2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

NARRATE: Versatile Language Architecture for Optimal Control in Robotics

Seif Ismail*, Antonio Arbues*, Ryan Cotterell, René Zurbrügg, Carmen Amo Alonso

***Abstract*— The impressive capabilities of Large Language Models (LLMs) have led to various efforts in enabling robots to be controlled through natural language instructions, opening exciting possibilities for human-robot interaction. The goal is for the motor-control task to be performed accurately, efficiently and safely while also enjoying the flexibility imparted by LLMs to specify and adjust the task through natural language. In this work, we demonstrate how a careful layering of an LLM in combination with a Model Predictive Control (MPC) formulation allows for accurate and flexible robotic control via natural language while taking into consideration safety constraints. In particular, we rely on the LLM to effectively frame constraints and objective functions as mathematical expressions, which are later used in the motor-control module via MPC. The transparency of the optimization formulation allows for interpretability of the task and enables adjustments through human feedback. We demonstrate the validity of our method through extensive experiments on long-horizon reasoning, contact-rich, and multi-object interaction tasks. Our evaluations show that NARRATE outperforms current existing methods on these benchmarks and effectively transfers to the real world on two different embodiments.**
Videos, Code and Prompts at narrate-mpc.github.io

I. INTRODUCTION

Efficiently translating high-level language instructions into precise low-level actions for robots and autonomous agents presents an important challenge in both artificial intelligence and robotics. While today's robots excel at following precise, low-level instructions (such as joint commands, end-effector poses, or cartesian trajectories), most systems require them to be pre-programmed for each specific task. This limits the potential for seamless integration of robots into daily life or their application to diverse and complex tasks. Natural language instructions offer a solution, as they allow humans to describe tasks in a familiar and flexible way without the need for intricate and expert programming.

Large language models (LLMs) have exhibited potential beyond natural language [1], [2], [3]. Their ability to extract semantics and generate plans according to the context of their instructions makes them well-suited for control from natural language, which is often underspecified and relies heavily on the given context. Yet, grounding their output in low-level actions remains an active area of research [4], [5]. As of today, a large number of methods aim to directly predict actions conditioned on language and visual observations using self-supervised and imitation learning [6], [7], [8], [9]. While these methods have shown impressive results, they are bottlenecked by the limited amount of robot-and task-specific data available. Moreover, their end-to-end architecture hinders interpretability and does not allow for insights into their decision-making.

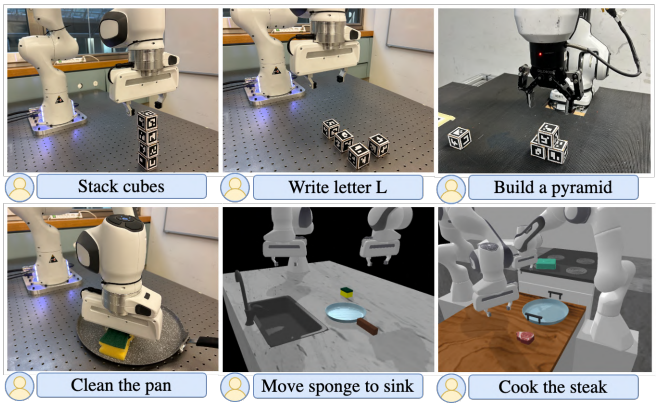


Fig. 1: **Benchmark Tasks:** Given an instruction in natural language, the robotic manipulator is able to autonomously perform a variety of different tasks, both in simulation and on different real robots.

Recent work has tried to overcome the limited availability of robotics training data by using generative models to synthesize artificial datasets [10] or by asking pretrained LLMs to directly output low-level actions [11]. Another commonly adopted alternative utilizes LLMs to output pre-defined expressions, such as motion- [12] or reward-function primitives [13]. However, restricting the network's output to the form of primitives generally limits it to a fixed library of skills, losing some of the adaptability that was gained from the introduction of LLMs.

To make full use of the versatility of LLMs, we introduce NARRATE (Natural language Architecture for Robot Control and Adaptive Task Execution), a modular architecture that interfaces language models with optimization-based controllers. Our main contribution lies in its novel integration of natural language instructions with classical robot control paradigms via the use of LLMs to predict free-form mathematical expressions that are directly used with Model Predictive Control (MPC) [14]. This allows us to surpass methods using pre-defined primitives, despite both utilizing prompts with comparable information. Our method's superiority stems from LLMs' ability to extrapolate general formulations as for instance with time-dependent functions and the seamless integration of constraints into the policy, showcasing the method's enhanced generality and flexibility. To the best of our knowledge, this is the first work to incorporate hard constraints into robotic policies prompted by LLMs, yielding significant improvements in the performance and efficiency of closed-loop controllers.

*Equal Contribution. The authors are with ETH Zürich. This research was partially supported by the ETH AI Center.

Constraints prove to be not merely beneficial, but essential for multiple aspects of robotic operation. They are paramount for ensuring safety, critical for the precise and reliable execution of tasks, and instrumental in optimizing overall system performance. As we will demonstrate in subsequent sections, the integration of constraints substantially enhances robotic control, enabling remarkable levels of accuracy, reliability, and operational effectiveness across various tasks and environments. In this way, our approach takes full advantage of the flexibility that LLMs offer while also ensuring accuracy and providing higher safety during task performance. Moreover, the relatively low latency required for LLM inference and the real-time properties of MPC allow for a very fast and intuitive interface between human users and NARRATE. This accommodates the inclusion of human feedback in natural language which can sensibly increase the system's adaptability and performance.

