# Language Models as Zero-Shot Trajectory Generators

Teyun Kwon<sup>1</sup>, Norman Di Palo<sup>1</sup>, Edward Johns<sup>1</sup>

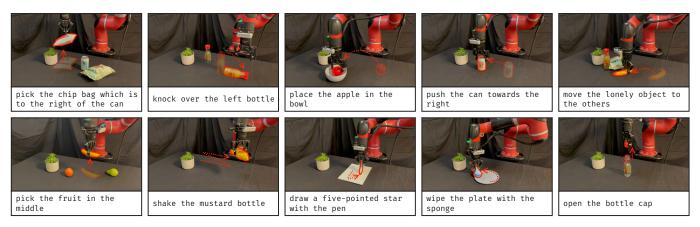


Figure 1: A selection of the tasks we use to study if a single, task-agnostic LLM prompt can generate a dense sequence of end-effector poses, when given only object detection and segmentation models, and no in-context examples, motion primitives, pre-trained skills, or external trajectory optimisers.

Abstract—Large Language Models (LLMs) have recently shown promise as high-level planners for robots when given access to a selection of low-level skills. However, it is often assumed that LLMs do not possess sufficient knowledge to be used for the low-level trajectories themselves. In this work, we address this assumption thoroughly, and investigate if an LLM (GPT-4) can directly predict a dense sequence of end-effector poses for manipulation tasks, when given access to only object detection and segmentation vision models. We designed a single, task-agnostic prompt, without any in-context examples, motion primitives, or external trajectory optimisers. Then we studied how well it can perform across 30 real-world language-based tasks, such as "open the bottle cap" and "wipe the plate with the sponge", and we investigated which design choices in this prompt are the most important. Our conclusions raise the assumed limit of LLMs for robotics, and we reveal for the first time that LLMs do indeed possess an understanding of low-level robot control sufficient for a range of common tasks, and that they can additionally detect failures and then re-plan trajectories accordingly. Videos, prompts, and code are available at: https://www.robotlearning.uk/language-models-trajectory-generators.

Index Terms—AI-Based Methods, Big Data in Robotics and Automation, Deep Learning in Grasping and Manipulation

Manuscript received: December 27, 2023; Revised April 12, 2024; Accepted May 13, 2024.

This paper was recommended for publication by Editor Aleksandra Faust upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by the Royal Academy of Engineering under the Research Fellowship Scheme.

<sup>1</sup> Teyun Kwon, Norman Di Palo and Edward Johns are with the Robot Learning Lab at Imperial College London. {john.kwon20, n.di-palo20, e.johns}@imperial.ac.uk

Digital Object Identifier (DOI): 10.1109/LRA.2024.3410155.

## I. INTRODUCTION

N recent years, Large Language Models (LLMs) have attracted significant attention and acclaim for their remarkable capabilities in reasoning about common, everyday tasks [1]. This widespread recognition has since led to efforts in the robotics community to adopt LLMs for high-level task planning [2]. However, for low-level control, existing proposals have relied on auxiliary components beyond the LLM, such as pre-trained skills, motion primitives, trajectory optimisers, and numerous language-based in-context examples (Fig. 2). Given the lack of exposure of LLMs to physical interaction data, it is often assumed that LLMs are incapable of low-level control [3], [4], [5].

However, until now, this assumption has not been thoroughly examined. In this paper, we now investigate if LLMs have sufficient understanding of low-level control to be adopted for **zero-shot dense trajectory generation for robot manipulators**, without the need for the aforementioned auxiliary components. We provide an LLM (GPT-4 [6]) with access to off-the-shelf object detection and segmentation models, and then require all remaining reasoning to be performed by the LLM itself. We also require that the same task-agnostic prompt is used for all tasks, such as "open the bottle cap" and "wipe the plate with the sponge", which we took from the recent literature. And through this investigation, we uncovered the underlying principles and strategies that empower LLMs to navigate the complexities of robot manipulation.

Consequently, our contributions are threefold: (1) We show, for the first time, that a pre-trained LLM, when provided with only an off-the-shelf object detection and segmentation model,

REQUIRES METHOD	LLM	vision model	predefined primitives	in-context examples	external trajectory optimisers	robotics- specific training data
VoxPoser	•	•		•	•	
Code as Policies	•	•	•	•		
SayCan	•	•		•		•
Language to Rewards	•	•			•	
ChatGPT for Robotics	•	•	•			
RT-2	•	•				•
Our Investigation	•	•				

Figure 2: A taxonomy of requirements of LLM-based zeroshot methods from the recent literature.

can guide zero-shot a robot manipulator by outputting a dense sequence of end-effector poses, without the need for pre-trained skills, motion primitives, trajectory optimisers, or in-context examples. (2) We present several ablation studies which shed light on what techniques and prompts lead to the emergence of these capabilities. (3) We study how, by analysing the trajectory of objects across an image, LLMs can also detect if a task has failed and subsequently re-plan an alternative trajectory.

## II. RELATED WORK

Preliminary versions of this paper were accepted at the ICRA 2024 VLMNM Workshop (Spotlight), the CoRL 2023 Workshop on Language and Robot Learning, and the NeurIPS 2023 Robot Learning Workshop (all non-archival). This work now presents our full paper, with further experiments and analysis.

While prior works have made significant strides in leveraging LLMs for various aspects of robotic control [2], several limitations and dependencies on external modules persist. The core motivation of our work is to **investigate whether these limitations are inherent, or if LLMs can be deployed for low-level control**, going from language to a dense sequence of end-effector poses. In this section, we provide an overview of the relevant literature and highlight key distinctions between prior approaches and our research focus.

