limited by object stability, the resulting KMPs may not satisfy

compatibility conditions. While manipulation provides a practical workaround, it introduces hyperparameters (region boundaries, scaling factors) and does not guarantee compatibility for arbitrary KMPs.

A deeper limitation stems from how KMPs distribute motion across the input domain. Each frame executes its critical motion in a specific region of the normalized domain. In a two-frame skill, critical motions typically occupy the first and second halves. In a three-frame skill, they are spread across beginning, middle, and end regions. Combining a middle-region KMP from a three-frame skill with a KMP from a two-frame skill can therefore be fundamentally incompatible due to overlapping critical regions. This structural limitation cannot be resolved by covariance manipulation alone.

6.5. Generalization and Transfer to Novel Objects

The framework unexpectedly generalized to objects not used during demonstration. The *grasp and place* skill, taught with the *CHEEZ-IT* cracker box and *Master Chef* coffee can, successfully operated on the *SPAM* can in both grasping and placing roles. This emerges naturally from the frame-relative encoding of TP-KMP, where trajectories are learned with respect to object coordinate frames instead of absolute positions.

Generalization is, however, limited by object size similarity. As analyzed in subsection 5.4.4, the distance between the robot EE and the object origin is learned from demonstrations and remains approximately constant when transferring to new objects. For significantly larger objects this may cause collisions and for smaller ones it can lead to gaps or incomplete task execution.

This has direct implications for deployment. Users must consider object size when reusing skills, and the current framework does not automatically detect incompatible sizes or adapt trajectories. Future extensions could incorporate object dimensions, either via explicit input or perception-based estimation, to scale trajectories or warn users about likely incompatibilities.

7. Conclusion

The deployment of robots in dynamic, human-centric environments has historically required extensive programming expertise and explicit specification of motion primitives. Traditional approaches impose significant barriers to adoption, particularly in settings where tasks change frequently or where non-expert operators must interact with robotic systems. While recent advances in foundation models demonstrate impressive capabilities for natural language understanding and zero-shot reasoning, their application to robotics remains constrained by the tension between data requirements and practical deployment constraints.

This thesis presents a framework that integrates TP-KMPs with a pretrained VLM to enable natural language-based robot control with minimal demonstration requirements. The modular architecture separates perception, reasoning, and execution into distinct components. A dedicated perception pipeline provides the geometric precision required for manipulation, the VLM interprets natural language commands and selects appropriate skills without fine-tuning, and TP-KMPs generate smooth trajectories conditioned on task parameters extracted from the environment.

The framework is validated on the DLR robot SARA. Manipulation skills are acquired through kinesthetic demonstration, requiring only 3–6 examples per skill to achieve robust execution. Dynamic tool generation via schemas enables seamless integration between the natural language interface and the skill library, allowing skills to be executed through commands at varying levels of abstraction. Learned skills generalize to novel object configurations, successfully executing tasks with objects positioned at locations not represented in the demonstration set. A probabilistic skill combination mechanism further enables the synthesis of novel behaviors from existing skills without additional demonstrations by fusing KMPs from different TP-KMPs through products of Gaussians.

These results indicate that data-efficient TPIL, combined with an intuitive natural language interface, can substantially lower the barrier to entry for both industrial and service robotics. Manufacturing environments with frequent task variation can benefit from quickly teachable skills that do not require extensive programming expertise, while domestic and healthcare settings can exploit the same mechanisms to acquire task-specific behaviors that remain accessible to non-expert users. The framework's ability to expand the skill library over time supports progressive capability development, where accumulated skills serve as building blocks for increasingly complex behaviors.

At the same time, the work highlights several limitations that constrain the current framework, including static scene assumptions, reliance on accurate object pose estimation with a limited set of objects, and restricted VLM capabilities for fully autonomous skill decomposition and combination. These observations motivate the directions for future work outlined in chapter 8.

Overall, the thesis demonstrates the viability of hybrid approaches that integrate classical robotics methods with modern foundation models. Rather than pursuing end-to-end learning as a universal solution, the proposed architecture employs foundation models for tasks where their strengths are most applicable, specifically high-level reasoning and natural language understanding. Complementary components preserve geometric awareness, uncertainty quantification, and data efficiency. This design philosophy offers a practical path toward robotic systems that combine accessible interaction with the precision and safety required for real-world deployment.

8. Future Work

8.1. Perception Capabilities

Although the perception pipeline is not formally part of the framework, several enhancements would increase its practical applicability. The current constraint to YCB Dataset objects limits object diversity, motivating methods for rapid integration of novel objects, for example through few-shot learning. Hybrid perception strategies combining traditional computer vision with VLM-based object identification could balance the precision needed for manipulation with the flexibility to recognize arbitrary objects. Handling occluded or partially visible objects, potentially via temporal information or multi-view reconstruction, and exploring mobile or multi-camera setups would improve robustness but introduce additional calibration and coordination challenges. As VLMs improve in spatial reasoning, it will be important to periodically reevaluate their suitability for pose estimation tasks and their potential to simplify the perception pipeline.

8.2. Real-Time Adaptation and Replanning

The static scene assumption represents a fundamental limitation for tasks involving dynamic objects or human-robot collaboration. Future work should explore continuous object pose tracking during execution and trajectory replanning based on updated task parameters. Human-robot collaboration scenarios would particularly benefit from such capabilities, as humans frequently could reposition objects or intervene during execution. Integrating reactive replanning with uncertainty-aware execution could enable the robot to handle disturbances gracefully by pausing, requesting clarification, or autonomously selecting alternative skills when the current trajectory becomes infeasible.

8.3. Improved Language Understanding and Reasoning

The VLM's limitations in autonomous skill decomposition and combination suggest several avenues for enhancement. A multi-VLM architecture, similar to those presented in section 2.4, could enhance the framework's performance. It could be employed such that one model proposes schemas or skill selections and another verifies their semantic correctness, improving automatic schema generation and ambiguity handling without relying solely on a single model's output. The same approach could strengthen clarification behavior by ensuring that the system requests user input whenever multiple objects match a command and avoids executing partially matching skills. More sophisticated prompt engineering strategies, including

few-shot examples of successful skill decomposition and combination, may further improve the VLM's ability to identify when combination is appropriate and to select compatible component skills.

8.4. Alternative Interaction Modalities

The frontend visualization proved valuable during development but contradicts the goal of hands-free interaction. Augmented reality visualization of predicted trajectories directly in the workspace could preserve the safety benefits of trajectory preview while reducing reliance on a separate display. Gesture-based skill parameterization could complement natural language in scenarios where pointing or demonstrative gestures communicate intent more effectively than verbal descriptions. Multi-modal feedback that combines visual, auditory, and haptic cues may improve user awareness of the robot's state and intended actions, particularly in industrial environments where visual attention is limited.

8.5. Advanced Skill Composition

Learning compatibility constraints directly from demonstration data, rather than relying on VLM-based reasoning alone, could improve robustness in skill combination. Analyzing the covariance structure of demonstrated skills to automatically identify compatible frames for fusion, with and without covariance manipulation, would reduce the need for overspecific user prompts and enable automatic generation of compatibility lists.

Extending the current frame-fusion mechanism toward hierarchical skill organization would allow more complex behaviors composed of primitive motions, where abstract skills internally invoke sequences of lower-level skills. Sequential skill chaining with intermediate conditions, informed by perception-based state estimation, would further enable task execution that reacts to environment changes rather than relying solely on fixed timing.