Our system is evaluated on a wide variety of tasks, ranging from long-horizon planning to multi-robot interactions. We further deploy NARRATE on two distinct robot manipulators and qualitatively evaluate our method on one of them, showcasing how effectively and seamlessly it transfers to the real world across multiple embodiments. In summary, our contributions are:

1) We introduce a new method to formulate general MPC policies from high-level natural language instructions for complex manipulation tasks.
2) We extensively evaluate NARRATE in simulation and hardware across different embodiments.
3) We show how NARRATE accommodates the natural interaction between humans and LLMs and how this collaboration improves the system performance.

II. Related Work

In **language-to-action** approaches, language instructions are directly mapped to low-level control actions [15], [16], [17]. In order to ensure good performance, this approach requires a significant amount of training data, either as offline datasets [15] or online environment interaction [16], [17]. Furthermore, the scalability of this approach to robot collaboration settings is limited. For instance, Zhao et al. [18] require extensive *a posteriori* sanity checks and often recomputation in order to find a suitable trajectory that avoids collisions among agents, which significantly hinders the efficiency of this approach.

In a **language-to-code** approach, language instructions are mapped into executable code [19]. This approach capitalizes on the ability of LLMs to write code that better captures the expressivity of natural language in a few-shot manner without specific training. It has proven successful for a variety of tasks [20], [21]. For instance, in the *Code-as-Policies* approach [12], an LLM is used to receive instructions in natural language and produce Python code encoding robot policies. This idea has further been tested in AI-embodied agents for navigation in virtual environments [22]. However, the effectiveness of these techniques is often hindered by the fact that code generally provides a very unstructured setting for robot control, together with the limitations that LLMs exhibit at reliable code writing [23], which can lead to consistency or even safety issues. To overcome this limitation, these approaches often use human-engineered control primitives, which in turn heavily restrict the flexibility of capturing nuances in diverse settings.

In order to overcome these limitations, **language-to-reward** approaches have been proposed [24]. In this framework, domain-specific models that map language instructions to reward functions are used. Recent work also tries to extract implicit reward functions via specific representations, such as 3D composable maps, that can be generated with code written by pre-trained LLMs [22]. An advantage of this approach is that these rewards can be used in optimization-based controllers, such as Model Predictive Control, to generate optimal control trajectories [13]. Since human instructions can often be formulated as an objective function to be optimized under some conditions (constraints), optimal control problems seem to be a good interface for translating human language into robot actions [25]. However, when only rewards are modeled, the space of possible actions has to be restricted in order to ensure safety and maintain acceptable success rates (accuracy-robustness tradeoff). In particular, the work in [13] restricts the possible outputs of the LLM to a set of functions and parameters from a predefined library. Moreover, the framework in [13] does not allow for constraint inclusion, which has been shown to boost performance and increase system safety [26].

Reward functions generated by LLMs have also been used to train Reinforcement Learning policies in order not to rely explicitly on the LLM for motor-control tasks [27]. Despite these policies being better suited to deal with the accuracy-robustness tradeoff, they still do not allow for the incorporation of constraints and introduce significant delays due to their training time. Furthermore, the successful deployment of these policies in the real world is particularly jeopardized by the sim-to-real gap.

III. Problem Statement

Consider a discrete-time system at time t with dynamics

$$x(t+1) = f(u(t), x(t)), \tag{1}$$

where $x(t) \in \mathbb{R}^{N_x}$ is the state of the system and $u(t) \in \mathbb{R}^{N_u}$ is the control input.

We introduce the *natural language control* problem. Here, the control input $u(t)$ is a function of a natural language string $\ell(t)$ given at time t, as well as the current state of the system $x(t)$. Hence, we study the problem of designing K such that

$$u(t) = K(x(t), \ell(t), t), \tag{2}$$

where $\ell(t) \in \Sigma^*$ for all t represents the string of natural language instructions over an alphabet[1] Σ and

[1]An alphabet is a finite, non-empty set. The Kleene closure of an alphabet Σ^* is defined as the following infinite union $\bigcup_{n=0}^{\infty} \Sigma^n$ where Σ^n is the n-times Cartesian product, i.e., all strings of length n.

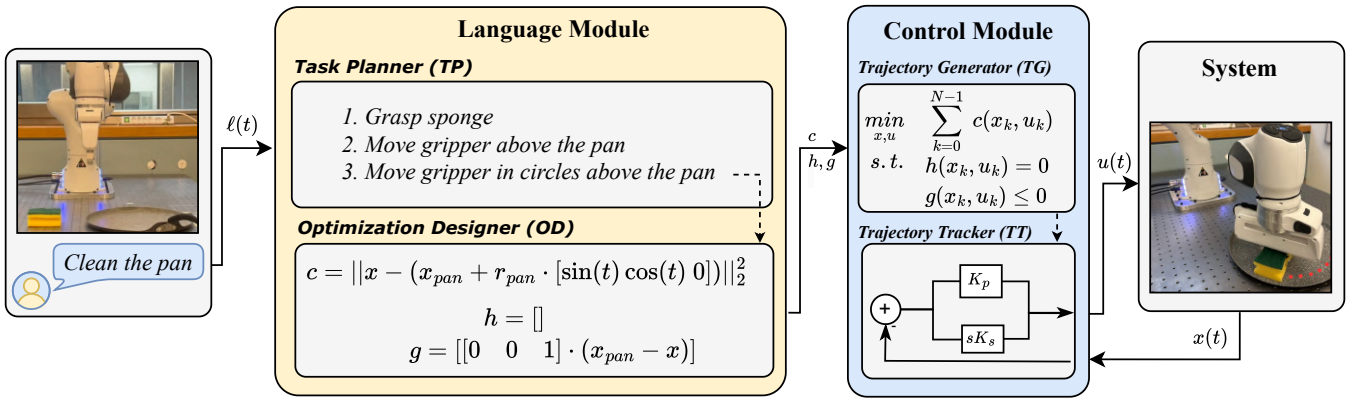


Fig. 2: **Proposed Architecture:** Overall architecture for the control of a robot manipulator via natural language. The user provides a task in natural language ℓ, which then gets translated into a series of steps (TP) and objective and constraints (OD) via two layered blocks of a language model in the language module. The objective and constraints c, h, g are then used as inputs to the control module, which generates a trajectory via MPC (TG) and the low-level control commands (TT) to be applied to the system, in this case, a robotic arm.