**LLMs for Robotics:** There have been a number of works which leverage the common-sense knowledge and instructionfollowing capabilities of LLMs, but they have relied on external components for the full low-level trajectory generation. Both Code as Policies [7] and ChatGPT for Robotics [8] rely on predefined motion primitives for robot control, and their focus is predominantly on high-level planning. In the case of SayCan [5], robotics-specific data is required to pre-train the skills. VoxPoser [3] and Language to Rewards [4] have explored the use of LLMs to generate high-reward regions for robot movement, but these methods still necessitate external trajectory optimisers to compute a trajectory, such as cost and reward functions used to evaluate randomly sampled trajectories along with Model Predictive Control (MPC) [3]. VoxPoser [3], Code as Policies [7], and SayCan [5] have also relied heavily on providing in-context examples to the LLM input. However, these methods can encounter challenges when

extrapolating beyond the demonstrated tasks. A summary of these works and their required auxiliary components is shown in Fig. 2.

Out of the aforementioned works, Code as Policies [7] is closest to our work. However, instead of relying on predefined primitives and in-context examples, we focus our investigation on the generation of trajectories more complex than a sequence of linear interpolations and without any task-specific guidance, thus broadening the scope of applicability and adaptability in the real world and reducing the reliance on human expertise. We also show that these complex trajectories are generated from the LLMs' internal knowledge of these tasks, instead of relying solely on the use of external libraries for code generation, and that this understanding is not only beneficial for trajectory generation, but also for task success detection as well, providing insight into the capabilities of LLMs to detect and recover from failures.

Foundation Models for Robotics: Recent works [9], [10] demonstrated that a Vision Language Model (VLM) [11] can be fine-tuned with a large robotics-related dataset of actions to enable zero-shot language-conditioned control. It has also been shown that such demonstrations can be provided directly to LLMs in the form of language tokens as in-context examples [12], [13]. Other works on foundation models include Socratic Models [14], which proposes to leverage multiple pretrained models via language-based exchange, and Inner Monologue [15] which focuses on environment feedback for success detection, scene description, and human interaction. VLMs and LLMs have also been used in reinforcement learning settings for their planning and success detection capabilities, especially in long-horizon tasks [16].

Language and Visual Grounding: Although not the main focus of our work, there have been numerous methods to ground the visual information for language-conditioned robot manipulation, such as embedding CLIP features into NeRFs [17], [18], [19], and keypoint extraction for object encoding [12] and demonstration retrieval [20], [21]. DALL-E-Bot [22] introduces web-scale image diffusion models to ground the target rearrangement scene, whereas CLIP similarity scores are used in Dream2Real [23] to determine the most visually plausible target scene.

## III. PROBLEM FORMULATION

We investigate if an LLM (GPT-4 [6]) can predict a dense sequence of end-effector poses to solve a range of manipulation tasks. We now explain what the assumptions and constraints are in our investigation, followed by details of our real-world experimental setup, and the tasks used for evaluation. Given this background, we then present our investigation and its results in Sec. IV.

Assumptions and Constraints: We design a task-agnostic prompt to study the zero-shot control capabilities of LLMs, with the following assumptions: (1) no pre-existing motion primitives, policies or trajectory optimisers: the LLM should output the *full sequence of end-effector poses itself*; (2) no in-context examples: we study the ability of LLMs to reason about tasks given their *internal knowledge alone*, and no part

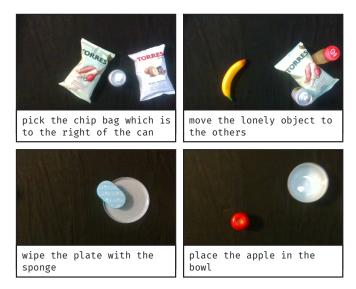


Figure 3: Example wrist-camera observations received by the robot at the start of each task, and their corresponding task instructions.

of any task is explicitly mentioned in the prompt, either in the form of examples or instructions; (3) the LLM can query a pretrained vision model to obtain information about the scene, but should *autonomously generate*, *parse and interpret the inputs and outputs*; (4) no additional pre-training or fine-tuning on robotics-specific data: we focus our research on *pre-trained and widely available models*, so that our work can easily be reproduced even with limited compute budget.

Real-World Experimental Setup: We run our experiments on a Sawyer robot equipped with a 2F-85 Robotiq gripper. We use two Intel RealSense D435 RGB-D cameras, one mounted on the wrist of the robot, and the other fixed on a tripod, to observe the environment. The wrist-mounted camera captures a top-down view of the environment at the beginning of an episode (Fig. 3), which is used by a vision model if queried by the LLM. We utilise a pre-trained object detection model, LangSAM [24] (based on Grounding DINO [25] and Segment Anything [26]), and whenever the LLM calls detect\_object, we automatically calculate 3-D bounding boxes of the queried objects from the segmentation maps returned by LangSAM using the camera calibration, and provide the bounding boxes to the LLM. The LLM then leverages this visual understanding of the environment to predict a sequence of 4-D end-effector poses (3 dimensions for position, 1 dimension for rotation about the vertical axis), as well as either open\_gripper or close\_gripper commands. This is then executed by the robot in an open loop, using a position controller to move sequentially between each pose, hence producing a full trajectory. After task execution, we use XMem [27] to track the segmentation maps over the entire duration of the task, which is then later used for detecting if the task was successful or not.

