VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, Li Fei-Fei
1Stanford University  2University of Illinois Urbana-Champaign

Abstract: Large language models (LLMs) are shown to possess a wealth of actionable knowledge that can be extracted for robot manipulation in the form of reasoning and planning. Despite the progress, most still rely on pre-defined motion primitives to carry out the physical interactions with the environment, which remains a major bottleneck. In this work, we aim to synthesize robot trajectories, i.e., a dense sequence of 6-DoF end-effector waypoints, for a large variety of manipulation tasks given an open-set of instructions and an open-set of objects. We achieve this by first observing that LLMs excel at inferring affordances and constraints given a free-form language instruction. More importantly, by leveraging their code-writing capabilities, they can interact with a vision-language model (VLM) to compose 3D value maps to ground the knowledge into the observation space of the agent. The composed value maps are then used in a model-based planning framework to zero-shot synthesize closed-loop robot trajectories with robustness to dynamic perturbations. We further demonstrate how the proposed framework can benefit from online experiences by efficiently learning a dynamics model for scenes that involve contact-rich interactions. We present a large-scale study of the proposed method in both simulated and real-robot environments, showcasing the ability to perform a large variety of everyday manipulation tasks specified in free-form natural language. Project website: voxposer.github.io.

Keywords: Manipulation, Large Language Models, Model-based Planning

Figure 1: VoxPoser extracts language-conditioned affordances and constraints from LLMs and grounds them to the perceptual space using VLMs, using a code interface and without additional training to either component. The composed map is referred to as a 3D value map, which enables zero-shot synthesis of trajectories for large varieties of everyday manipulation tasks with an open-set of instructions and an open-set of objects.

Correspondence to Wenlong Huang <wenlongh@stanford.edu>.

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.
1 Introduction

Language is a compressed medium through which humans distill and communicate their knowledge and experience of the world. Large language models (LLMs) have emerged as a promising approach to capture this abstraction, learning to represent the world through projection into language space [1–4]. While these models are believed to internalize generalizable knowledge as text, it remains a question about how to use it to enable embodied agents to physically act in the real world.

We look at the problem of grounding abstract language instructions (e.g., “set up the table”) in robot actions [5]. Prior works have leveraged lexical analysis to parse the instructions [6–8], while more recently language models have been used to decompose the instructions into a textual sequence of steps [9–11]. However, to enable physical interactions with the environment, existing approaches typically rely on a repertoire of pre-defined motion primitives (i.e., skills) that may be invoked by an LLM or a planner, and this reliance on individual skill acquisition is often considered a major bottleneck of the system due to the lack of large-scale robotic data. The question then arises: how can we leverage the wealth of internalized knowledge of LLMs at the even fine-grained action level for robots, without requiring laborious data collection or manual designs for each individual primitive?

In addressing this challenge, we first note that it is impractical for LLMs to directly output control actions in text, which are typically driven by high-frequency control signals in high-dimensional space. However, we find that LLMs excel at inferring language-conditioned affordances and constraints, and by leveraging their code-writing capabilities, they can compose dense 3D voxel maps that ground them in the visual space by orchestrating perception calls (e.g., via CLIP [12] or open-vocabulary detectors [13–15]) and array operations (e.g., via NumPy [16]). For example, given an instruction “open the top drawer and watch out for the vase”, LLMs can be prompted to infer: 1) the top drawer handle should be grasped, 2) the handle needs to be translated outwards, and 3) the robot should stay away from the vase. By generating Python code to invoke perception APIs, LLMs can obtain spatial-geometric information of relevant objects or parts and then manipulate the 3D voxels to prescribe reward or cost at relevant locations in observation space (e.g., the handle region is assigned high values while the surrounding of the vase is assigned low values). Finally, the composed value maps can serve as objective functions for motion planners to directly synthesize robot trajectories that achieve the given instruction, without requiring additional training data for each task or for the LLM. An illustration diagram and a subset of tasks we considered are shown in Fig. 1.

We term this approach Voxposer, a formulation that extracts affordances and constraints from LLMs to compose 3D value maps in observation space for guiding robotic interactions. Rather than relying on robotic data that are often of limited amount or variability, the method leverages LLMs for open-world reasoning and VLMs for generalizable visual grounding in a model-based planning framework that directly enables physical robot actions. We demonstrate its zero-shot generalization for open-set instructions with open-set objects for various everyday manipulation tasks. We further showcase how VoxPoser can also benefit from limited online interactions to efficiently learn a dynamics model that involves contact-rich interactions.