8.6. Enhanced Demonstration Collection

The orientation variation constraints observed during demonstration and how it effects the skill composition inspires an alternative handling of KMPs and TP-KMPs. The idea is teaching atomic motions per KMP instead of teaching a complete skill to a TP-KMP. To form a skill, multiple atomic KMPs would be combined into a new TP-KMP, where each atomic KMP is augmented with compatibility padding in the temporal regions corresponding to other atomic motions. For example, combining three atomic motions would result in a structure where the first KMP contains its atomic motion followed by padding, the second contains padding, its atomic motion, then padding, and the third contains padding followed by its atomic motion. These paddings would include exaggerated variation in all DoFs, eliminating the need for later covariance manipulation. However, this approach would mean that every skill is a combination of frames, each containing an atomic motion, which increases the

cognitive burden on the VLM and introduces the challenge of defining what constitutes an atomic motion.

Semi-autonomous demonstration augmentation, where the system automatically generates synthetic demonstrations by perturbing recorded trajectories within feasible bounds, could further reduce the demonstration burden while ensuring sufficient variation.

The current reliance on physical buttons for demonstration control could be replaced with VLM-based voice commands, enabling fully hands-free operation during kinesthetic teaching. This would improve platform independence and enhance the naturalness of the teaching process.

8.7. Robustness and Safety

Enhancing robustness and safety is essential for deployment in dynamic, human-centric environments. Formal verification of combined skill trajectories could provide guarantees that motions remain within workspace limits, avoid known obstacles, and respect velocity and acceleration constraints. Uncertainty-aware execution with rollback capabilities would allow the robot to detect significant deviations from predicted trajectories and revert to safe states. Collision prediction and avoidance, together with graceful degradation strategies for perception failures, would further improve safety and reliability.

8.8. Closing Remarks

The directions outlined above collectively point toward a vision of robotic systems that combine the accessibility of natural language interaction with the precision and safety required for real-world deployment. As foundation models continue to evolve and their integration with robotics matures, the principles demonstrated in this work, specifically modular architecture, structured interfaces, hybrid approaches, and data-efficient learning, provide a foundation for systems that learn continuously from interaction while maintaining the geometric awareness and uncertainty quantification essential for manipulation tasks.

The democratization of robot programming through intuitive interfaces and minimal training requirements has the potential to significantly expand the applicability of robotic automation. By reducing the expertise required for teaching new skills and enabling non-expert operators to interact naturally with robotic systems, frameworks like the one presented here can bring the benefits of automation to domains where traditional programming approaches have proven prohibitive. The path toward robots that learn continuously from interaction, accumulating skills over their operational lifetime while adapting to the evolving needs of their environments, represents an ambitious but achievable goal that would fundamentally transform human-robot collaboration.

A. Appendix

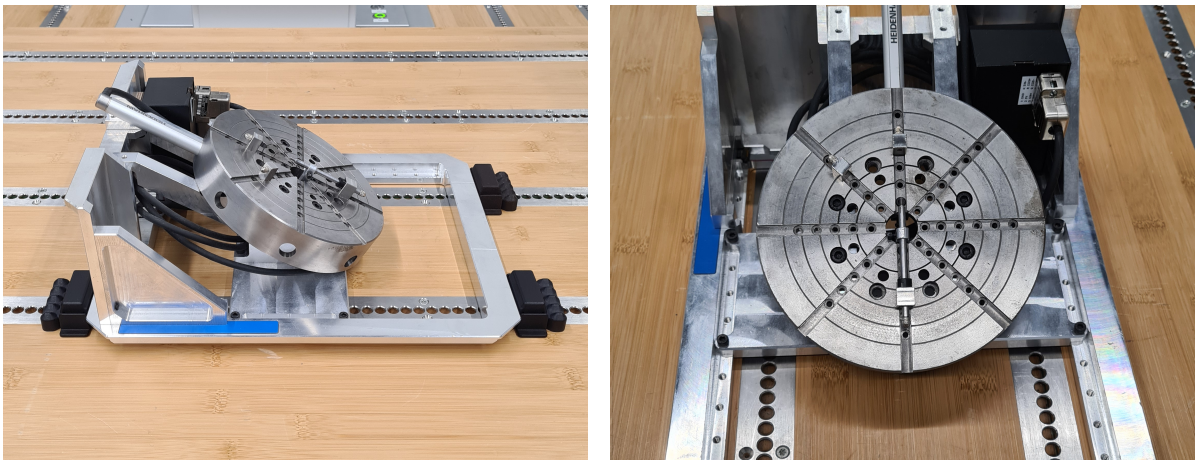

Figure A.1.: Images of the bearing ring measurement unit used during the evaluation (see subsection 5.2.3). (left) Side view. (right) Front view.

```
system_prompt = (
    """You are 'SARA', a robot assistant who interacts with and modulates the robot's behavior through skills.\n"""
    """You will act as the robot and stay in character.\n"""
    """Your primary task is to decide which skill(s) to use based on the user's query.\n"""
    """Skills that the robot can execute are exposed as tools prefixed with 'Skill'.\n"""
    """You will also be provided with a list of objects detected by the perception system.\n"""
    """Only use objects explicitly listed by the perception system. Do NOT reference or use objects that are not detected.\n"""
    """If a user references an object that is not detected or whose pose cannot be determined, inform the user clearly and do not attempt to use it in a skill.\n"""
    """If the perception system is missing an object that appears important based on the query or image, notify the user about this limitation.\n"""
    """If you are unable to execute a task due to a missing or unsuitable skill, you can combine skills into a new one to satisfy the user demand.\n"""
    """If you are still unable to identify an appropriate skill, inform the user clearly.\n"""
    """\n"""
    """IMPORTANT: Only execute skills if the user's query explicitly requests an action to be performed by the robot.\n"""
    """If the user's input is a greeting, general question, or informational request (e.g., 'hi', 'what can you do?', 'tell me about yourself'), DO NOT call any skills.\n"""
    """Instead, respond appropriately with text, clarifications, or descriptions, without triggering skill execution.\n"""
    """\n"""
```

```
    """Before deciding to execute any skill or combine skills, ask yourself:\n"""
    """- Does the user explicitly ask me to perform an action?\n"""
    """- Is there a clear command or task in the query that requires robot behavior?\n"""
    """Only if the answer is yes, proceed to find a matching skill or combine skills.\n"""
    """Otherwise, respond with a helpful message or ask for clarification.\n"""
    """\n"""
    """If no single existing skill fully satisfies the task, you MUST consider combining two skills to create a new, more appropriate one.\n"""
    """The system supports dynamic skill composition via the CombineSkills tool.\n"""
    """Combined skills inherit input fields from all source skills. You must provide a combination_dict that maps each source skill's action_name to the input field (with 'self.' prefix) that should be used in the combined skill.\n"""
    """\n"""
    """Never fall back to a partially matching skill if combining skills would fulfill the user's request better.\n"""
    """For example, if the query involves multiple sequential actions (e.g., grasp + rotate + place), and no existing skill handles the full sequence, use CombineSkills to create a hybrid skill.\n"""
    """Clearly explain the combination to the user and proceed with parameterizing the new combined skill.\n"""
    """\n"""
    """Only use a pre-existing single skill if it completely satisfies the user's query.\n"""
    """If no skill or combination is sufficient, explain that to the user directly.\n"""
    """\n"""
    """Always provide a concise, clear summary to the user explaining the actions you took or why a request could not be fulfilled.\n"""
    """Respond to the following query using ONLY your tool calls. You may call multiple tools per query.\n"""
)
```

Listing A.1: system prompt of the VLM receiving the user prompts.