Task Selection: In pursuit of objectivity, we opt to benchmark our zero-shot LLM-guided robotic control against a challenging repertoire of everyday manipulation tasks. We recreated 30 everyday manipulation tasks from recent robotics papers published at leading conferences [5], [28],

[29], [30], [3], often tackled by relying on hundreds of manual demonstrations. This serves as a representative benchmark of real-world challenges, mirroring the complexity and diversity of the tasks encountered in contemporary robotics research. We choose tasks which semantically cover the most representative tabletop robot behaviours expressed in these papers, and success criteria are human-evaluated and designed to mirror those proposed in the original papers. For each combination of task and method in the following experimental sections, we calculate the success rate over 5 trials, randomising the positions and orientations of the objects for each trial. The task description is provided in natural language to the LLM, after which no additional human feedback or intervention is allowed. The full list of tasks is shown in Fig. 6, and example task success and failure videos are available at: https://www.robot-learning.uk/language-models-trajectorygenerators.

## IV. PROMPT DEVELOPMENT

Full Prompt: The core motivation of our work is to investigate whether LLMs can inherently guide robots with minimal dependence on specialised external models and components, in order to provide effective and useful insights for the robotics community. Through this investigation, we designed a single task-agnostic prompt for a range of everyday manipulation tasks, which does not require any in-context examples or task-specific guidance. Fig. 4 illustrates the main information flow in our framework, showing how the task-agnostic prompt interfaces with the vision models and the robot.

Through our experiments outlined in this section, our final prompt formulation instructs the LLM to self-summarise and decompose the predicted plan into steps, before generating Python code which, when run by a standard Python interpreter, outputs a dense sequence of poses for the end-effector to follow; this pipeline resulted in the best performance across those we experimented with. We include details fundamental to all tasks, such as coordinate definitions, as well as functions available for the LLM to call, such as detect\_object, which returns the calculated 3-D bounding boxes of the queried objects directly to the LLM. We also include instructions which aim to improve the correctness and reliability of the generated trajectories, such as guidance on step-by-step reasoning, code generation, and collision avoidance.

**Prompt Ablations:** During the design of this full prompt, we identified several challenges when using LLMs for low-level control, without access to other external dependencies. In this section, we now outline these challenges which motivated the final design of the prompt, and accompany them with results from ablation studies conducted across a diverse set of tasks (Fig. 5), where certain parts of the full prompt were removed.

(1) LLMs often require step-by-step reasoning to solve tasks. Prior work has shown that the reasoning capabilities of LLMs can be improved by asking them to break down the task in a step-by-step manner [31], [32]. Adopting this strategy, we prompt the LLM (1) to break down the trajectory into a sequence of sub-trajectory steps, and (2) to include

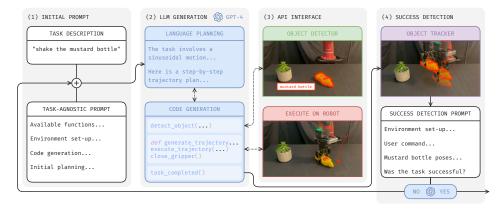


Figure 4: An overview of the pipeline. (1) The main prompt along with the task instruction is provided to the LLM, from which it (2) generates high-level natural language reasoning steps before outputting Python code (3) to interface with a pre-trained object detection model and execute the generated trajectories on the robot. (4) After task execution, an off-the-shelf object tracking model is used to obtain 3-D bounding boxes of the previously detected objects over the duration of the task, which are then provided to the LLM as numerical values to detect whether the task was executed successfully or not.

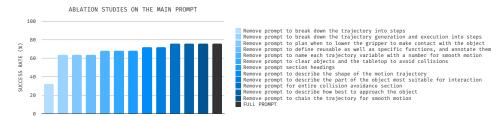


Figure 5: We investigate the effect of removing parts of the main prompt on task success rates.

in the plan when to lower the gripper to make contact with an object. We find that, without including these step-by-step reasoning prompts, the LLM often omits key trajectory steps required to execute the task successfully, such as opening or closing the gripper, or aligning the gripper to be parallel to the graspable side of the object, which are not stated explicitly in the prompt. Indeed, the first three columns in Fig. 5 show that prompting the LLM to think step by step resulted in the highest performance increase.

(2) LLMs can be prone to write code which results in errors, both syntactically and semantically. While much improvement has been made in the domain of code generation by LLMs [33], [6], their outputs can still throw errors, as well as produce undesirable results when executed. In order to mitigate this, and inspired again by the power of LLMs performing an internal monologue with natural language reasoning, we prompt the LLM to document any functions it defines, with their expected inputs and outputs, and their data types. In addition, we include a prompt instructing the LLM to define reusable functions for common motions (for example, linear trajectory from one point to another), to prevent instances where, as a notable example, it would hard-code the height of the gripper inside a function definition, and reuse that function for another sub-trajectory step which should have been executed at a different height. Similarly, we prompt the LLM to name each sub-trajectory step variable with a number to relate it to each of the steps in the high-level trajectory plan, and to minimise the chance of omitting a sub-trajectory step. The effects of removing these prompt components are, again,

noticeable (fourth and fifth columns in Fig. 5).