2 Related Works

Grounding Language Instructions. Language grounding has been studied extensively both in terms of intelligent agents [19–22] and of robotics [23, 6, 24, 25, 5, 7, 26], where language can be used as a tool for compositional goal specification [5, 27–33], semantic anchor for training multimodal representation [12, 34, 35], or as an intermediate substrate for planning and reasoning [36–38, 9, 10, 39, 40]. Prior works have looked at using classical tools such as lexical analysis, formal logic, and graphical models to interpret language instructions [27, 7, 6, 26]. More recently, end-to-end approaches, popularized by successful applications to offline domains [41–43], have been applied to directly ground language instructions in robot interactions by learning from data with

1 The approach also bears resemblance and connections to potential field methods in path planning [17] and constrained optimization methods in manipulation planning [18].
language annotations, spanning from model learning [44], imitation learning [45, 46, 30, 47–54], to reinforcement learning [55–57]. Most closely related to our work is Sharma et al. [50], where an end-to-end cost predictor is optimized via supervised learning to map language instructions to 2D costmaps, which are used to steer a motion planner to generate preferred trajectories in a collision-free manner. In contrast, we rely on pre-trained language models for their open-world knowledge and tackle the more challenging robotic manipulation in 3D.

**Language Models for Robotics.** Leveraging pre-trained language models for embodied applications is an active area of research, where a large body of works focus on planning and reasoning with language models [9–11, 58, 31, 39, 59–72, 36, 73, 74]. To allow language models to perceive the physical environments, textual descriptions of the scene [39, 11, 59] or perception APIs [75] can be given, vision can be used during decoding [67] or can be directly taken as input by multi-modal language models [68, 2]. In addition to perception, to truly bridge the perception-action loop, an embodied language model must also know how to act, which typically is achieved by a library of pre-defined primitives. Liang et al. [75] showed that LLMs exhibit behavioral commonsense that can be useful for low-level control. Despite the promising signs, hand-designed motion primitives are still required, and while LLMs are shown to be capable of composing sequential policy logic, it remains unclear whether composition can happen at spatial level. A related line of works has also explored using LLMs for reward specification in the context of reward design [76] and exploration in reinforcement learning [77–80], and human preference learning [81]. In contrast, we focus exclusively on grounding the reward generated by LLMs in the 3D observation space of the robot, which we identify as most useful for manipulation tasks.

**Learning-based Trajectory Optimization.** Many works have explored leveraging learning-based approaches for trajectory optimization. While the literature is vast, they can be broadly categorized into those that learn the models [82–90] and those that learn the cost/reward or constraints [91–94, 50, 95], where data are typically collected from in-domain interactions. To enable generalization in the wild, a parallel line of works has explored learning task specification from large-scale offline data [96–98, 35, 34, 44, 99, 100, 54], particularly egocentric videos [101, 102], or leveraging pre-trained foundation models [103–105, 33, 106, 107]. The learned cost functions are then used by reinforcement learning [103, 100, 108], imitation learning [98, 97], or trajectory optimization [96, 35] to generate robot actions. In this work, we leverage LLMs for zero-shot in-the-wild cost specification with superior generalization. Compared to prior works that leverage foundation models, we ground the cost directly in 3D observation space with real-time visual feedback, which makes VoxPoser amenable to closed-loop MPC that’s robust in execution.

## 3 Method

We first provide the formulation of VoxPoser as an optimization problem (Sec. 3.1). Then we describe how VoxPoser can be used as a general zero-shot framework to map language instructions to 3D value maps (Sec. 3.2). We subsequently demonstrate how trajectories can be synthesized in closed-loop for robotic manipulation (Sec. 3.3). While zero-shot in nature, we demonstrate how VoxPoser can learn from online interactions to efficiently solve contact-rich tasks (Sec. 3.4).