B. Tools for VLM

B.1. Static Tools

Example Tool Structure

This template illustrates the standard structure for all tool definitions in this appendix. The docstring section (shown here) describes the tool's purpose and behavior. The VLM reads this description to understand when the tool should be applied.

Parameters:

`parameter_name` (*type*):
Description of the parameter's purpose, expected values, and any constraints. The VLM uses the parameter name, type, and description to generate appropriate tool invocations.

`another_parameter` (*type, optional*):
Tools may have multiple parameters. Each follows the same documentation pattern: name, type, requirement, and descriptive text explaining its role in the tool's operation.

NoToolIsAvailable

If there are no tools that can solve this problem, use this function.

Parameters:

`why_cannot_do_anything` (*str*):
Explain why you cannot call other functions adapting the persona of the robot.

TellUserAMessage

If there is something you want to communicate or say to the user use this tool, instead of saying it directly.

Parameters:

`message` (*str*):
Act as the robot and tell the user a message. Use the persona of the robot.

StartRobot

If you want to start the robot, use this function.

Parameters:

This tool requires no parameters.

StopRobot

If you want to stop the robot, use this function.

Parameters:

This tool requires no parameters.

CombineSkills

If you want to combine skills, use this function, by specifying which skill provides each input field in the new hybrid skill. The combined skill inherits fields from all skills. For each field in the new skill, the 'combination_dict' indicates from which original skill the framework should take that field's input.
The order of the skills in the combination_dict indicates the order of the skills in the new skill.
:param combination_ dict:
A dictionary mapping each **skill's action_name** to the **specific field name** of that skill that should be used in the combined skill.
- The keys are the action names of the two skills being combined.
- The values are the field names (including 'self.' prefix) from those skills that correspond to parts of the combined skill's input.

Parameters:

`combination_dict` (*dict[str, str]*):
Dictionary specifying which skill provides which field for the combined skill. Keys: skill action names. Values: field names from that skill (with 'self.' prefix).

B.2. Dynamic Tools

SkillStackObjects

This skill allows the robot to stack one object on top of another. The robot first identifies and grasps the object_to_stack, then carefully places it on top of the base_object. This ensures a stable stack of the two objects.

Object order:

1. `self.object_to_stack`
2. `self.base_object`

Parameters:

`object_to_stack` (*str*):
The name of the object that should be stacked on top. Choose an object from the list of detected objects.

`base_object` (*str*):
The name of the object that serves as the base for stacking. Choose an object from the list of detected objects.

SkillGraspAndPour

This skill enables the robot to grasp a specified object sideways and pour its contents into a target container by tilting the grasped object. The robot identifies and grasps the object sideways, then moves it over the target container and pours the contents into it.
Object order:

1. `self.object_to_grasp`
2. `self.target_container`

Parameters:

`object_to_grasp` (*str*):
The name of the object that should be grasped. Choose an object from the list of detected objects.

`target_container` (*str*):
The name of the container where the contents of the grasped object should be poured into. Choose an object from the list of detected objects.

SkillInsertRing

This skill is used to do a quality check on bearing rings. It assumes that the robot is already holding a ring in the gripper. The robot will approach the measurement unit that is used to do the quality check of the ring, insert the ring into the unit, and then move away from the measurement unit, leaving the ring inserted.
Object order:

1. `self.measurement_unit`

Parameters:

`measurement_unit` (*str*):
The measurement unit used to do quality checks on the rings. Choose an object from the list of detected objects.

For schema generation, a new connection to the VLM is established specifically for that task. This connection is separated from the one used to communicate with the user. Both connections maintain their own history and system prompt. Schema generation can be invoked when new skills are created either through user demonstrations (see section 4.2) or during skill combination (see section 4.4). If the user provides demonstrations to develop a new skill, the input modality of the VLM is images. For skill combination, on the other hand, the schemas of the skills being combined are passed to the VLM. Both cases use different system prompts, which can be found in Listing B.2 and Listing B.3. Both system prompts are extended with example schemas (see Listing B.1).

```
examples = (
  "## Example 1:\n"
  "action_name: 'SkillGraspAndPour'\n\n"
  "explanation: >\n"
  "  This skill enables the robot to grasp a specified object sideways and pour its contents\n"
  "  into a target container, by tilting the grasped object. The robot identifies and grasps the object sideways, then\n"
  "  moves it over the target container and pours the contents into it.\n\n"
  "possible_fields:\n"
  "  object_to_grasp:\n"
  "    type: 'str'\n"
  "    description: 'The name of the object that should be grasped. Choose an object from the list of detected objects.'\n"
  "  target_container:\n"
  "    type: 'str'\n"
  "    description: 'The name of the container where the grasped object should be poured into. Choose an object from the list of detected objects.'\n\n"
  "object_order:\n"
  "  – 'self.object_to_grasp'\n"
  "  – 'self.target_container'\n\n"
  "## Example 2:\n"
  "action_name: 'SkillGraspAndPlace'\n\n"
  "explanation: >\n"
  "  This skill directs the robot to grasp a specified object from the top and place it at a \n"
  "  designated location or on another object. The robot identifies the object to grasp, \n"
  "  picks it up from above, and then moves it to the target place, ensuring accurate placement.\n\n"
  "possible_fields:\n"
  "  object_to_grasp:\n"
  "    type: 'str'\n"
  "    description: 'the name of the object that should be grasped. Choose an object from the list of detected objects.'\n"
  "  object_to_place:\n"
  "    type: 'str'\n"
  "    description: 'the name of the object/place where the grasped object should be placed. Choose an object from the list of detected objects.'\n\n"
  "object_order:\n"
  "  – 'self.object_to_grasp'\n"
  "  – 'self.object_to_place'\n\n"
  "Now define a new robotic skill in the same format and structure."
```

```
)
```

Listing B.1: Example schemas in a python string. This string gets appended to the system prompts for generating schemas.

B.2.1. Generate Schema from Images

Figure B.1.: Images used to generate the schema for the skill `SkillStackObjects` (see subsubsection 4.1.2). The images were manually selected, since the perception module is not part of the framework (see subsection 4.1.1). (left) Grasping stage. (right) Stacking stage.

```
task_description_image = (
    "You have to create the content of a YAML file that is used to define a robotic skill, using the images you will be provided.\n"
    "There are fields that the YAML file has to contain, and there are fields that you have to come up with by yourself.\n"
    "A skill involves objects and describes an interaction the robot can perform.\n"
    "Use the format and structure shown in the examples below.\n"
    "Your output must only contain the YAML content and no extra text.\n\n"
    "Mandatory fields are:\n"
    "– 'action_name': the name of the action, written in CamelCase and prefixed with 'Skill'.\n"
    "– 'explanation': a detailed description of what the skill does and what happens in the scene when the skill is executed.\n"
    "– 'object_order': a list defining the order in which the objects are relevant to the skill. This refers to the order the robot is approaching/interacting with the objects. The items in this list must exactly match the keys used in 'possible_fields'.\n\n"
    "You must also include a 'possible_fields' section, where each key describes a required object in the skill, with its type (always 'str') and a short description.\n"
    "The names of these fields must be meaningful and follow snake_case formatting.\n\n"
    "Follow the YAML formatting from the examples. Output ONLY valid YAML content. Do NOT include any markdown code fences (e.g., '''yaml) or extra text.\n\n"
) + examples
```

Listing B.2: system prompt schema generation for new skills using images.