(3) LLMs are trained on limited grounded physical **interaction data.** Due to the scarcity of grounded physical interaction data in their training corpora [34], LLMs often fail to take into account possible collisions between the objects being manipulated. We therefore prompt the LLM to pay attention to the dimensions of the objects and to generate additional waypoints and sub-trajectories, which could help with avoiding collisions. We also include in the prompt a specific phrase which we noticed during our investigation was being used frequently by the LLM for its internal reasoning ("clear objects and the tabletop"). Our experiments show that, while removing this particular phrase from the collision avoidance prompt lowered performance (sixth column in Fig. 5), LLMs do possess some inherent understanding of possible collisions between different objects, as they performed well even after removing the entire collision avoidance prompt (tenth column in Fig. 5).

(4) LLMs often fail to reason about complex trajectory shapes. In a manner similar to Challenge (1), we employ a two-step strategy, where initially, we explicitly ask the LLM to generate a textual description of the *shape of the motion trajectory* as internal reasoning (for example, shaking involves a sinusoidal motion), before outputting the actual sequence of poses required to execute this trajectory (however, the difference is that in Challenge (1), we prompted the LLM to output a more detailed step-by-step trajectory plan). This has been shown to be beneficial in prior work [4], and indeed this result is also reflected in the eighth column in Fig. 5.

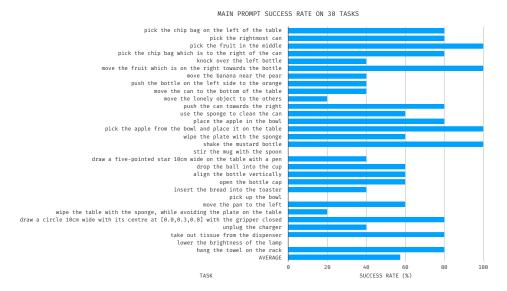


Figure 6: Success rates of the full prompt for 30 manipulation tasks.

(5) LLMs often fail to reason about how to interact with objects. In our experiments, we found that LLMs often simplified and failed to reason about more intricate details of object interaction, such as realising that some objects require interaction with a specific part (for example, the rim of a bowl, or the handle of a pan). In order to enable the LLM to detect the most suitable object part to interact with, we prompt it to describe the object part in high-level natural language, and we see in the ninth column in Fig. 5 that this results in more tasks being executed successfully.

Full Prompt Evaluation: Here, we now investigate the LLM's ability to solve zero-shot a range of manipulation tasks, by evaluating the full prompt on the full set of tasks taken from the recent literature. These tasks and their success rates are presented in Fig. 6. Remarkably, our experiments reveal that LLMs, when equipped with an off-the-shelf vision model and no external motion primitives, policies, or trajectory optimisers, do indeed exhibit notable proficiency in executing the majority of these tasks, by directly predicting a dense sequence of end-effector poses. In the original papers from which these tasks are selected [29], [28], [30], solving these tasks required numerous human demonstrations. As such, these findings underscore the potential of LLMs as intuitive and versatile guides for robotic manipulation that minimise the need for human time and supervision. The full LLM prompts, together with sample LLM outputs, are available at: https://www.robot-learning.uk/language-models-trajectorygenerators.

## V. FURTHER INVESTIGATIONS

In this section, we detail further ablation studies conducted regarding the modality of the trajectory generation, the extent to which each output modality is executable by the robot, the performance of the different available LLMs, and the ability of LLMs to detect whether a task was executed successfully or not and subsequently re-plan the trajectory. We also detail the various sources of error which led to task execution failures when performing the 30 tasks with the main prompt, and

provide a detailed comparison against Code as Policies [7], which is closest to our work.

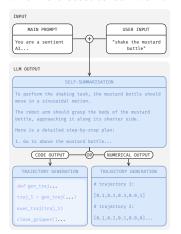


Figure 7: Given the full main prompt and the user input command, the LLM first outputs a high-level natural language self-summarisation of the trajectory plan, before generating either code which computes and executes the trajectory, or the final trajectory directly as a list of numerical values.

(1) How should the final trajectory be represented? In this set of experiments, we explore the optimal way for the LLM to output the sequence of endeffector poses. Specifically, we conduct ablation studies to evaluate whether this should be represented as a list of numerical values or as code for trajectory generation. Fig. 7 shows the distinction between these two output modes.

The results, summarised in Fig. 8 C, offer valuable insights. Notably, our investigation shows that outputting code that generates the trajectory outperforms predicting the trajectory directly as an explicit list of numerical poses for the end-effector to follow, represented as language tokens

(Fig. 7). In particular, we observe that representing trajectories as numerical values or as code yields similar performance for most tasks, with distinctions emerging in cases involving more intricate trajectories (for example, drawing a circle or a five-pointed star), where outputting code that generates such trajectories prevails (60% success rates for code output compared to 10% for numerical output). This suggests a *fundamental property of LLMs for control*: while not trained on physical interactions and trajectories, their **understanding of both code, and mathematical and geometrical structures** [6],

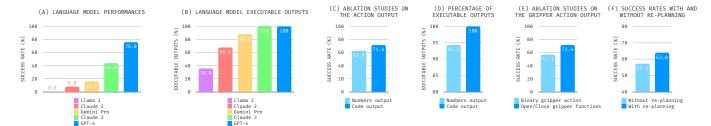


Figure 8: (A) We compare the performance of different widely used LLMs. (B) We compare the abilities of these LLMs to generate outputs that are directly executable by the robot. (C) We compare different modes for the trajectory output. (D) We measure the percentage of control outputs from the LLM that are directly executable by the robot. (E) We compare different modes for controlling the gripper. (F) We demonstrate the ability of LLMs to detect failures and re-plan autonomously.

[35] can bridge these two modes of thinking. Once the overall trajectory shape has been identified by the LLM, while it can be challenging to follow it directly in numbers, it is proficient at generating code that itself can follow complex paths.