$K : \Sigma^* \times \mathbb{R}^{N_x} \times \mathbb{R} \to \mathbb{R}^{N_u}$. Notice that a time-varying control policy is allowed in this setup.

Since the natural language control problem formulation posed in problem (2) is broad, we restrict ourselves to the optimal control setting where we can restrict the set of safe trajectories via hard constraints. To that end, we introduce the following assumption.

Natural language MPC-based control: We assume that the natural language commands $\ell(t)$ given to the robot encode a goal that can be achieved via an optimal control problem. In particular, we assume that the goal of the controller K in equation (2) is to design a trajectory $(x_1, \ldots, x_N; u_1, \ldots, u_{N-1})$ that optimizes an objective function (cost function), while also being subject to $n_g + n_h$ hard constraints on the state and the input. Hence, the expression for K at time t is implicitly encoded through the MPC subroutine

$$\min_{\substack{x_1,\ldots,x_N \\ u_1,\ldots,u_{N-1}}} \quad \sum_{k=0}^{N-1} c(x_k, u_k; \ell(t)) \tag{3a}$$

$$s.t. \quad x_{k+1} = f(x_k, u_k) \tag{3b}$$

$$h(x_k, u_k; \ell(t)) = 0 \tag{3c}$$

$$g(x_k, u_k; \ell(t)) \leq 0 \tag{3d}$$

$$x_0 = x(t) \tag{3e}$$

Note the cost function $c(x_k, u_k; \ell(t)) \in \mathbb{R}$, the constraints $h(x_k, u_k; \ell(t)) \in \mathbb{R}^{n_h}$ and $g(x_k, u_k; \ell(t)) \in \mathbb{R}^{n_g}$ are specified via natural language through the string $\ell(t)$.

The goal of this paper is to create a system's architecture that captures the simplicity and flexibility of natural language instructions while facilitating a concrete and safe execution of the predefined task. To do so, we leverage pre-trained models in natural language as well as optimal control techniques and study how to best interface these systems to achieve such goals.

IV. Language to Control

To solve the *natural language control* problem, we make use of the capabilities of a pre-trained LLM (GPT4 [23]) and well-established optimal control techniques (MPC and impedance control). Our architecture consists of a planning and control module, each of them containing two blocks. An overview of the architectural components can be seen in Figure 2. We note that this architecture is fairly general and could be applied to a variety of environments where control actions are needed to steer the behavior of a dynamic system. Environment information is provided via prompting, and while we do not implement a general perception module in this work, our implementation on real robot manipulators demonstrates robust performance even in the face of disturbances and imperfect perception. Our method incorporates scene descriptions (including the size and location of available objects) by appending this information to each user instruction. Information about the robot and its current state is embedded in the system prompt of the Language Module at the episode's outset.

A. Language Module

The Language Module, $L : \Sigma^* \to C^1 \times \cdots \times C^1$,[2] is designed to receive a language utterance $\ell(t)$ as input at each time step t, and generate the cost and constraints functions c, h and g as an output, i.e.,

$$(c, \underbrace{h_1, \ldots, h_{n_h}}_{h}, \underbrace{g_1, \ldots, g_{n_g}}_{g}) = L(\ell(t)). \tag{4}$$

The goal of this module is to convert instructions given in natural language into an output that can readily be used by MPC controllers, which are frequently used for controlling dynamic systems, such as robots, in common practice. Since human instructions often lack specificity and can be ambiguous, we divide this task into two different steps. In fact, distributing one comprehensive instruction across multiple precisely prompted LLMs has been shown to significantly increase performance compared to offloading it entirely on one model prompted generally [22]. Hence, the Language

[2]Due to the lack of explainability of current language models, we cannot guarantee that the output of map L will sit in $C^1 \times \cdots \times C^1$. In those cases, task execution will fail.

Module in the proposed architecture consists of two layered blocks: the Task Planner and the Optimization Designer.

Task Planner (TP): The Task Planner block receives the user instructions as the input and generates a plan of actions that the system is required to execute. In particular, we use a pre-trained LLM as a decoder to convert the user message into a list of specific subtasks formulated in natural language that need to be carried out in order to fulfill the original task. To do this, the language model is prompted with information about the system to be controlled (i.e., a robotic arm, etc.), the environment (i.e., the presence of different objects), and general guidelines on how to generate the sequence of subtasks (few-shot examples), which are crucial to ground the LLM responses. We refer the reader to our project website[3] for implementation details, exact prompts and benchmark environments.

Optimization Designer (OD): The Optimization Designer block consists of an LLM that receives a given subtask from TP and generates appropriate objective and constraint functions c, h, g. In this block, we sequentially use each of the subtasks and return an optimization formulation in mathematical terms as an output. To do this, we rely on the ability of pre-trained models to code optimization functions symbolically using CasADi [28] (a non-linear optimization framework) as shown in a representative example in Figure 4. This provides our system with a greater versatility as compared to prior works, where the language model is often constrained to use predefined parametrized functions [13]. Besides symbolic functions in CasADi, the language model also has been instructed in the availability of separate functions specific for the task, i.e., in the case of robot control: `open_gripper()`,`close_gripper()`.