B.2.2. Generate Schema from Combined Skills

```
task_description_combining = (
    "You will be given the full YAML content of existing robotic skills, each describing a skill with its fields and object interaction order.\n\n"
    "For each skill, you will also be told a specific object from that skill's 'object_order' list, indicating the field to use from that skill.\n\n"
    "Your task is to generate a new combined skill YAML by:\n"
    "- Taking the specified object field from each input skill as input parameters.\n"
    "- Merging the 'possible_fields' sections to include only fields related to these specified objects.\n"
    "- Combining the 'object_order' list to reflect the order of the specified objects.\n"
    "- Creating a new 'action_name' that meaningfully reflects the combined skill.\n"
    "- Writing an 'explanation' that describes the combined behavior of the skills and how each object is used.\n"
    "- Ensuring all field names are meaningful, use snake_case, and follow the formatting of the examples.\n"
    "Your output must only contain the YAML content and no extra text.\n\n"
    "Example:\n"
    "If you receive:\n"
    "- Skill A YAML with object_order: ['self.object_to_grasp', 'self.target_container'] and are instructed to use 'self.object_to_grasp' from Skill A.\n"
    "- Skill B YAML with object_order: ['self.object_to_grasp', 'self.object_to_place'] and are instructed to use 'self.object_to_place' from Skill B.\n\n"
    "You should produce a combined skill YAML that includes:\n"
    "- The field 'object_to_grasp' from Skill A.\n"
    "- The field 'object_to_place' from Skill B.\n"
    "- The combined 'object_order' reflecting ['self.object_to_grasp', 'self.object_to_place'].\n"
    "- A new, meaningful 'action_name'.\n"
    "- A merged 'explanation' that covers grasping sideways (from Skill A) and placing (from Skill B).\n\n"
    "Output ONLY valid YAML content. Do NOT include any markdown code fences (e.g., ```yaml) or extra text.\n\n"
) + examples
```

Listing B.3: system prompt schema generation for combination of skills.

```
def _check_yaml_skill(yaml_content: str) -> tuple[bool, str]:
    """
    Checks if the given string represents a YAML skill.
    :param yaml_content: string in YAML format
    :return: A tuple containing a boolean indicating if the given string represents a YAML skill plus a description of the error if there is one.
    """
    try:
        data = yaml.safe_load(yaml_content)
    except yaml.YAMLError as e:
        return False, f"YAML parsing error: {e}"

    # Check mandatory top-level fields
    for field in ["action_name", "explanation", "possible_fields", "object_order"]:
        if field not in data:
            return False, f"Missing mandatory field: {field}"

    def is_camel_case_skill_name(name: str) -> bool:
        return bool(re.fullmatch(r"Skill[A-Z][a-zA-Z0-9]*", name))
```

```
19
20  def is_snake_case(s: str) -> bool:
21      return bool(re.fullmatch(r"[a-z]+(_[a-z0-9]+)*", s))
22
23  # Check action_name format
24  if not is_camel_case_skill_name(data["action_name"]):
25      return False, "Field 'action_name' must be CamelCase and start with 'Skill'"
26
27  # Check explanation is non-empty string
28  if not isinstance(data["explanation"], str) or not data["explanation"].strip():
29      return False, "Field 'explanation' must be a non-empty string"
30
31  # Check possible_fields structure
32  pf = data["possible_fields"]
33  if not isinstance(pf, dict) or not pf:
34      return False, "Field 'possible_fields' must be a non-empty dictionary"
35
36  for key, val in pf.items():
37      if not is_snake_case(key):
38          return False, f"possible_fields key '{key}' is not in snake_case"
39      if not isinstance(val, dict):
40          return False, f"possible_fields['{key}'] must be a dictionary"
41      if val.get("type") != "str":
42          return False, f"possible_fields['{key}']['type'] must be 'str'"
43      if "description" not in val or not isinstance(val["description"], str) or not val["description"].strip():
44          return False, f"possible_fields['{key}']['description'] must be a non-empty string"
45
46  # Check object_order is list with valid references
47  obj_order = data["object_order"]
48  if not isinstance(obj_order, list) or not obj_order:
49      return False, "Field 'object_order' must be a non-empty list"
50
51  possible_keys = set(pf.keys())
52  for item in obj_order:
53      if not isinstance(item, str):
54          return False, f"object_order entry '{item}' is not a string"
55      if not item.startswith("self."):
56          return False, f"object_order entry '{item}' must start with 'self.'"
57      ref_key = item[len("self.") :]
58      if ref_key not in possible_keys:
59          return False, f"object_order entry '{item}' references unknown field '{ref_key}'"
60
61  return True, "YAML skill definition is valid"
```

Listing B.4: Function used to validate the format and fields of a generated schema.

Listing B.5: Schema of the combined skill containing the motion to grasp an object and then insert it into a bearing ring measurement unit. The schema was generated using the `Qwen2.5-VL-72B-Instruct` [97].

```
action_name: 'SkillGraspAndCheck'
```

```
explanation: "This skill combines the actions of grasping an object from the top and performing a quality
      check using a measurement unit. The robot first identifies and securely picks up the specified object from
      above. After grasping the object, the robot approaches the designated measurement unit for quality
      checking. The object is then inserted into the measurement unit for inspection, and the robot moves away,
      leaving the object in place for further analysis."

possible_fields:
  object_to_grasp:
    type: 'str'
    description: 'the name of the object that should be grasped. Choose an object from the list of detected
        objects.'
  measurement_unit:
    type: 'str'
    description: 'The measurement unit to do quality checks on the rings. Choose an object from the list of
        detected objects.'

object_order:
  - 'self.object_to_grasp'
  - 'self.measurement_unit'
```

List of Figures

List of Tables

Glossary

chain of thought A prompting technique that encourages foundation models to generate intermediate reasoning steps before arriving at a final answer, enhancing their problem-solving capabilities. 15

context window The amount of information or text that a foundation model can consider at one time when generating responses or making predictions. 2, 62

few-shot Few-shot learning, is a machine learning approach where a model is trained to perform tasks with only a small number of examples or demonstrations. It can also refer to a system prompt that includes these examples. 67, 68

fine-tuning The process of adapting a pretrained machine learning model to a specific task or domain by continuing training on task-specific data, typically with a lower learning rate to preserve the general knowledge acquired during pretraining. iv, v, 3, 10, 11, 13, 21, 32, 50, 60, 61, 65

kinesthetic demonstration A method of teaching robots by physically guiding them through desired movements or tasks, allowing them to learn from direct experience. iv, v, 3, 18, 21, 38, 65, 81

prompt engineering The process of designing and refining prompts to effectively communicate with foundation models and elicit desired responses or behaviors. 14, 53, 54, 67

sim-to-real gap The performance degradation that occurs when transferring policies learned in simulation to real-world robotic systems, caused by differences in physics modeling, sensor noise, and environmental factors between simulated and physical environments. 5

system prompt A directive or instruction given to a foundation model to guide its responses or behavior. 2, 19, 40, 46, 54, 62, 63, 71, 75–77

zero-shot Zero-shot Learning, is a machine learning approach where a model is able to perform tasks without any prior examples or demonstrations, often by leveraging knowledge from related tasks or domains. iv, v, 2, 9, 10, 65