Additionally, we study whether directly generating a list of numerical poses, or code that then generates the poses itself, leads to executable outputs more often. Giving lowlevel control to the LLM poses the risk of the robot receiving wrongly formatted outputs that cannot be executed by the robot. Therefore, in this ablation, we investigate how often the output of the LLM is formatted such that it is executable by the robot. We include prompts instructing the LLM to follow a specific format for the trajectory generation (for the numerical poses, we require a list between the \trajectory \ and ⟨/trajectory⟩ tags without any Python functions, and for the code, we require any Python code to be between the "python and "tags). Given the output of the LLM, if an error is thrown during automatic parsing according to this format, we provide the LLM with the error message and ask it to correct the output, for up to three times. Measuring the percentage of executable outputs (Fig. 8 D) demonstrates that outputting code results in 100% of executable trajectories, while direct numerical values cannot be parsed even after three self-corrections for some episodes.

(2) How should the LLM output the gripper action? We also investigate the optimal way of letting the LLM control the gripper open or close action: we compare using a binary variable  $a \in \{0,1\}$  or explicit functions open gripper and close gripper. Our results, in Fig. 8 E, demonstrate that the LLM achieves better performance when using explicit functions, while using a binary variable leads to more errors. A notable failure case stemmed from the LLM hard-coding the gripper state to be open in one of the functions it defined for itself, such that when the same function was then used to generate the object approach-and-lift sub-trajectory steps, the gripper failed to close and grasp the object. Having explicit functions to open and close the gripper, on the other hand, allowed a decoupling of these fundamental actions and enabled the correct functions to be called at any time during the overall trajectory generation plan.

(3) Which of the currently available LLMs performs best at following instructions outlined in the prompt? Next, we present results on ablation studies conducted to determine the best-performing LLM on the main prompt. We see in

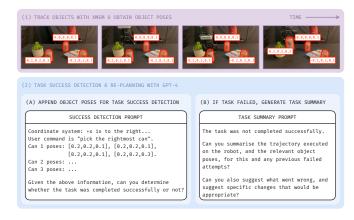


Figure 9: Our experiments demonstrate that LLMs can interpret the trajectories of objects to detect successful and unsuccessful episodes.

Fig. 8 B that out of the five most popular LLMs currently available for public use to date [6], [36], [37], [38], [39], and under the same constraints as the executable output study presented in Fig. 8 D, only GPT-4 and Claude 3 (Opus) were able to generate code which was able to be executed 100% of the time (regardless of whether the resulting code output completed the task successfully or not), whereas the outputs for Gemini 1.0 Pro, Claude 2 and Llama 2 (70B Chat) were only executable 88.0%, 68.0% and 36.0% of the time, respectively. The corresponding success rates for the five LLMs are shown in Fig. 8 A.

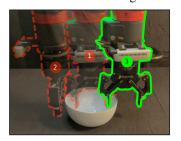


Figure 10: (1) The LLM attempts to grasp the bowl at its centroid, recognises failure, and (2) proposes a new trajectory. (3) On its third attempt after re-planning again, it successfully grasps the bowl.

(4) Can LLMs recognise unsuccessful trajectories, and adapt their plan? We also delve into the ability of LLMs to recognise and respond to failures during task execution. Our experiments demonstrate that, by analysing the numerical trajectories of objects, LLMs can autonomously detect failure outcomes and initiate re-planning to rectify them. We therefore demonstrate that LLMs possess not only the ability to generate

trajectories, but also to discern whether they represent successful or unsuccessful episodes, given the tasks requested by the user. Our proposed pipeline for task success detection and re-planning is shown in Fig. 9.

For each of the 5 trials of a task, when a failure is identified, the LLM modifies the original plan to tackle the possible issue. In Fig. 8 F, we demonstrate that this leads to a small improvement in performance, without the need for any human intervention. As a notable example, the LLM always fails at grasping a bowl on its first try (Fig. 6), attempting to grasp it by the centroid (Fig. 10). Through a sequence of two replanning iterations, however, the LLM adapts its trajectory and then successfully grasps the bowl by its rim, leading to an increase from 0% to 20% in the overall task execution success rate.

(5) What were the main failure modes? We present the main failure modes of the main prompt on the 30 manipulation tasks. We group the sources of error into the following categories: (1) gripper pose prediction error, where the LLM was unable to predict accurate enough goal poses for the gripper, and this resulted in wrong parameterisations of trajectory generation functions (which the LLM defined for itself previously); (2) task planning error, where the LLM was unable to plan a correct high-level sequence of steps to complete the task successfully (for example, calling close gripper before lowering the gripper to the object to grasp it); (3) trajectory generation function definition error, where the LLM coded wrongly the function which would be used to generate the dense sequence of end-effector poses (the function itself would be executable by a Python interpreter, but as opposed to Error (1) which was concerned with the parameterisation of such functions, the function itself was wrongly defined; for example, hard-coding the gripper to be open or closed, or its height within the function definition); (4) object detection error, where the wrong segmentation map was returned by the object detector based on the text queried by the LLM; (5) camera calibration error, where the LLM might have generated the correct trajectory for the bounding box it was provided with, but the bounding box itself was incorrectly calculated due to noisy camera data and this resulted in the task being unsuccessful. The full breakdown is shown in Fig. 11.