In the cube stacking example, as introduced in §V-A, the *OD* block is able to formulate objective and constraints that ensure an optimal trajectory while avoiding collisions with other cubes. In this case, we define the state $x \in \mathbb{R}^4$ as the 3D position and orientation ψ around the z-axis of the robot gripper, and the control inputs $u \in \mathbb{R}^4$ as the time derivative of the states ($u = \dot{x}$). This choice is conveyed to the *OD* block through an instruction in natural language.

As an example, the natural language utterance $\ell =$"*move on top of the blue cube avoiding collisions with red and green cubes*" gets translated into the following objective function

$$c(\ell, x.u) = \|\text{diag(1,1,1,0)}(x - x_{\text{blue}} + [0, 0, d])\|_2^2 \quad (5)$$

and the following collision-avoidance constraints

$$g_1(\ell, x, u) = d_{\text{min}} - \|\text{diag(1,1,1,0)}(x - x_{\text{red}})\|_2 \leq 0 \quad (6a)$$

$$g_2(\ell, x, u) = d_{\text{min}} - \|\text{diag(1,1,1,0)}(x - x_{\text{green}})\|_2 \leq 0 \quad (6b)$$

where $x_{\text{blue}}, x_{\text{red}}, x_{\text{green}} \in \mathbb{R}^4$ are position and orientation of the corresponding cubes, $d \in \mathbb{R}$ is the side length of the cubes, and d_{min} is the minimum distance to avoid collisions, in this example, $d_{min} \stackrel{\text{def}}{=} d/2$.

This process of generating a mathematical representation given a concrete sub-task expressed in natural language is repeated until all sub-tasks are carried out. Proceeding to the next sub-task occurs if one of three conditions is satisfied:

$$J(t) \leq \epsilon_1, \quad \Delta J(t) \leq \epsilon_2, \quad t - t_0 \geq t_{max} \quad (7)$$

Where $J(t)$ represents the value of the MPC objective at the current time step and $\Delta J(t)$ represents its difference from one step to the next. t_0 represents the first time step where the new MPC formulation has been applied and $\epsilon_1, \epsilon_2, t_{max}$ represents the limit thresholds that are to be tuned.

B. Control Module

The control module, $C : \mathbb{R}^{N_x} \times C^1 \times \cdots \times C^1 \to \mathbb{R}^{N_u}$, receives the objective function and constraints c, h and g together with input the system state at each time t as $x(t)$, and generates an optimal control signal u as an output, i.e.

$$u(t) = C(x(t); c, h, g). \quad (8)$$

The goal of this module is to ensure that the system (in this case the robotic arm) generates and follows an optimal trajectory while satisfying constraints. Once again, we divide this task into two different steps. This separation produces accurate high-level motions that can be tracked at high frequencies by two separate blocks descrobed below.

Trajectory Generator (TG): The Trajectory Generator block consists of an MPC controller that receives the objective function c and constraints h, g from the *OD* and solves for an optimal trajectory, i.e., sequence of states and inputs x_n and u_n for $n = 1, \ldots, N$ where N is some predetermined time horizon. The TG solves the MPC subroutine (3) at every time step t for the new measurement $x(t)$ using a predefined system model (3b). Usually, a simplified model is used at the TG level to reduce latency and modeling complexity. In this way, it is still possible to generate trajectories that minimize the objective function while also respecting the constraints received from the LLM without adding modeling overhead at the planning level. The separation of TG and TT specifically allows us to deal with this issue by relying on a much simpler approximation of the robot manipulator system, i.e.

$$x(t+1) = Ax(t) + Bu(t), \quad (9)$$

where the diagonal matrices $A = I_4$ and $B = I_4 \cdot \Delta t$ represent the kinematics matrices of a point mass in 3D position and rotation around the z-axis of the end-effector with I and Δt representing the identity matrix and discretization time respectively. Additions to the MPC, such as regularization terms and pre-defined constraints, can be appended in an additive manner to the objective function and constraints received from the OD. We introduce safety constraints on the end-effector state $x \in \mathbb{R}^4$ to be within a feasible set (i.e. above the table surface) as

$$g_1(x, u) = \begin{bmatrix} 0 & 0 & -1 & 0 \end{bmatrix} x \leq 0. \quad (10)$$

We also introduce safety constraints to bound the gripper velocity $u \in \mathbb{R}$ in a similar fashion to (10). Hence, the TG for all tasks described in V-A solves subroutine (3), where the dynamics (3b) are replaced by expression (9), and constraint (10) is included as part of the formulation (3d).

[3] `https://narrate-mpc.github.io`

The choice of MPC for the TG block is motivated by the simplicity of its formulation, together with the computation efficiency and the ability to provide guarantees on the generated trajectory. While similar approaches make use of the generated objective (reward) function to train learning-based controllers [29], [27], they require the controller to be trained for each task independently, which severely limits their generalizability and makes it challenging to use them in a real-world setting. The ability of LLMs to account for human feedback alongside the real-time regime of MPC, on the other hand, enables interactive collaboration between human users and NARRATE, which can further improve task performance as shown in §VI.