Acronyms

AAE Augmented Autoencoder. 33, 50, 60

API Application Programming Interface. 18, 80

CLEA Closed-Loop Embodied Agents. 11, 15, 16, 20–22, 80

DAE Denoising Autoencoder. 33

DLR German Aerospace Center. iv, v, 48, 62, 65

DMP Dynamic Movement Primitive. 18, 19, 21, 23

DoF Degree of Freedom. iv, v, 36, 37, 68

EE End-Effector. 23, 28, 34, 37, 39, 50, 52, 56, 58, 59, 64

FIFO First In, First Out. 15

GMM Gaussian Mixture Model. 2, 6, 7, 23

GMR Gaussian Mixture Regression. 6, 7, 24

GP Gaussian Process. 2, 6, 7

GPR Gaussian Process Regression. 6, 7, 26

GPU Graphical Processing Unit. 13, 20, 21

ICL In-Context Learning. 2, 14, 16, 18, 21, 22, 62

ICP Iterative Closest Point. 33

IL Imitation Learning. iv, v, 5–8, 37

IMU Inertial Measurement Unit. 37

JSON JavaScript Object Notation. 35, 61

KL-Divergence *Kullback-Leibler* Divergence. 7, 23, 24

KMP Kernelized Movement Primitive. iv, v, 2, 4, 7, 23, 24, 26–29, 31, 32, 37, 41–47, 50, 56, 58, 59, 62–65, 68, 81

LED Light-Emitting Diode. 37

LfD Learning from Demonstration. 5, 12, 16

LLM Large Language Model. 2, 9–16, 18, 19, 21, 47, 60, 61, 80

MLP Multi-Layer Perceptron. 15, 21

PCIe Peripheral Component Interface Express. 37

POMDP Partially Observable Markov Decision Process. 15

ProMP Probabilistic Movement Primitive. 23

PTP Point-to-Point. 1

RAP Retrieval-Augmented Planning. 11, 14, 15, 20–22, 80

RSLS Robotic Skill Learning System. 19–22

SARA Safe, Autonomous Robotic Assistant. iv, v, 36, 37, 40, 47, 49, 52, 59, 62, 65, 81

TCP Tool Center Point. 40

TP-GMM Task-Parameterized Gaussian Mixture Model. 8, 28, 60

TPIL Task-Parameterized Imitation Learning. iv, v, 3, 8, 9, 60, 65

TP-KMP Task-Parameterized Kernelized Movement Primitive. iv, v, 3, 4, 7, 8, 11–14, 19–22, 28–32, 34–43, 45, 46, 48, 50, 52, 55, 56, 60–62, 64, 65, 68, 81

VLA Vision-Language-Action Model. iv, v, 2, 3, 9, 10, 20

VLM Vision-Language Model. iv, v, 2–4, 9–13, 15, 19, 21, 31, 32, 34–36, 39–41, 45–47, 51–55, 59–63, 65, 67–69, 71, 72, 75

YAML Yet Another Markup Language. 35, 36, 39, 40, 46, 61

YCB Dataset *Yale-CMU-Berkeley* Object and Model Set. 32, 47, 48, 50, 51, 61, 67, 82

Bibliography

[1] S. Kumar, C. Savur, and F. Sahin. "Survey of Human–Robot Collaboration in Industrial Settings: Awareness, Intelligence, and Compliance". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51.1 (2021), pp. 280–297. DOI: 10.1109/TSMC.2020.3041231.

[2] U. Othman and E. Yang. "Human–Robot Collaborations in Smart Manufacturing Environments: Review and Outlook". In: *Sensors* 23.12 (2023). ISSN: 1424-8220. DOI: 10.3390/s23125663. URL: https://www.mdpi.com/1424-8220/23/12/5663.

[3] L. Pietrantoni, M. Favilla, F. Fraboni, E. Mazzoni, S. Morandini, M. Benvenuti, and M. De Angelis. "Integrating collaborative robots in manufacturing, logistics, and agriculture: Expert perspectives on technical, safety, and human factors". In: *Frontiers in Robotics and AI* Volume 11 - 2024 (2024). ISSN: 2296-9144. DOI: 10.3389/frobt.2024.1342130. URL: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2024.1342130.

[4] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. "An Algorithmic Perspective on Imitation Learning". In: *Foundations and Trends® in Robotics* 7.1–2 (2018), pp. 1–179. ISSN: 1935-8261. DOI: 10.1561/2300000053. URL: http://dx.doi.org/10.1561/2300000053.

[5] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard. "Recent Advances in Robot Learning from Demonstration". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), pp. 297–330. DOI: 10.1146/annurev-control-100819-063206.

[6] S. Schaal. "Is imitation learning the route to humanoid robots?" In: *Trends in Cognitive Sciences* 3.6 (1999), pp. 233–242. ISSN: 1364-6613. DOI: https://doi.org/10.1016/S1364-6613(99)01327-3. URL: https://www.sciencedirect.com/science/article/pii/S1364661399013273.

[7] C. Lynch and P. Sermanet. *Language Conditioned Imitation Learning over Unstructured Data*. 2021. arXiv: 2005.07648 [cs.RO]. URL: https://arxiv.org/abs/2005.07648.

[8] A. Bucker, L. Figueredo, S. Haddadin, A. Kapoor, S. Ma, S. Vemprala, and R. Bonatti. *LATTE: LAnguage Trajectory TransformEr*. 2022. arXiv: 2208.02918 [cs.RO]. URL: https://arxiv.org/abs/2208.02918.

[9] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence. *PaLM-E: An Embodied Multimodal Language Model*. 2023. arXiv: 2303.03378 [cs.LG]. URL: https://arxiv.org/abs/2303.03378.

[10] M. Shridhar, L. Manuelli, and D. Fox. *CLIPort: What and Where Pathways for Robotic Manipulation*. 2021. arXiv: 2109.12098 [cs.RO]. URL: https://arxiv.org/abs/2109.12098.

[11] Q. Luo, Y. Li, and Y. Wu. *Grounding Object Relations in Language-Conditioned Robotic Manipulation with Semantic-Spatial Reasoning*. 2023. arXiv: 2303.17919 [cs.RO]. URL: https://arxiv.org/abs/2303.17919.

[12] X. Xiao, J. Liu, Z. Wang, Y. Zhou, Y. Qi, Q. Cheng, B. He, and S. Jiang. *Robot Learning in the Era of Foundation Models: A Survey*. 2023. arXiv: 2311.14379 [cs.RO]. URL: https://arxiv.org/abs/2311.14379.

[13] R. Firoozi, J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman, B. Ichter, D. Driess, J. Wu, C. Lu, and M. Schwager. *Foundation Models in Robotics: Applications, Challenges, and the Future*. 2023. arXiv: 2312.07843 [cs.RO]. URL: https://arxiv.org/abs/2312.07843.

[14] M. Ahn, D. Dwibedi, C. Finn, M. G. Arenas, K. Gopalakrishnan, K. Hausman, B. Ichter, A. Irpan, N. Joshi, R. Julian, S. Kirmani, I. Leal, E. Lee, S. Levine, Y. Lu, I. Leal, S. Maddineni, K. Rao, D. Sadigh, P. Sanketi, P. Sermanet, Q. Vuong, S. Welker, F. Xia, T. Xiao, P. Xu, S. Xu, and Z. Xu. *AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents*. 2024. arXiv: 2401.12963 [cs.RO]. URL: https://arxiv.org/abs/2401.12963.

[15] R. Bommasani, D. A. Hudson, E. Adeli, et al. *On the Opportunities and Risks of Foundation Models*. 2022. arXiv: 2108.07258 [cs.LG]. URL: https://arxiv.org/abs/2108.07258.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[17] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. URL: https://arxiv.org/abs/2103.00020.

[18] R. Sapkota, Y. Cao, K. I. Roumeliotis, and M. Karkee. *Vision-Language-Action Models: Concepts, Progress, Applications and Challenges*. 2025. arXiv: 2505.04769 [cs.CV]. URL: https://arxiv.org/abs/2505.04769.

[19] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. 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*. 2024. arXiv: 2406.09246 [cs.RO]. URL: https://arxiv.org/abs/2406.09246.