### 3.1 Problem Formulation

Consider a manipulation problem given as a free-form language instruction $L$ (e.g., “open the top drawer”). Generating robot trajectories according to $L$ can be very challenging because $L$ may be arbitrarily long-horizon or under-specified (i.e., requires contextual understanding). Instead, we focus on individual phases (sub-tasks) of the problem $\ell_i$ that distinctively specify a manipulation task (e.g., “grasp the drawer handle”, “pull open the drawer”), where the decomposition $L \rightarrow (\ell_1, \ell_2, \ldots, \ell_n)$ is given by a high-level planner (e.g., an LLM or a search-based planner) ². The central problem investigated in this work is to generate a motion trajectory $\tau^r_i$ for robot $r$ and each manipulation

²Note that the decomposition and sequencing of these sub-tasks are also done by LLMs in this work, though we do not investigate this aspect extensively as it is not the focus of our contributions.
def affordance_map():
    msize = (100, 100, 100)
    map = np.zeros(msize)
    handles = detect('handle')
    k = lambda x: x.pos[2]
    handles.sort(key=k)
    for handle in handles:
        top_handle = handle
    return smooth(map)

constraint_map().

def constraint_map():
    msize = (100, 100, 100)
    map = np.zeros(msize)
    vases = detect('vase')
    vase.occupancy_grid = VoxelGrid(vase.occupancy_grid)
    return smooth(map)

Figure 2: Overview of VoxPoser. Given the RGB-D observation of the environment and a language instruction, LLMs generate code, which interacts with VLMs, to produce a sequence of 3D affordance maps and constraint maps (collectively referred to as value maps) grounded in the observation space of the robot (a). The composed value maps then serve as objective functions for motion planners to synthesize trajectories for robot manipulation (b). The entire process does not involve any additional training.

phase described by instruction \( \ell_i \). We represent \( \tau^r_i \) as a sequence of dense end-effector waypoints to be executed by an Operational Space Controller [109], where each waypoint consists of a desired 6-DoF end-effector pose, end-effector velocity, and gripper action. However, it is worth noting that other representations of trajectories, such as joint space trajectories, can also be used. Given the \( i \)-th sub-task described by \( \ell_i \), we formulate an optimization problem defined as follows:

\[
\min_{\tau^r_i} \{ F_{task}(T_i, \ell_i) + F_{control}(\tau^r_i) \} \quad \text{subject to} \quad C(T_i)
\]  

where \( T_i \) is the evolution of environment state, and \( \tau^r_i \subseteq \tau_i \) is the robot trajectory. \( F_{task} \) scores the extent of \( T_i \) completes the instruction \( \ell_i \) while \( F_{control} \) specifies the control costs, e.g., to encourage \( \tau^r_i \) to minimize total control effort or total time. \( C(T_i) \) denotes the dynamics and kinematics constraints, which are enforced by the known model of the robot and a physics-based or learning-based model of the environment. By solving this optimization for each sub-task \( \ell_i \), we obtain a sequence of robot trajectories that collectively achieve the overall task specified by the instruction \( I \).

3.2 Grounding Language Instruction via VoxPoser

Calculating \( F_{task} \) with respect to free-form language instructions is extremely challenging, not only because of the rich space of semantics language can convey but also because of the lack of robot data labeled with \( T \) and \( \ell \). However, we provide a critical observation that a large number of tasks can be characterized by a voxel value map \( V \in \mathbb{R}^{w \times h \times d} \) in robot’s observation space, which guides the motion of an “entity of interest” in the scene, such as the robot end-effector, an object, or an object part. For example, consider the task “open the top drawer” and its first sub-task “grasp the top drawer handle” (inferred by LLMs) in Fig. 2. The “entity of interest” is the robot end-effector, and the voxel value map should reflect the attraction toward the drawer handle. By further commanding “watch out for the vase”, the map can also be updated to reflect the repulsion from the vase. We denote the “entity of interest” as \( e \) and its trajectory as \( \tau^e \). Using this voxel value map for a given instruction \( \ell_i \), \( F_{task} \) can be approximated by accumulating the values of \( e \) traversing through \( V_i \), formally calculated as

\[
F_{task} = -\sum_{j=1}^{\tau^r_i} V(p^e_j), \quad \text{where} \quad p^e_j \in \mathbb{N}^3 \quad \text{is the discretized} \quad (x, y, z) \quad \text{position of} \quad e \quad \text{at step} \quad j.
\]