Trajectory Tracker (TT): The Trajectory Tracker block is the lowest-level block of the architecture stack and aims to follow the trajectory generated by the TG as optimally as possible. Its purpose is to map the trajectories obtained from TT into specific low-level actions for the given hardware, i.e., torque inputs to be applied to the motors of a robot, etc. Once the TT receives the optimal trajectory $\left[x_1^*, \ldots, x_N^*, u_1^*, \ldots, u_{N-1}^*\right]$ from the TG, it computes the desired acceleration of the gripper w via a cartesian impedance controller:

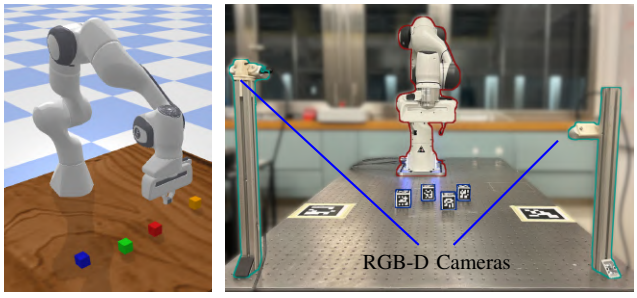
$$\dot{w}_k^* = k_p(x_k^* - x(k)) + k_d(u_k^* - \dot{x}(k)), \tag{11a}$$

$$\tau_k = J_e^T(\Lambda \dot{w}_k^* + \mu + r) \tag{11b}$$

where k_p and k_d represent the gains of the PD controller, respectively, and $x(k), \dot{x}(k), x_k^*, u_k^*$ represent the measured and desired gripper position and velocity at time k, respectively. J_e, Λ, μ, r are obtained from the dynamics model as shown in [30].

V. Experimental Setup

We extensively evaluate our method in custom simulation environments using PandaGym [32] and on two real-world platforms consisting of the Franka Emika Panda and an in-house built manipulator (DynaArm).



(a) Simulation [32] (b) Hardware

Fig. 3: **Evaluation setup** We use PandaGym [32] to build our custom simulation environments. For the quantitative, real-world evaluation, we use a Franka Emika Panda with two RGB-D cameras (D435i, D455) and ArUco markers to extract object poses.

A. Tasks

We evaluate our method on a total of six different tasks with varying difficulties. The single-robot tasks are tested both in simulation as well as on the real world. An overview of the different tasks is depicted in Figure 1.

```
TP: move both grippers above the `stove` and keep them at a constant distance between each other.
OD:{"objective": "ca.norm_2(x_left[:3]-(stove.x[:3]+[0, 0, 0.1]))**2 + ca.norm_2(x_rig[:3]ht-(stove[:3]+[0, 0, 0.1]))**2",
"equality_constraints": ["ca.norm_2(x0_left[:3] - x0_right[:3]) - ca.norm_2(x_left[:3] - x_right[:3])"],
"inequality_constraints": []}
```

Fig. 4: **TP-OD responses for the *Cook Steak* task**: the robots have to collaborate in order to move the frying pan above the stove using its two handles. This OD response requires the robot grippers to remain at a constant distance to prevent the pan from falling.

Stack, Pyramid, L: In these tasks, the robot is asked to re-arrange four cubes according to a specific pattern. All cubes are initially placed on a flat surface, and the state of the cubes and robots are known throughout the full experiment. The task is solved successfully if all 4 cubes are stacked on top of each other, form a pyramid with 2 predefined cubes at the base and one at the top or are correctly re-arranged to form an L shape flat on the table. Illustrative examples of the queries used for these tasks are: "*stack all cubes on top of the blue one*", "*build a pyramid with red and blue cube at the base*" and "*write the letter L flat on the table*".

Clean: A sponge and a pan are located on a table, and the robot is instructed to clean the pan. The task is considered to be solved successfully if the robot performs a continuous motion (i.e. circles, back and forth,...) and the sponge is in contact with the plate. The query used for this task is "*clean the pan with the sponge*".

Move Wet: A wet sponge and a pan are located on a table that also features a sink. The task is formulated as a collaborative task, where two robots must coordinate and move the wet sponge to the sink without dropping water. The task is solved successfully if the sponge is moved to the sink while the pan is held beneath it throughout the trajectory to avoid water from dropping. This task can be solved by one robot carrying the pan while the other carrying the sponge above the pan. It is also accepted if the sponge is directly dropped into the pan first and then moved together to the sink. The query used for this task is "*move the sponge to the sink with the left robot but since it's wet make sure not to drop water using the pan*".

Cook Steak: A frying pan with two handles and a steak are placed on the table. Again, two robots are present, and the task needs to be solved collaboratively. The task is considered successful if the frying pan is moved on top of the burner plate and, subsequently, the steak is placed into the pan. The query used for this task is "*cook the steak*".

B. Simulation Setup

For each task, both the TP and OD are provided with general system prompts that remain unchanged across the single-robot tasks and across the collaborative tasks, showing the versatility of NARRATE to generalize in different

Method	Stack		L-Shape		Pyramid		Clean Plate		Move Wet		Cook Steak	
	$SR_{\%}\uparrow$	$Co/Pl/Cd_{\%}\downarrow$	$SR_{\%}\uparrow$	$Co/Pl/Cd_{\%}\downarrow$	$SR_{\%}\uparrow$	$Co/Pl/Cd_{\%}\downarrow$	$SR_{\%}\uparrow$	$Co/Pl/Cd_{\%}\downarrow$	$SR_{\%}\uparrow$	$Co/Pl/Cd_{\%}\downarrow$	$SR_{\%}\uparrow$	$Co/Pl/Cd_{\%}\downarrow$
CaP [12]	**98**	0 / 2 / 0	10	60 / 16 / 14	16	2 / 76 / 6	22	10 / 48 / 20	4	36 / 40 / 20	0	20 / 48 / 32
VoxPoser [31]	26	48 / 0 / 26	0	76 / 0 / 24	16	38 / 22 / 24	48	6 / 22 / 24	6	10 / 24 / 60	0	0 / 74 / 26
NARRATE†	50	48 / 2 / 0	24	66 / 10 / 0	32	58 / 10 / 0	**64**	10 / 26 / 0	64	18 / 18 / 0	0	50 / 50 / 0
NARRATE	92	2 / 6 / 0	**58**	12 / 30 / 0	**76**	2 / 22 / 0	62	4 / 34 / 0	**68**	16 / 16 / 0	**30**	30 / 40 / 0