[20] Y. Zhao, Z. Zhou, and R. Xiong. "Grounding Language for Robotic Manipulation via Skill Library". In: *2023 2nd International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM)*. 2023, pp. 385–392. DOI: 10.1109/MLCCIM60412.2023.00062.

[21] Y. Yin, Z. Wang, Y. Sharma, D. Niu, T. Darrell, and R. Herzig. *In-Context Learning Enables Robot Action Prediction in LLMs*. 2025. arXiv: 2410.12782 [cs.RO]. URL: https://arxiv.org/abs/2410.12782.

[22] S. Wang, S. Zhang, J. Zhang, R. Hu, X. Li, T. Zhang, J. Li, F. Wu, G. Wang, and E. Hovy. *Reinforcement Learning Enhanced LLMs: A Survey*. 2025. arXiv: 2412.10400 [cs.CL]. URL: https://arxiv.org/abs/2412.10400.

[23] S. Calinon, F. Guenter, and A. Billard. "On Learning, Representing, and Generalizing a Task in a Humanoid Robot". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.2 (2007), pp. 286–298. DOI: 10.1109/TSMCB.2006.886952.

[24] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006. ISBN: 978-0-262-18253-9. URL: http://www.gaussianprocess.org/gpml/.

[25] Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell. *Kernelized Movement Primitives*. 2018. arXiv: 1708.08638 [cs.RO]. URL: https://arxiv.org/abs/1708.08638.

[26] S. Calinon, D. Bruno, and D. G. Caldwell. "A task-parameterized probabilistic model with minimal intervention control". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 3339–3344. DOI: 10.1109/ICRA.2014.6907339.

[27] Y. Huang, J. Silvério, L. Rozo, and D. G. Caldwell. *Generalized Task-Parameterized Skill Learning*. 2018. arXiv: 1707.01696 [cs.RO]. URL: https://arxiv.org/abs/1707.01696.

[28] S. Calinon. "A tutorial on Task-Parameterized Movement Learning and Retrieval". In: *Intelligent Service Robotics* 9.1 (Jan. 1, 2016), pp. 1–29. ISSN: 1861-2784. DOI: 10.1007/s11370-015-0187-9. URL: https://doi.org/10.1007/s11370-015-0187-9 (visited on 05/19/2025).

[29] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. "A survey of robot learning from demonstration". In: *Robotics and Autonomous Systems* 57.5 (2009), pp. 469–483. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2008.10.024. URL: https://www.sciencedirect.com/science/article/pii/S0921889008001772.

[30] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi. *A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges*. 2023. arXiv: 2309.02473 [cs.LG]. URL: https://arxiv.org/abs/2309.02473.

[31] O. Heimann and J. Guhl. "Industrial Robot Programming Methods: A Scoping Review". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2020, pp. 696–703. DOI: 10.1109/ETFA46521.2020.9211997.

[32] A. Correia and L. A. Alexandre. *A Survey of Demonstration Learning*. 2023. arXiv: 2303.11191 [cs.LG]. URL: https://arxiv.org/abs/2303.11191.

[33] J. Kober, J. Bagnell, and J. Peters. "Reinforcement Learning in Robotics: A Survey". In: *The International Journal of Robotics Research* 32 (Sept. 2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.

[34] D. Han, B. Mulyana, V. Stankovic, and S. Cheng. "A Survey on Deep Reinforcement Learning Algorithms for Robotic Manipulation". In: *Sensors* 23 (Apr. 2023). DOI: 10.3390/s23073762.

[35] W. Zhao, J. Peña Queralta, and T. Westerlund. "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey". In: Dec. 2020, pp. 737–744. DOI: 10.1109/SSCI47803.2020.9308468.

[36] E. Pignat and S. Calinon. "Bayesian Gaussian Mixture Model for Robotic Policy Imitation". In: *IEEE Robotics and Automation Letters* 4.4 (Oct. 2019), pp. 4452–4458. ISSN: 2377-3774. DOI: 10.1109/lra.2019.2932610. URL: http://dx.doi.org/10.1109/LRA.2019.2932610.

[37] J. Ko, D. J. Klein, D. Fox, and D. Haehnel. "Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 742–747. DOI: 10.1109/ROBOT.2007.363075.

[38] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. *One-Shot Imitation Learning*. 2017. arXiv: 1703.07326 [cs.AI]. URL: https://arxiv.org/abs/1703.07326.

[39] S. Xiao, X. Chen, Y. Lu, J. Ye, and H. Wu. "A KMP-based interactive learning approach for robot trajectory adaptation with obstacle avoidance". In: *Industrial Robot: the international journal of robotics research and application* 51.2 (Jan. 18, 2024). Publisher: Emerald Publishing Limited, pp. 326–339. ISSN: 0143-991X. DOI: 10.1108/IR-11-2023-0284. URL: https://www.emerald.com/insight/content/doi/10.1108/ir-11-2023-0284/full/html (visited on 05/19/2025).

[40] M. Knauer, A. Albu-Schäffer, F. Stulp, and J. Silvério. "Interactive Incremental Learning of Generalizable Skills With Local Trajectory Modulation". In: *IEEE Robotics and Automation Letters* 10.4 (2025), pp. 3398–3405. DOI: 10.1109/LRA.2025.3542209.

[41] J. Richter, J. Oliveira, C. Scheurer, J. Steil, and N. Dehio. *Task-Parameterized Imitation Learning with Time-Sensitive Constraints*. 2023. arXiv: 2312.03506 [cs.RO]. URL: https://arxiv.org/abs/2312.03506.

[42] J. Zhu, M. Gienger, and J. Kober. "Learning Task-Parameterized Skills From Few Demonstrations". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4063–4070. DOI: 10.1109/LRA.2022.3150013.

[43] H. Perez-Villeda, J. Piater, and M. Saveriano. "Learning and extrapolation of robotic skills using task-parameterized equation learner networks". In: *Robotics and Autonomous Systems* 160 (Feb. 2023), p. 104309. ISSN: 0921-8890. DOI: 10.1016/j.robot.2022.104309. URL: http://dx.doi.org/10.1016/j.robot.2022.104309.

[44] J. O. von Hartz, T. Welschehold, A. Valada, and J. Boedecker. *The Art of Imitation: Learning Long-Horizon Manipulation Tasks from Few Demonstrations*. 2024. arXiv: 2407.13432 [cs.RO]. URL: https://arxiv.org/abs/2407.13432.

[45] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL: https://arxiv.org/abs/2307.09288.

[46] K. Kawaharazuka, T. Matsushima, A. Gambardella, J. Guo, C. Paxton, and A. Zeng. "Real-world robot applications of foundation models: a review". In: *Advanced Robotics* 38.18 (Sept. 2024), pp. 1232–1254. ISSN: 1568-5535. DOI: 10.1080/01691864.2024.2408593. URL: http://dx.doi.org/10.1080/01691864.2024.2408593.

[47] J. Huang and K. C.-C. Chang. *Towards Reasoning in Large Language Models: A Survey*. 2023. arXiv: 2212.10403 [cs.CL]. URL: https://arxiv.org/abs/2212.10403.

[48] H. Liu, A. Chen, Y. Zhu, A. Swaminathan, A. Kolobov, and C.-A. Cheng. *Interactive Robot Learning from Verbal Correction*. 2023. arXiv: 2310.17555 [cs.RO]. URL: https://arxiv.org/abs/2310.17555.

[49] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox. *Correcting Robot Plans with Natural Language Feedback*. 2022. arXiv: 2204.05186 [cs.RO]. URL: https://arxiv.org/abs/2204.05186.