Notably, we observe large language models, by being pre-trained on Internet-scale data, exhibit capabilities not only to identify the “entity of interest” but also to compose value maps that accurately reflect the task instruction by writing Python programs. Specifically, when an instruction is given as a comment in the code, LLMs can be prompted to 1) call perception APIs (which invoke vision-language models (VLM) such as an open-vocabulary detector [13–15]) to obtain spatial-geometrical information of relevant objects, 2) generate NumPy operations to manipulate 3D arrays, and 3) prescribe precise values at relevant locations. We term this approach as VoxPoser. Concretely, we aim to obtain a voxel value map \( V^e_i = \text{VoxPoser}(o^i, \ell_i) \) by prompting an LLM and executing the code via a Python interpreter, where \( o^i \) is the RGB-D observation at time \( t \) and \( \ell_i \) is the current
Figure 3: Visualization of composed 3D value maps and rollouts in real-world environments. The top row demonstrates where “entity of interest” is an object or part, and the value maps guide them toward target positions. The bottom two rows showcase tasks where “entity of interest” is the robot end-effector. The bottom-most task involves two phases, which are also orchestrated by LLMs.

Additional Trajectory Parametrization. The above formulation of VoxPoser uses LLMs to compose $V: \mathbb{N}^3 \to \mathbb{R}$ to map from discretized coordinates in voxel space to a real-valued “cost”, which we can use to optimize a path consisting only of the positional terms. To extend to SE(3) poses, we can also use LLMs to compose rotation maps $V_r: \mathbb{N}^3 \to \text{SO}(3)$ at coordinates relevant to the task objectives (e.g., “end-effector should face the support normal of the handle”). Similarly, we further compose gripper maps $V_g: \mathbb{N}^3 \to \{0, 1\}$ to control gripper open/close and velocity maps $V_v: \mathbb{N}^3 \to \mathbb{R}$ to specify target velocities. Note that while these additional trajectory parametrizations are not mapped to a real-valued “cost”, they can also be factored in the optimization procedure (Equation 1) to parametrize the trajectories.

3.3 Zero-Shot Trajectory Synthesis with VoxPoser

After obtaining the task cost $F_{\text{task}}$, we can now approach the full problem defined in Equation 1 to plan a motion trajectory. We use simple zeroth-order optimization by randomly sampling trajectories and scoring them with the proposed objective. The optimization is implemented in a model predictive control framework that iteratively replans the trajectory at every step using the current observation to robustly execute the trajectories even under dynamic disturbances $^3$, where either a learned or physics-based model can be used. However, because VoxPoser effectively provides “dense rewards” in the observation space and we are able to replan at every step, we surprisingly find that the overall system can already achieve a large variety of manipulation tasks considered in this work even with simple heuristics-based models. Since some value maps are defined over “entity of interest”, which may not necessarily be the robot, we also use the dynamics model to find the needed robot trajectory to minimize the task cost (i.e., what interactions between the robot and the environment achieve the desired object motions).

3.4 Efficient Dynamics Learning with Online Experiences

While Sec. 3.3 presents a zero-shot framework for synthesizing trajectories for robot manipulation, VoxPoser can also benefit from online experiences by efficiently learning a dynamics model. Consider the standard setup where a robot interleaves between 1) collecting environment transition data $(o_t, a_t, o_{t+1})$, where $o_t$ is the environment observation at time $t$ and $a_t = \text{MPC}(o_t)$, and 2) training a dynamics model $g_\theta$ parametrized by $\theta$ by minimizing the L2 loss between predicted next...
observation \( o_{t+1} \) and \( o_{t+2} \). A critical component that determines the learning efficiency is the action sampling distribution \( P(a_t | o_t) \) in MPC, which typically is a random distribution over the full action space \( A \). This is often inefficient when the goal is to solve a particular task, such as opening a door, because most actions do not interact with the relevant objects in the scene (i.e., the door handle) nor do they necessarily interact with the objects in a meaningful way (i.e., pressing down the door handle). Since VoxPoser synthesizes robot trajectories with LLMs, which have a wealth of commonsense knowledge, the zero-shot synthesized trajectory \( \tau^0_r \) can serve as a useful prior to bias the action sampling distribution \( P(a_t | o_t, \tau^0_r) \), which can significantly speed up the learning process. In practice, this can be implemented by only sampling actions in the vicinity of \( \tau^0_r \) by adding small noise \( \varepsilon \) to encourage local exploration instead of exploring in the full action space \( A \).