TABLE I: **Simulation Results:** Comparison of NARRATE against *CaP* and *VoxPoser* on the six simulation tasks as illustrated in Figure 1. We report the success rates for each method alongside the failure reason categorized into Collision (*Co*), Planning (*Pl*) and Code Execution (*Cd*) errors. † represents the version without constraints.

settings. Moreover, a user message that specifies the actual task is provided in the form of an utterance alongside a list of objects the robot can interact with. The OD uses symbolic variables to represent the robot states and the objects in the scene. These are converted into their corresponding values when the MPC is initialized at each times step.

C. Hardware setup

We evaluate our approach with real-world experiments using a Franka Emika Panda Robotic arm and two external realsense cameras, depicted in Figure 3b. We evaluate our method on a subset of the previously introduced task, namely *Stack*, *L*, *Pyramid*, and *Clean Plate* and rely on two external RGB-D cameras and AruCo markers to extract the poses of the cubes. Additionally, we demonstrate the embodiment transfer on a custom, in-house built manipulator (Dynaarm) and that our modular structure allows us to easily switch out the Trajectory Tracker to account for different dynamics.

VI. Results

A. Simulation Results

For each of the tasks introduced in §V-A, we evaluate our method and report the success rate as well as the failure reason categorized into *Collision*, *Planning* and *Code Execution* errors. A *Collision* failure occurs if the run was not successful due to a collision, i.e. the gripper knocks down the stack of cubes when trying to place the next cube on top of it. A *Planning* failure occurs if the task instruction is not satisfied at the end of the run, i.e. the steak is placed into the pan but the pan is not moved on top of the burner plate for *Cook Steak*. Finally, a *Code Execution* failure occurs if the LLM response raises a code exception when being evaluated. It is important to note that failures are not exclusive, meaning that a failure related to *Code Execution* does not imply that the run would not have had *Planning* or *Collision* errors.

We evaluate our architecture over a total of 50 runs for each task and compare against current state-of-the-art methods such as *Code-as-Policies* (*CaP*) [12] and *VoxPoser* [31]. We implement conservative pick and place and motion primitives for *CaP*, while for *VoxPoser*, we programmatically derive the objects' point cloud given their known geometry and pose. The final comparisons are shown in Table I and we refer the reader to our project page for additional demos. We also evaluate NARRATE without the incorporation of constraints, where the OD only generates the objective function of the MPC formulation. This version falls short both in success rate and the number of collisions, highlighting the importance of incorporating constraints into the controller formulation. This is particularly evident for the *Cook Steak* task, where constraints are required for the robots to collaboratively move the pan without dropping it. The OD formulation for this specific sub-task, as seen in Figure 4, enforces that the distance between the grippers remains constant. We highlight that our method achieves the highest success rates in five out of six tasks while showing that the adoption of constraints not only significantly reduces the number of collisions but also increases performance.

B. Efficiency Evaluations

We further evaluate the efficiency of our method by comparing the average task execution time and distance covered by the end-effector during *successful* runs, as found in Table II. The methods that include collision avoidance are generally more efficient in terms of distance covered by the end-effector. In fact, both *CaP* and NARRATE without constraints need to be prompted such that the robot first has to go on top of an object instead of directly going to its location in order to grasp it. While *VoxPoser* is relatively efficient in terms of distance covered by the end-effector, thanks to the use of collision maps, its trajectory generator (a sampling-based MPC) requires significant computational efforts leading to high execution times compared to the other methods. NARRATE achieves the best results both in terms of time and distance efficiency thanks to its simple formulation and the ability to use constraints for collision avoidance.

Method	(Cubes + Clean)		Wet		Steak	
	$Time_{[s]}\downarrow$	$Dist_{[m]}\downarrow$	$Time_{[s]}\downarrow$	$Dist_{[m]}\downarrow$	$Time_{[s]}\downarrow$	$Dist_{[m]}\downarrow$
CaP [12]	$35_{\pm 3}$	$2.2_{\pm .1}$	$188_{\pm 24}$	$2.6_{\pm 0.5}$	–	–
VoxPoser [31]	$409_{\pm 89}$	$2.1_{\pm .2}$	$203_{\pm 17}$	$1.6_{\pm 0.1}$	–	–
NARRATE†	$38_{\pm 6}$	$2.5_{\pm 0.2}$	$91_{\pm 11}$	$1.3_{\pm 0.2}$	–	–
NARRATE	$\mathbf{23}_{\pm \mathbf{2}}$	$\mathbf{1.7}_{\pm \mathbf{0.2}}$	$\mathbf{65}_{\pm \mathbf{7}}$	$\mathbf{1.2}_{\pm \mathbf{0.1}}$	$91_{\pm 13}$	$2.0_{\pm 0.2}$

TABLE II: **Efficiency Evaluation:** Comparison of the efficiency of NARRATE against *CaP* and *VoxPoser* as defined by the average task execution time and end-effector covered distance for *succesful* runs. We average the efficiencies for the single-robot tasks and keep the others separated. † represents the version without constraints.