[50] T. Ma, J. Zhou, Z. Wang, R. Qiu, and J. Liang. *Contrastive Imitation Learning for Language-guided Multi-Task Robotic Manipulation*. 2024. arXiv: 2406.09738 [cs.RO]. URL: https://arxiv.org/abs/2406.09738.

[51] P. Sundaresan, S. Belkhale, D. Sadigh, and J. Bohg. *KITE: Keypoint-Conditioned Policies for Semantic Manipulation*. 2023. arXiv: 2306.16605 [cs.RO]. URL: https://arxiv.org/abs/2306.16605.

[52] C. Yang, P. Zhou, and J. Qi. *Integrating Visual Foundation Models for Enhanced Robot Manipulation and Motion Planning: A Layered Approach*. 2023. arXiv: 2309.11244 [cs.RO]. URL: https://arxiv.org/abs/2309.11244.

[53] Y. Chen, S. Tian, S. Liu, Y. Zhou, H. Li, and D. Zhao. *ConRFT: A Reinforced Fine-tuning Method for VLA Models via Consistency Policy*. 2025. arXiv: 2502.05450 [cs.RO]. URL: https://arxiv.org/abs/2502.05450.

[54] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV]. URL: https://arxiv.org/abs/2304.07193.

[55] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. *Sigmoid Loss for Language Image Pre-Training*. 2023. arXiv: 2303.15343 [cs.CV]. URL: https://arxiv.org/abs/2303.15343.

[56] Z. Li, C. Xie, and E. D. Cubuk. *Scaling (Down) CLIP: A Comprehensive Analysis of Data, Architecture, and Training Strategies*. 2024. arXiv: 2404.08197 [cs.CV]. URL: https://arxiv.org/abs/2404.08197.

[57] 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 for Robotics*. 2024. arXiv: 2406.10721 [cs.RO]. URL: https://arxiv.org/abs/2406.10721.

[58] J. Yang, W. Tan, C. Jin, K. Yao, B. Liu, J. Fu, R. Song, G. Wu, and L. Wang. "Transferring Foundation Models for Generalizable Robotic Manipulation". In: *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2025, pp. 1999–2010. DOI: 10.1109/WACV61041.2025.00201.

[59] K. T. Ly, K. Lu, and I. Havoutis. *InteLiPlan: An Interactive Lightweight LLM-Based Planner for Domestic Robot Autonomy*. 2025. arXiv: 2409.14506 [cs.RO]. URL: https://arxiv.org/abs/2409.14506.

[60] T. Kojima, Y. Zhu, Y. Iwasawa, T. Kitamura, G. Yan, S. Morikuni, R. Takanami, A. Solano, T. Matsushima, A. Murakami, and Y. Matsuo. *A Comprehensive Survey on Physical Risk Control in the Era of Foundation Model-enabled Robotics*. 2025. arXiv: 2505.12583 [cs.RO]. URL: https://arxiv.org/abs/2505.12583.

[61] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. *Do As I Can, Not As I Say: Grounding Language in Robotic Affordances*. 2022. arXiv: 2204.01691 [cs.RO]. URL: https://arxiv.org/abs/2204.01691.

[62] T. Kagaya, T. J. Yuan, Y. Lou, J. Karlekar, S. Pranata, A. Kinose, K. Oguri, F. Wick, and Y. You. *RAP: Retrieval-Augmented Planning with Contextual Memory for Multimodal LLM Agents*. 2024. arXiv: 2402.03610 [cs.LG]. URL: https://arxiv.org/abs/2402.03610.

[63] M. Lei, G. Wang, Y. Zhao, Z. Mai, Q. Zhao, Y. Guo, Z. Li, S. Cui, Y. Han, and J. Ren. *CLEA: Closed-Loop Embodied Agent for Enhancing Task Execution in Dynamic Environments*. 2025. arXiv: 2503.00729 [cs.RO]. URL: https://arxiv.org/abs/2503.00729.

[64] S. Karamcheti, S. Nair, A. Balakrishna, P. Liang, T. Kollar, and D. Sadigh. *Prismatic VLMs: Investigating the Design Space of Visually-Conditioned Language Models*. 2024. arXiv: 2402.07865 [cs.CV]. URL: https://arxiv.org/abs/2402.07865.

[65] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[66] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. *BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning*. 2022. arXiv: 2202.02005 [cs.RO]. URL: https://arxiv.org/abs/2202.02005.

[67] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. *MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale*. 2021. arXiv: 2104.08212 [cs.RO]. URL: https://arxiv.org/abs/2104.08212.

[68] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. *PaLM: Scaling Language Modeling with Pathways*. 2022. arXiv: 2204.02311 [cs.CL]. URL: https://arxiv.org/abs/2204.02311.

[69] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. *Finetuned Language Models Are Zero-Shot Learners*. 2022. arXiv: 2109.01652 [cs.CL]. URL: https://arxiv.org/abs/2109.01652.

[70] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht. *ALFWorld: Aligning Text and Embodied Environments for Interactive Learning*. 2021. arXiv: 2010.03768 [cs.CL]. URL: https://arxiv.org/abs/2010.03768.

[71] S. Yao, H. Chen, J. Yang, and K. Narasimhan. *WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents*. 2023. arXiv: 2207.01206 [cs.CL]. URL: https://arxiv.org/abs/2207.01206.

[72] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry. *Gymnasium Robotics*. Version 1.4.0. 2024. URL: http://github.com/Farama-Foundation/Gymnasium-Robotics.

[73] T. Yu, D. Quillen, Z. He, R. Julian, A. Narayan, H. Shively, A. Bellathur, K. Hausman, C. Finn, and S. Levine. *Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning*. 2021. arXiv: 1910.10897 [cs.LG]. URL: https://arxiv.org/abs/1910.10897.

[74] OpenAI. *GPT-3.5 – Function Calling and Other Updates*. https://openai.com/blog/function-calling-and-other-api-updates/. Accessed: 2025-07-29. 2023.

[75] H. Liu, C. Li, Q. Wu, and Y. J. Lee. *Visual Instruction Tuning*. 2023. arXiv: 2304.08485 [cs.CV]. URL: https://arxiv.org/abs/2304.08485.

[76] Q. Team. *Qwen2.5: A Party of Foundation Models*. Sept. 2024. URL: https://qwenlm.github.io/blog/qwen2.5/.

[77] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, Q. Jiang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang. *Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection*. 2024. arXiv: 2303.05499 [cs.CV]. URL: https://arxiv.org/abs/2303.05499.

[78] J. Grannen, S. Karamcheti, S. Mirchandani, P. Liang, and D. Sadigh. *Vocal Sandbox: Continual Learning and Adaptation for Situated Human-Robot Collaboration*. 2024. arXiv: 2411.02599 [cs.RO]. URL: https://arxiv.org/abs/2411.02599.

[79] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. URL: https://arxiv.org/abs/2304.02643.

[80] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. *Diffusion Policy: Visuomotor Policy Learning via Action Diffusion*. 2024. arXiv: 2303.04137 [cs.RO]. URL: https://arxiv.org/abs/2303.04137.

[81] N. Ingelhag, J. Munkeby, J. van Haastregt, A. Varava, M. C. Welle, and D. Kragic. "A Robotic Skill Learning System Built Upon Diffusion Policies and Foundation Models". In: *2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN)*. 2024, pp. 748–754. DOI: 10.1109/RO-MAN60168.2024.10731242.