We can see that nearly half of the failure cases can be attributed to the more difficult task of predicting accurate gripper poses (48.3%), compared to high-level planning (25.9%), of which numerous works have already shown that LLMs are capable [5], [40]. Regarding the object detection errors, most of the errors were due to the object detector not being able to segment the object parts correctly when queried by the LLM, such as the rim of a bowl, or the slot of a toaster.

(6) How does our work compare to Code as Policies? Finally, we perform two further sets of experiments in simulation and with ground-truth object poses, comparing our zero-shot task-agnostic prompt to the prompt structure set out in Code as Policies (CaP) [7], which makes use of in-context examples and motion primitives. Firstly, we compare the performance of the original CaP prompt (containing 77 in-context examples) on the 30 tasks, using the original CaP API code, and compare to our zero-shot prompt. We find that while for basic pick-and-

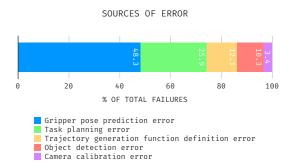


Figure 11: A breakdown of the different error types which caused task execution failures with the main prompt across the 30 tasks.

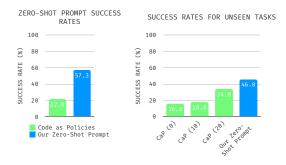


Figure 12: (**Left**) We compare our zero-shot task performance across 30 tasks to that of Code as Policies (CaP). (**Right**) We demonstrate that our prompt is able to generalise to unseen tasks with higher performance than using CaP-style prompting with a varying number of in-context examples (in brackets).

place tasks, for which examples had already been provided in the original CaP prompt, it has near-perfect success rates, it is unable to generalise at all to more complex tasks, resulting in a lower average success rate of 22.0%, compared to 57.3% for our prompt. These results are shown in Fig. 12 (Left).

Secondly, we study the role of CaP-style in-context examples in enabling (or limiting) unseen task generalisation, and compare on 10 unseen tasks the success rates of CaP prompts with a varying number of additional manually engineered in-context examples (0, 10, and 20). For the unseen tasks, we select 10 from the same set of 30 tasks, and for the additional in-context examples, we write one example for each of the remaining 20 tasks, and select 0, 10, or all 20 of them. We write these additional examples in the style of CaP, but covering a wider range of tasks beyond simple pickand-place tasks, and append them to the original CaP prompt. We find that with these additional examples, the CaP prompt is able to generalise better to unseen tasks, but even with the 20 additional manually engineered in-context examples (which cover tasks of similar complexity to the 10 unseen tasks), the three CaP prompt variants are unable to perform as well (highest 34.0%) as our zero-shot task-agnostic prompt (46.0%). These results are shown in Fig. 12 (Right).

## VI. CONCLUSIONS

The primary contribution of this paper is an investigation into to what extent an LLM can successfully predict dense

sequences of end-effector poses for a range of real-world manipulation tasks. To differentiate this from other previous works, we imposed constraints that the LLM must use a single task-agnostic prompt without any in-context examples, and has access to only off-the-shelf object detection and segmentation vision models, with no other auxiliary components. Our experiments encompassed 30 diverse tasks drawn from the recent literature, and we showed that GPT-4, together with the prompt we designed, can perform well on many of these tasks.

With today's LLMs and our zero-shot prompt, there are several types of tasks which are too challenging, such as those requiring high precision, complex trajectory shapes, and richer perception beyond just computing bounding boxes. As LLMs continue to improve, we may see improved abilities to solve these tasks, and recent advances in VLMs will help to better interface language and vision. But for now, this paper raises the assumed limit of the utility of LLMs for robotics, and we hope that the insights we provide will help those developing their own LLM-based robot controllers, and will encourage those leading the field in LLMs and VLMs to further align developments with robotics applications.

## ACKNOWLEDGEMENT

The authors wish to thank Kamil Dreczkowski, Georgios Papagiannis and Pietro Vitiello for their valuable discussion and feedback during the writing of the paper.

#### REFERENCES

- [1] W. X. Zhao et al., "A Survey of Large Language Models," arXiv e-prints, p. arXiv:2303.18223, Mar. 2023.
- [2] L. Wang et al., "A Survey on Large Language Model based Autonomous Agents," arXiv e-prints, p. arXiv:2308.11432, Aug. 2023.
- [3] W. Huang et al., "VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models," arXiv e-prints, p. arXiv:2307.05973, July 2023.
- [4] W. Yu et al., "Language to Rewards for Robotic Skill Synthesis," arXiv e-prints, p. arXiv:2306.08647, June 2023.
- [5] M. Ahn et al., "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances," arXiv e-prints, p. arXiv:2204.01691, Apr. 2022.
- [6] OpenAI, "GPT-4 Technical Report," arXiv e-prints, p arXiv:2303.08774, Mar. 2023.
- [7] J. Liang et al., "Code as policies: Language model programs for embodied control," in 2023 IEEE International Conference on Robotics and Automation (ICRA), May 2023, pp. 9493–9500.
- [8] S. Vemprala et al., "Chatgpt for robotics: Design principles and model abilities," Microsoft, Tech. Rep. MSR-TR-2023-8, February 2023. [Online]. Available: https://www.microsoft.com/en-us/research/ publication/chatgpt-for-robotics-design-principles-and-model-abilities/
- [9] A. Brohan et al., "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control," arXiv e-prints, p. arXiv:2307.15818, July 2023.
- [10] D. Driess et al., "PaLM-e: An embodied multimodal language model," in Proceedings of the 40th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, A. Krause et al., Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 8469–8488. [Online]. Available: https://proceedings.mlr.press/v202/driess23a.html
- [11] J.-B. Alayrac *et al.*, "Flamingo: a visual language model for few-shot learning," in *Advances in Neural Information Processing Systems*, S. Koyejo *et al.*, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 23716–23736. [Online]. Available: https://proceedings.neurips.cc/paper\_files/paper/2022/file/960a172bc7fbf0177ccccbb411a7d800-Paper-Conference.pdf
- [12] N. Di Palo and E. Johns, "Keypoint Action Tokens Enable In-Context Imitation Learning in Robotics," arXiv e-prints, p. arXiv:2403.19578, Mar. 2024.
- [13] S. Mirchandani et al., "Large Language Models as General Pattern Machines," arXiv e-prints, p. arXiv:2307.04721, July 2023.