C. Continual Learning and Feedback

We further investigate the ability to integrate feedback into the optimization design. If an action is not carried out correctly, the user can instruct the LLM in natural language on how to readjust for the error. In Figure 6b, we present comparisons of the performance of the LLM at various levels of human intervention for the given tasks. We evaluate the success rate of the system for the six tasks in three different settings: in case of no human intervention, in the setting where the human is allowed to provide one feedback message

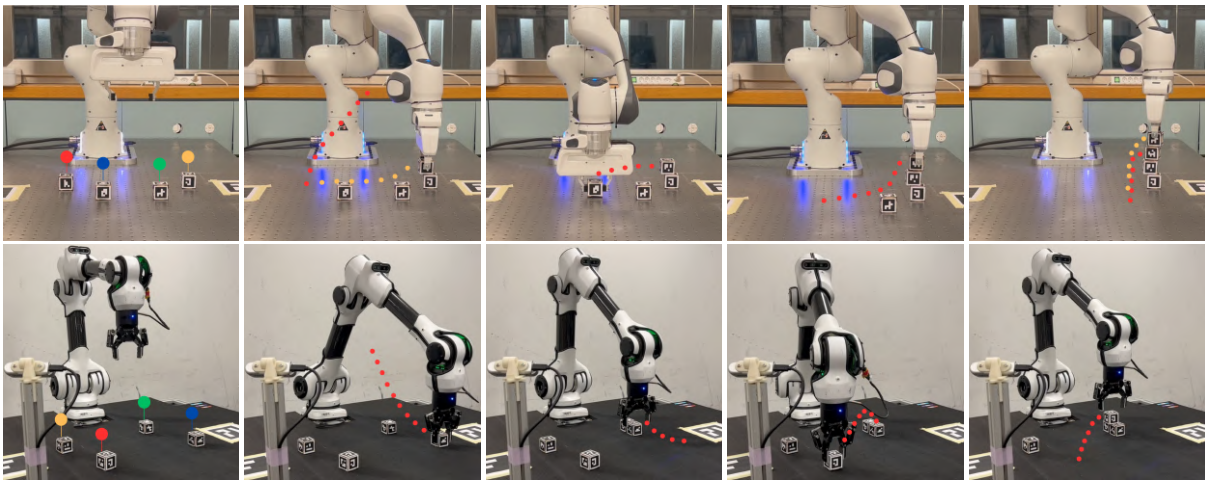


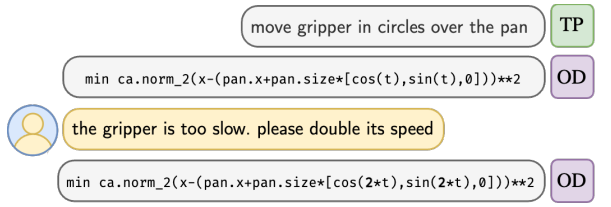
Fig. 5: **Real World Examples:** Visualization of two real world evaluations. The respective queries for the tasks were given as (a) "*make a stack of cubes on top of the yellow cube*" and (b) "*build a pyramid with the green and blue cubes at the base and red cube at the top. keep the green cube at its original position*". The Aruco markers encode the color of each cube.

after the TP has generated a plan, and finally, where the human substitutes the TP. Success rates, where the human acts as the TP, were evaluated as the average of two different (expert) people carrying out five runs each.

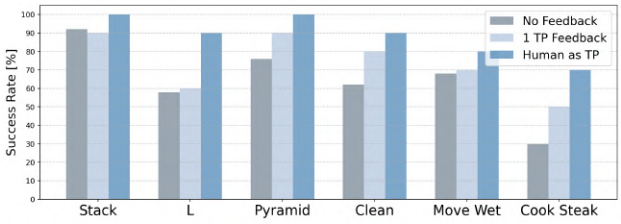
Since most failures are attributed to an incoherent plan formulated by the TP, the addition of feedback leads to tangible improvements. In fact, the best performance is achieved when the human prompts the *OD* one task at a time. As illustrated in Figure 6a, the intuitiveness and versatility of the designed interface leads to an easy way of incorporating feedback.

D. Real World Deployment

We finally evaluate the single-robot tasks (*Stack*, *Pyramid*, *L*, and *Clean Plate*) on a Franka robot, highlighting that our method can directly be deployed on hardware. We again compare the success rates for *VoxPoser*, *CaP*, and NARRATE as reported in Table III. The experiments confirm the validity of the method in terms of success rate, with an increased collision rate compared to simulation. Often times the collisions are mild, and we attribute them to the imperfect perception system, which has approximately 1cm maximum slack in pose estimation accuracy. In Figure 5, we additionally demonstrate the versatility and generalizability of our method by deploying it on an additional manipulator (DynaaArm), which differs from the one used in the previous experiments, showcasing that our method can easily be used with different embodiments, thanks to its modular design.



(a) Human-LLM interactive behavior



(b) Task Performance with Feedback

Fig. 6: **Effects of adding feedback:** (a) Illustrates the interaction between a human and the LLM. The user asks to double the gripper's speed which leads to the OD multiplying the time by two in the objective function. (b) Success rate comparison with no human feedback, with one feedback message sent to the TP after the plan is generated, and when an expert human acts as the TP and provides each sub-task instruction to the OD.

Method	Stack SR%↑	Stack Co/Pl/Cd%↓	L-Shape SR%↑	L-Shape Co/Pl/Cd%↓
CaP [12]	**70**	30 / 0 / 0	10	10 / 40 / 40
VoxPoser [31]	10	90 / 0 / 0	20	60 / 0 / 20
NARRATE	**70**	30 / 0 / 0	**70**	0 / 30 / 0
	Pyramid		Clean Plate	
CaP [12]	10	0 / 70 / 20	70	0 / 20 / 10
VoxPoser [31]	10	80 / 0 / 10	70	0 / 0 / 30
NARRATE	**90**	0 / 10 / 0	**90**	0 / 10 / 0

TABLE III: **Real World Experiments:** Comparison of NARRATE against *CaP* and *VoxPoser* on four single-robot tasks in the real world as seen in Figure 1, 10 repetitions per experiment. We report the success rates for each method and the failure reason categorized into Collision (*Co*), Planning (*Pl*) and Code Execution (*Cd*) errors.