4 Experiments and Analysis

We first discuss our implementation details. Then we validate VoxPoser for real-world everyday manipulation (Sec. 4.1). We also study its generalization in simulation (Sec. 4.2). We further demonstrate how VoxPoser enables efficient learning of more challenging tasks (Sec. 4.3). Finally, we analyze its source of errors and discuss how improvement can be made (Sec. 4.4).

**LLMs and Prompting.** We follow prompting structure by Liang et al. [75], which recursively calls LLMs using their own generated code, where each language model program (LMP) is responsible for a unique functionality (e.g., processing perception calls). We use GPT-4 [2] from OpenAI API. For each LMP, we include 5-20 example queries and corresponding responses as part of the prompt. An example can be found in Fig. 2 (simplified for clarity). Full prompts are in Appendix.

**VLMs and Perception.** Given an object/part query from LLMs, we first invoke open-vocab detector OWL-ViT [15] to obtain a bounding box, then feed it into Segment Anything [110] to obtain a mask, and finally track the mask using video tracker XMEM [111]. The tracked mask is used with RGB-D observation to reconstruct the object/part point cloud.

**Value Map Composition.** We define the following types of value maps: affordance, avoidance, end-effector velocity, end-effector rotation, and gripper action. Each type uses a different LMP, which takes in an instruction and outputs a voxel map of shape \((100, 100, 100, k)\), where \( k \) differs for each value map (e.g., \( k = 1 \) for affordance and avoidance as it specifies cost, and \( k = 4 \) for rotation as it specifies \( \text{SO}(3) \)). We apply Euclidean distance transform to affordance maps and Gaussian filters for avoidance maps. On top of value map LMPs, we define two high-level LMPs to orchestrate their behaviors: \( p_{\text{lanner}} \) takes user instruction \( L \) as input (e.g., “open drawer”) and outputs a sequence of sub-tasks \( \ell_{1:N} \), and \( p_{\text{composer}} \) takes in sub-task \( \ell_i \) and invokes relevant value map LMPs with detailed language parameterization.

**Motion Planner.** We consider only affordance and avoidance maps in the planner optimization, which finds a sequence of collision-free end-effector positions \( p_{1:N} \in \mathbb{R}^3 \) using greedy search. Then we enforce other parametrization at each \( p \) by the remaining value maps (e.g., rotation map, velocity map). The cost map used by the motion planner is computed as the negative of the weighted sum of normalized affordance and avoidance maps with weights 2 and 1. After a 6-DoF trajectory is synthesized, the first waypoint is executed, and then a new trajectory is re-planned at 5 Hz.

**Dynamics Model.** We use the known robot dynamics model in all tasks, where it is used in motion planning for the end-effector to follow the waypoints. For the majority of our considered tasks where the “entity of interest” is the robot, no environment dynamics model is used (i.e., scene is assumed to be static), but we replan at every step to account for the latest observation. For tasks in which the “entity of interest” is an object, we study only a planar pushing model parametrized by contact point, push direction, and push distance. We use a heuristic-based dynamics model that translates an input point cloud along the push direction by the push distance. We use MPC with random shooting to optimize for the action parameters. Then a pre-defined pushing primitive is executed based on the action parameters. However, we note that a primitive is not necessary when action parameters are defined over the end-effector or joint space of the robot, which would likely yield
4.1 VoxPoser for Everyday Manipulation Tasks

We study whether VoxPoser can zero-shot synthesize robot trajectories to perform everyday manipulation tasks in the real world. Details of the environment setup can be found in Appendix A.3. While the proposed method can generalize to an open-set of instructions and an open-set of objects as shown in Fig. 1, we pick 5 representative tasks to provide quantitative evaluations in Table 1. Qualitative results including environment rollouts and value map visualizations are shown in Fig. 3.

We find that VoxPoser can effectively synthesize robot trajectories for everyday manipulation tasks with a high average success rate. Due to fast replanning capabilities, it is also robust to external disturbances, such as moving targets/obstacles and pulling the drawer open after it has been closed by the robot. We further compare to a variant of Code as Policies [75] that uses LLMs to parameterize a pre-defined list of simple primitives (e.g., `move_to_pose`, `open_gripper`). We find that compared to chaining sequential policy logic, the ability to compose spatially while considering other constraints under a joint optimization scheme is a more flexible formulation, unlocking the possibility for more manipulation tasks and leading to more robust execution.