[82] Z. Zhou, J. Song, X. Xie, Z. Shu, L. Ma, D. Liu, J. Yin, and S. See. "Towards Building AI-CPS with NVIDIA Isaac Sim: An Industrial Benchmark and Case Study for Robotics Manipulation". In: *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP '24. ACM, Apr. 2024, pp. 263–274. DOI: 10.1145/3639477.3639740. URL: http://dx.doi.org/10.1145/3639477.3639740.

[83] OpenAI, J. Achiam, S. Adler, et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[84] G. Team, R. Anil, S. Borgeaud, et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2025. arXiv: 2312.11805 [cs.CL]. URL: https://arxiv.org/abs/2312.11805.

[85] A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". English. In: *Neural Computation* 25.2 (Feb. 2013), pp. 328–373. ISSN: 0899-7667. DOI: 10.1162/NECO_a_00393.

[86] A. Paraschos, C. Daniel, and J. Peters. "Probabilistic Movement Primitives". In: Jan. 2013.

[87] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. ISSN: 00034851. URL: http://www.jstor.org/stable/2236703 (visited on 10/04/2025).

[88] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV]. URL: https://arxiv.org/abs/2207.02696.

[89] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel. "Implicit 3D Orientation Learning for 6D Object Detection from RGB Images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[90] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar. "Yale-CMU-Berkeley dataset for robotic manipulation research". In: *The International Journal of Robotics Research* 36.3 (2017), pp. 261–268. DOI: 10.1177/0278364917700714. eprint: https://doi.org/10.1177/0278364917700714. URL: https://doi.org/10.1177/0278364917700714.

[91] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. "The YCB object and Model set: Towards common benchmarks for manipulation research". In: *2015 International Conference on Advanced Robotics (ICAR)*. 2015, pp. 510–517. DOI: 10.1109/ICAR.2015.7251504.

[92] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research* 11.110 (2010), pp. 3371–3408. URL: http://jmlr.org/papers/v11/vincent10a.html.

[93] Z. Zhang. *Iterative point matching for registration of free-form curves*. Research Report RR-1658. INRIA, 1992, p. 42. URL: https://inria.hal.science/inria-00074899.

[94] Z. Wang, Z. Cheng, H. Zhu, D. Fried, and G. Neubig. *What Are Tools Anyway? A Survey from the Language Model Perspective*. 2024. arXiv: 2403.15452 [cs.CL]. URL: https://arxiv.org/abs/2403.15452.

[95] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. *Toolformer: Language Models Can Teach Themselves to Use Tools*. 2023. arXiv: 2302.04761 [cs.CL]. URL: https://arxiv.org/abs/2302.04761.

[96] Y. Qin, S. Hu, Y. Lin, W. Chen, N. Ding, G. Cui, Z. Zeng, Y. Huang, C. Xiao, C. Han, Y. R. Fung, Y. Su, H. Wang, C. Qian, R. Tian, K. Zhu, S. Liang, X. Shen, B. Xu, Z. Zhang, Y. Ye, B. Li, Z. Tang, J. Yi, Y. Zhu, Z. Dai, L. Yan, X. Cong, Y. Lu, W. Zhao, Y. Huang, J. Yan, X. Han, X. Sun, D. Li, J. Phang, C. Yang, T. Wu, H. Ji, Z. Liu, and M. Sun. *Tool Learning with Foundation Models*. 2024. arXiv: 2304.08354 [cs.CL]. URL: https://arxiv.org/abs/2304.08354.

[97] Q. Team. *Qwen2.5-VL*. Jan. 2025. URL: https://qwenlm.github.io/blog/qwen2.5-vl/.

[98] M. Iskandar, C. Ott, O. Eiberger, M. Keppler, A. Albu-Schäffer, and A. Dietrich. "Joint-Level Control of the DLR Lightweight Robot SARA". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 8903–8910. DOI: 10.1109/IROS45743.2020.9340700.

[99] N. Hogan. "Impedance control of industrial robots". In: *Robotics and Computer-Integrated Manufacturing* 1.1 (1984), pp. 97–113. ISSN: 0736-5845. DOI: 10.1016/0736-5845(84)90084-X. URL: https://www.sciencedirect.com/science/article/pii/073658458490084X.

[100] A. Albu-Schaffer, C. Ott, U. Frese, and G. Hirzinger. "Cartesian impedance control of redundant robots: recent results with the DLR-light-weight-arms". In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 3. 2003, 3704–3709 vol.3. DOI: 10.1109/ROBOT.2003.1242165.

[101] P. Agrawal, S. Antoniak, E. B. Hanna, B. Bout, D. Chaplot, J. Chudnovsky, D. Costa, B. D. Monicault, S. Garg, T. Gervet, S. Ghosh, A. Héliou, P. Jacob, A. Q. Jiang, K. Khandelwal, T. Lacroix, G. Lample, D. L. Casas, T. Lavril, T. L. Scao, A. Lo, W. Marshall, L. Martin, A. Mensch, P. Muddireddy, V. Nemychnikova, M. Pellat, P. V. Platen, N. Raghuraman, B. Rozière, A. Sablayrolles, L. Saulnier, R. Sauvestre, W. Shang, R. Soletskyi, L. Stewart, P. Stock, J. Studnia, S. Subramanian, S. Vaze, T. Wang, and S. Yang. *Pixtral 12B*. 2024. arXiv: 2410.07073 [cs.CV]. URL: https://arxiv.org/abs/2410.07073.

[102] DeepSeek-AI, D. Guo, D. Yang, et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. arXiv: 2501.12948 [cs.CL]. URL: https://arxiv.org/abs/2501.12948.

[103] L. Molinari. "Gram-Schmidt'sches Orthogonalisierungsverfahren". In: *Numerische Prozeduren: Aus Nachlass und Lehre*. Ed. by W. Gander, L. Molinari, and H. Švecová. Basel: Birkhäuser Basel, 1977, pp. 77–93. ISBN: 978-3-0348-7176-1. DOI: 10.1007/978-3-0348-7176-1_4. URL: https://doi.org/10.1007/978-3-0348-7176-1_4.

[104] K. Gu, Z. Li, S. Liu, J. Liu, S. Xu, Y. Yan, M. B. Mi, K. Kawaguchi, and A. Yao. *Learning Unorthogonalized Matrices for Rotation Estimation - Sec. 3.2 '6D Representation for 3D Rotations'*. 2023. arXiv: 2312.00462 [cs.CV]. URL: https://arxiv.org/abs/2312.00462.

[105] 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*. 2023. arXiv: 2306.08647 [cs.RO]. URL: https://arxiv.org/abs/2306.08647.

[106] T. Kwon, N. D. Palo, and E. Johns. "Language Models as Zero-Shot Trajectory Generators". In: *IEEE Robotics and Automation Letters* 9.7 (July 2024), pp. 6728–6735. ISSN: 2377-3774. DOI: 10.1109/lra.2024.3410155. URL: http://dx.doi.org/10.1109/LRA.2024.3410155.

[107] C. Ming, J. Lin, P. Fong, H. Wang, X. Duan, and J. He. *HiCRISP: An LLM-based Hierarchical Closed-Loop Robotic Intelligent Self-Correction Planner*. 2024. arXiv: 2309.12089 [cs.RO]. URL: https://arxiv.org/abs/2309.12089.

[108] L. Sun, D. K. Jha, C. Hori, S. Jain, R. Corcodel, X. Zhu, M. Tomizuka, and D. Romeres. *Interactive Planning Using Large Language Models for Partially Observable Robotics Tasks*. 2023. arXiv: 2312.06876 [cs.RO]. URL: https://arxiv.org/abs/2312.06876.