E. Limitations

While NARRATE demonstrates promising results, several challenges remain. The approach's generalization to tasks beyond point-to-point navigation, object grasping and obstacle avoidance is limited, as it requires specific prompt engineering to ground the language model to each task (e.g., specifying the location for cleaning a pan). This reliance on task-specific prompts highlights a lack of robust methodology for prompt design, which currently does not have a systematic, empirical approach [33]. Furthermore, NARRATE inherits

the inherent shortcomings of LLMs, including potential code execution and planning errors. Although safety constraints are incorporated, there is no formal verification process for these constraints or the generated code execution, leaving room for potential safety issues. The absence of a perception module limits adaptability to diverse environments where existing objects and their properties are not known a priori. These factors collectively impact the method's performance and scalability, presenting avenues for future improvement.

VII. Conclusion and Future Work

In this work, we propose an architecture to interface natural language instructions with low-level system actions via a language pre-trained module and an optimal control module. By using optimal control, we are able to impose constraints on the actions of the system, which increases its safety and reliability. In this way, we can take advantage of the flexibility of natural language instructions, since we do not need pre-defined functions nor expensive reinforcement learning iterations to ensure good system behavior. We illustrate in simulation and on real robots, the effectiveness of our approach as compared to other strategies. We show that LLMs are suitable for writing control policies as free-form mathematical expressions, which allow for versatile control policies that can solve a set of complex tasks with higher success rates and higher efficiency compared to existing methods. For future work, we aim to create guarantees for the safety of both the language module as well as the control module. Additionally, we plan to tightly integrate visual observations and introduce feedback directly to the Language Module.

References

[1] K. Hausman, B. Ichter, S. Levine, A. Toshev, F. Xia, and C. Parada, "Natural language control of a robot," Oct. 5 2023, uS Patent App. 18/128,953.

[2] D. Shah, M. Equi, B. Osinski, F. Xia, B. Ichter, and S. Levine, "Navigation with large language models: Semantic guesswork as a heuristic for planning," *arXiv preprint arXiv:2310.10103*, 2023.

[3] B. M. Lake and M. Baroni, "Human-like systematic generalization through a meta-learning neural network," *Nature*, pp. 1–7, 2023.

[4] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE ICRA*. IEEE, 2023, pp. 11 523–11 530.

[5] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *Microsoft Auton. Syst. Robot. Res*, vol. 2, p. 20, 2023.

[6] A. B. et al., "Rt-2: Vision-language-action models transfer web knowledge to robotic control," 2023.

[7] O. X.-E. Collaboration, "Open X-Embodiment: Robotic learning datasets and RT-X models," https://arxiv.org/abs/2310.08864, 2023.

[8] K. B. et al., "Robocat: A self-improving generalist agent for robotic manipulation," 2023.

[9] Z. Fu, T. Z. Zhao, and C. Finn, "Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation," 2024.

[10] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, Z. Erickson, D. Held, and C. Gan, "Robogen: Towards unleashing infinite data for automated robot learning via generative simulation," 2023.

[11] Y.-J. Wang, B. Zhang, J. Chen, and K. Sreenath, "Prompt a robot to walk with large language models," 2023.

[12] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[13] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, *et al.*, "Language to rewards for robotic skill synthesis," *arXiv preprint arXiv:2306.08647*, 2023.

[14] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[15] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, "Vima: General robot manipulation with multimodal prompts," *arXiv preprint arXiv:2210.03094*, 2022.

[16] C. Wu, S. Yin, W. Qi, X. Wang, Z. Tang, and N. Duan, "Visual chatgpt: Talking, drawing and editing with visual foundation models," *arXiv preprint arXiv:2303.04671*, 2023.

[17] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conference on Robot Learning*, 2022.

[18] Z. Mandi, S. Jain, and S. Song, "Roco: Dialectic multi-robot collaboration with large language models," *arXiv preprint arXiv:2307.04738*, 2023.

[19] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, *et al.*, "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, 2021.

[20] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[21] D. Trivedi, J. Zhang, S.-H. Sun, and J. J. Lim, "Learning to synthesize programs as interpretable and generalizable policies," *Advances in neural information processing systems*, vol. 34, pp. 25 146–25 163, 2021.

[22] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv:2305.16291*, 2023.

[23] OpenAI, "Gpt-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023. [Online]. Available: https://arxiv.org/abs/2303.08774

[24] P. Goyal, S. Niekum, and R. J. Mooney, "Using natural language for reward shaping in reinforcement learning," *arXiv preprint arXiv:1903.02020*, 2019.

[25] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, 2021.

[26] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox, "Correcting robot plans with natural language feedback," *arXiv preprint arXiv:2204.05186*, 2022.

[27] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv:2310.12931*, 2023.

[28] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[29] A. Suhr and Y. Artzi, "Continual learning for instruction following from realtime feedback," *arXiv preprint arXiv:2212.09710*, 2022.

[30] H. Baruh, *Analytical Dynamics*, ser. McGraw-Hill International Editions Series. Maidenhead, England: McGraw Hill Higher Education, Oct. 1998.

[31] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," *arXiv preprint arXiv:2307.05973*, 2023.

[32] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen, "panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning," *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

[33] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering in large language models: a comprehensive review," 2023.