4.2 Generalization to Unseen Instructions and Attributes

To provide rigorous quantitative evaluations on generalization, we set up a simulated environment that mirrors our real-world setup [112] but features 13 highly-randomizable tasks with 2766 unique instructions. Each task comes with a templated instruction (e.g., “push [obj] to [pos]”) that contains randomizable attributes chosen from a pre-defined list. Details are in Appendix A.4. Seen instructions/attributes may appear in the prompt (or in the training data for supervised baselines). The tasks are grouped into 2 categories, where “Object Interactions” are tasks that require interactions with objects, and “Spatial Composition” are tasks involving spatial constraints (e.g., moving slower near a particular object). For baselines, we ablate the two components of VoxPoser, LLM and motion planner, by comparing to a variant of [75] that combines an LLM with primitives and to a variant of [50] that learns a U-Net [113] to synthesize costmaps for motion planning. Table 2 shows the success rates averaged across 20 episodes per task. We find VoxPoser exhibits superior generalization in all scenarios. Compared to learned cost specification, LLMs generalize better by explicitly reasoning about affordances and constraints. On the other hand, grounding LLM knowledge in robot perception through value map composition rather than directly specifying primitive parameters offers more flexibility that generalizes beyond the prompt examples.

4.3 Efficient Dynamics Learning with Online Experiences

As discussed in Sec. 3.4, we investigate how VoxPoser can optionally benefit from online experiences for tasks that involve more intricacies of contact, such as opening doors, fridges, and windows, in a simulated environment. Specifically, we first synthesize $k$ zero-shot trajectories using VoxPoser,
<table>
<thead>
<tr>
<th>Task</th>
<th>Zero-Shot No Prior</th>
<th>w/ Prior</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Success</td>
<td>Time(s)</td>
</tr>
<tr>
<td>Door</td>
<td>6.7%±4.4%</td>
<td>58.3±4.4%</td>
</tr>
<tr>
<td>Window</td>
<td>3.3%±3.3%</td>
<td>36.7±1.7%</td>
</tr>
<tr>
<td>Fridge</td>
<td>18.3%±3.3%</td>
<td>70.0%±2.9%</td>
</tr>
</tbody>
</table>

Table 3: VoxPoser enables efficient dynamics learning by using zero-shot synthesized trajectories as prior. TLE (time limit exceeded) means exceeding 12 hours. Results are reported over 3 runs using different seeds.

4.4 Error Breakdown

In this section, we analyze the errors resulting from each component of VoxPoser and how the overall system can be further improved. We conduct experiments in simulation where we have access to ground-truth perception and dynamics model (i.e., the simulator). “Dynamics error” refers to errors made by the dynamics model\(^4\). “Perception error” refers to errors made by the perception module\(^5\). “Specification error” refers to errors made by the module specifying cost or parameters for the low-level motion planner or primitives. Examples for each method include 1) noisy prediction by the U-Net, 2) incorrect parameters specified by the LLM, and 3) incorrect value maps specified by the LLM. As shown in Fig. 4, VoxPoser achieves lowest “specification error” due to its generalization and flexibility. We also find that having access to a more robust perception pipeline and a physically-realistic dynamics model can contribute to better overall performance. This observation aligns with our real-world experiment, where most errors are from perception. For example, we find that the detector is sensitive to initial poses of objects and is less robust when detecting object parts.

5 Conclusion, Limitations, & Future Works

In this work, we present **VoxPoser**, a general framework for extracting affordances and constraints, grounded in 3D perceptual space, from LLMs and VLMs for everyday manipulation tasks in the real world, offering significant generalization advantages for open-set instructions and objects. Despite compelling results, VoxPoser has several limitations. First, it relies on external perception modules, which is limiting in tasks that require holistic visual reasoning or understanding of fine-grained object geometries. Second, while applicable to efficient dynamics learning, a general-purpose dynamics model is still required to achieve contact-rich tasks with the same level of generalization. Third, our motion planner considers only end-effector trajectories while whole-arm planning is also feasible and likely a better design choice [115–117]. Finally, manual prompt engineering is required for LLMs. We also see several exciting venues for future work. For instance, recent success of multi-modal LLMs [68, 2, 118] can be directly translated into VoxPoser for direct visual grounding. Methods developed for alignment [119, 120] and prompting [121–124] may be used to alleviate prompt engineering effort. Finally, more advanced trajectory optimization methods can be developed that best interface with value maps synthesized by VoxPoser.