- [14] A. Zeng *et al.*, "Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language," *arXiv e-prints*, p. arXiv:2204.00598, Apr. 2022
- [15] W. Huang et al., "Inner Monologue: Embodied Reasoning through Planning with Language Models," arXiv e-prints, p. arXiv:2207.05608, July 2022.
- [16] N. Di Palo et al., "Towards A Unified Agent with Foundation Models," arXiv e-prints, p. arXiv:2307.09668, July 2023.
- [17] W. Shen *et al.*, "Distilled Feature Fields Enable Few-Shot Language-Guided Manipulation," *arXiv e-prints*, p. arXiv:2308.07931, July 2023.
- [18] J. Kerr et al., "LERF: Language Embedded Radiance Fields," arXiv e-prints, p. arXiv:2303.09553, Mar. 2023.
- [19] A. Rashid et al., "Language Embedded Radiance Fields for Zero-Shot Task-Oriented Grasping," arXiv e-prints, p. arXiv:2309.07970, Sept. 2023
- [20] N. Di Palo and E. Johns, "DINOBot: Robot Manipulation via Retrieval and Alignment with Vision Foundation Models," arXiv e-prints, p. arXiv:2402.13181, Feb. 2024.
- [21] ——, "On the Effectiveness of Retrieval, Alignment, and Replay in Manipulation," *arXiv e-prints*, p. arXiv:2312.12345, Dec. 2023.
- [22] I. Kapelyukh et al., "DALL-E-Bot: Introducing Web-Scale Diffusion Models to Robotics," arXiv e-prints, p. arXiv:2210.02438, Oct. 2022.
- [23] —, "Dream2Real: Zero-Shot 3D Object Rearrangement with Vision-Language Models," arXiv e-prints, p. arXiv:2312.04533, Dec. 2023.
- [24] L. Medeiros, "Langsam: Language segment-anything," https://github. com/luca-medeiros/lang-segment-anything, accessed: 2023-10-01.
- [25] S. Liu et al., "Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection," arXiv e-prints, p. arXiv:2303.05499, Mar. 2023.
- [26] A. Kirillov et al., "Segment Anything," arXiv e-prints, p. arXiv:2304.02643, Apr. 2023.
- [27] H. K. Cheng and A. G. Schwing, "Xmem: Long-term video object segmentation with an atkinson-shiffrin memory model," in *Computer Vision – ECCV 2022*, S. Avidan *et al.*, Eds. Cham: Springer Nature Switzerland, 2022, pp. 640–658.
- [28] T. Xiao et al., "Robotic Skill Acquisition via Instruction Augmentation with Vision-Language Models," arXiv e-prints, p. arXiv:2211.11736, Nov. 2022.
- [29] A. Brohan et al., "RT-1: Robotics Transformer for Real-World Control at Scale," arXiv e-prints, p. arXiv:2212.06817, Dec. 2022.
- [30] T. Yu et al., "Scaling Robot Learning with Semantically Imagined Experience," arXiv e-prints, p. arXiv:2302.11550, Feb. 2023.
- [31] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in Advances in Neural Information Processing Systems, S. Koyejo et al., Eds., vol. 35. Curran Associates, Inc., 2022, pp. 24824–24837. [Online]. Available: https://proceedings.neurips.cc/paper\_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf
- [32] T. Kojima et al., "Large language models are zero-shot reasoners," in Advances in Neural Information Processing Systems, S. Koyejo et al., Eds., vol. 35. Curran Associates, Inc., 2022, pp. 22199–22213. [Online]. Available: https://proceedings.neurips.cc/paper\_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf
- [33] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv e-prints, p. arXiv:2107.03374, July 2021.
- [34] J. Hoffmann et al., "Training Compute-Optimal Large Language Models," arXiv e-prints, p. arXiv:2203.15556, Mar. 2022.
- [35] H. Luo et al., "WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct," arXiv e-prints, p. arXiv:2308.09583, Aug. 2023.
- [36] Anthropic, "Model card and evaluations for claude models," https:// www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf, accessed: 2023-12-14.
- [37] H. Touvron et al., "Llama 2: Open Foundation and Fine-Tuned Chat Models," arXiv e-prints, p. arXiv:2307.09288, July 2023.
- [38] Anthropic, "The claude 3 model family: Opus, sonnet, haiku," https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model\_Card\_Claude\_3.pdf, accessed: 2024-04-11.
- [39] Gemini Team, "Gemini: A Family of Highly Capable Multimodal Models," arXiv e-prints, p. arXiv:2312.11805, Dec. 2023.
- [40] W. Huang et al., "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in Proceedings of the 39th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, K. Chaudhuri et al., Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 9118–9147. [Online]. Available: https://proceedings.mlr.press/v162/huang22a.html

#### 9

## **APPENDIX**

## PIPELINE ALGORITHM

## Algorithm 1 Zero-Shot Trajectory Generation with LLMs

```
Given: full task-agnostic prompt p, and task instruction l
 1: prompt \leftarrow p \oplus l
                                                    2: task\_completed \leftarrow False
 3: while task\_completed = False do
        output \leftarrow \text{LLM}(prompt)
 4:
        prompt \leftarrow None
 5:
 6:
        if output contains code then
                                                    7:
            try exec(output)
            except Exception then
 8:
                 prompt \leftarrow error message
 9:
                                               ▷ predefined APIs
            else
10:
                 if detect object(object) is called then
11:
                     pos, orn, dim \leftarrow \text{detect\_object}(object)
12:
                     prompt \leftarrow pos, orn, dim
13:
                 else if execute\_trajectory(t) is called then
14:
                     move robot(t)
                                               \triangleright t is list of poses
15:
                 else if task completed() is called then
16:
                     obj\_poses \leftarrow XMem(traj\_frames)
17:
                     task\_completed \leftarrow LLM(obj\_poses)
18:
                     if task\_completed = False then
19:
                         reset_environment()
20:
                         summary \leftarrow LLM(obj\_poses)
21:
22:
                         prompt \leftarrow p \oplus l \oplus summary
                     end if
23.
24:
                 end if
            end try
25.
        end if
26:
27: end while
```

## TASKS SELECTED FOR ABLATION STUDIES

- A. Ablation Studies on the Main Prompt & Language Model Comparisons
  - pick the chip bag which is to the right of the can
  - place the apple in the bowl
  - shake the mustard bottle
  - open the bottle cap
  - move the pan to the left
- B. Ablation Studies on the Action Output
  - pick the chip bag which is to the right of the can
  - place the apple in the bowl
  - shake the mustard bottle
  - open the bottle cap
  - move the pan to the left
  - draw a five-pointed star 10cm wide on the table with a pen
  - draw a circle 10cm wide with its centre at [0.0,0.3,0.0] with the gripper closed

## C. Code as Policies Comparison

- open the bottle cap
- insert the bread into the toaster
- pick up the bowl
- move the pan to the left
- wipe the table with the sponge, while avoiding the plate on the table
- draw a circle 10cm wide with its centre at [0.0,0.3,0.0] with the gripper closed
- unplug the charger
- take out tissue from the dispenser
- lower the brightness of the lamp
- hang the towel on the rack

## TASK SUCCESS CRITERIA

- pick the chip bag on the left of the table: the left chip bag is lifted at least 10cm off the table.
- *pick the rightmost can*: the rightmost can is lifted at least 10cm off the table.
- *pick the fruit in the middle*: the horizontally middle fruit is lifted at least 10cm off the table.
- pick the chip bag which is to the right of the can: the chip bag to the right of the can is lifted at least 10cm off the table.
- *knock over the left bottle*: the left bottle is knocked over so that it is lying on its side.
- move the fruit which is on the right towards the bottle: the right fruit is placed within 5cm of the bottle.
- move the banana near the pear: the banana is placed within 5cm of the pear.
- push the bottle on the left side to the orange: the left bottle is pushed (remains in contact with the table throughout) within 5cm of the orange.
- move the can to the bottom of the table: the can is placed within 10cm of the bottom edge of the table.
- move the lonely object to the others: the object which is furthest away is placed within 5cm of one of the other objects.
- *push the can towards the right*: the can is pushed (remains in contact with the table throughout) to the right by at least 10cm.
- use the sponge to clean the can: the sponge touches the can.
- place the apple in the bowl: the apple is placed inside the bowl.
- pick the apple from the bowl and place it on the table: the apple is placed anywhere on the table out of the bowl.
- wipe the plate with the sponge: the sponge is moved in any wiping motion (zigzag, circular, etc.) while remaining in contact with the plate throughout.
- *shake the mustard bottle*: the mustard bottle is moved in any shaking motion (up-and-down, left-and-right, etc.) at least twice, at any speed.
- stir the mug with the spoon: the spoon is picked up from an upright pose, inserted into the mug, and is moved in a circular motion inside the mug.

- draw a five-pointed star 10cm wide on the table with a pen: the pen is picked up from an upright pose, and is moved while remaining in contact with the table throughout such that it draws a five-pointed star 10cm wide.
- *drop the ball into the cup*: the ball is placed inside the cup.
- *align the bottle vertically*: the bottle is rotated such that it is pointing either the top or bottom edge of the table while lying on its side.
- *open the bottle cap*: the bottle cap is rotated from a closed position so that it can be lifted off the bottle.
- insert the bread into the toaster: the bread is placed inside any of the toaster slots.
- *pick up the bowl*: the bowl is lifted at least 10cm off the table.
- move the pan to the left: the pan is moved to the left by

- at least 10cm.
- wipe the table with the sponge, while avoiding the plate on the table: the sponge is moved in any wiping motion (zigzag, circular, etc.) while remaining in contact with the table throughout, and it should not touch the plate.
- draw a circle 10cm wide with its centre at [0.0,0.3,0.0] with the gripper closed: the gripper is closed and moved in a circular motion 10cm wide with its centre on the table and 30cm directly in front of the base of the robot.
- *unplug the charger*: the charging block is completely removed from the extension plug socket.
- take out tissue from the dispenser: the tissue is completely removed from the dispenser.
- lower the brightness of the lamp: the dimmer switch is rotated anticlockwise.
- hang the towel on the rack: the towel is placed stably on the rack without touching the table.