---

4 LLM + Primitives [75] does not use model-based planning, thus not having a dynamics module.
5 U-Net + MP [50] maps RGB-D to costmaps using U-Net [113, 114], thus not having perception module. Errors by which are attributed to “specification error”.

---

Figure 4: Error breakdown of components. VoxPoser significantly reduces specification error.
Acknowledgments

We would like to thank Andy Zeng, Igor Mordatch, and the members of the Stanford Vision and Learning Lab for the fruitful discussions. This work was in part supported by AFOSR YIP FA9550-23-1-0127, ONR MURI N00014-22-1-2740, ONR MURI N00014-21-1-2801, ONR N00014-23-1-2355, the Stanford Institute for Human-Centered AI (HAI), JPMC, and Analog Devices. Wenlong Huang is partially supported by Stanford School of Engineering Fellowship. Ruohan Zhang is partially supported by Wu Tsai Human Performance Alliance Fellowship.

References


A Appendix

A.1 Emergent Behavioral Capabilities

Emergent capabilities refer to unpredictable phenomena that are only present in large models [125]. As VoxPoser uses pre-trained LLMs as backbone, we observe similar embodied emergent capabilities driven by the rich world knowledge of LLMs. In particular, we focus our study on the behavioral capabilities that are unique to VoxPoser. We observe the following capabilities:

- **Behavioral Commonsense Reasoning**: During a task where robot is setting the table, the user can specify behavioral preferences such as “I am left-handed”, which requires the robot to comprehend its meaning in the context of the task. VoxPoser decides that it should move the fork from the right side of the bowl to the left side.

- **Fine-grained Language Correction**: For tasks that require high precision such as “covering the teapot with the lid”, the user can give precise instructions to the robot such as “you’re off by 1cm”. VoxPoser similarly adjusts its action based on the feedback.

- **Multi-step Visual Program** [126, 127]: Given a task “open the drawer precisely by half” where there is insufficient information because object models are not available, VoxPoser can come up with multi-step manipulation strategies based on visual feedback that first opens the drawer fully while recording handle displacement, then close it back to the midpoint to satisfy the requirement.

- **Estimating Physical Properties**: Given two blocks of unknown mass, the robot is tasked to conduct physics experiments using an existing ramp to determine which block is heavier. VoxPoser decides to push both blocks off the ramp and choose the block traveling the farthest as the heavier block. Interestingly, this mirrors a common human oversight: in an ideal, frictionless world, both blocks would traverse the same distance under the influence of gravity. This serves as a lighthearted example that language models can exhibit limitations similar to human reasoning.
A.2 APIs for VoxPoser

Central to VoxPoser is an LLM generating Python code that is executed by a Python interpreter. Besides exposing NumPy [16] and the Transforms3d library to the LLM, we provide the following environment APIs that LLMs can choose to invoke:

**detect**(**obj_name**): Takes in an object name and returns a list of dictionaries, where each dictionary corresponds to one instance of the matching object, containing center position, occupancy grid, and mean normal vector.

**execute**(**movable, affordance_map, avoidance_map, rotation_map, velocity_map, gripper_map**): Takes in an “entity of interest” as “movable” (a dictionary returned by detect) and (optionally) a list of value maps and invokes the motion planner to execute the trajectory. Note that in MPC settings, “movable” and the input value maps are functions that can be re-evaluated to reflect the latest environment observation.

**cm2index**(**cm, direction**): Takes in a desired offset distance in centimeters along direction and returns 3-dim vector reflecting displacement in voxel coordinates.

**index2cm**(**index, direction**): Inverse of cm2index. Takes in an integer “index” and a “direction” vector and returns the distance in centimeters in world coordinates displaced by the “integer” in voxel coordinates.

**pointat2quat**(**vector**): Takes in a desired pointing direction for the end-effector and returns a satisfying target quaternion.

**set_voxel_by_radius**(**voxel_map, voxel_xyz, radius_cm, value**): Assigns “value” to voxels within “radius_cm” from “voxel_xyz” in “voxel_map”.

**get_empty_affordance_map**(): Returns a default affordance map initialized with 0, where a high value attracts the entity.

**get_empty_avoidance_map**(): Returns a default avoidance map initialized with 0, where a high value repulses the entity.

**get_empty_rotation_map**(): Returns a default rotation map initialized with current end-effector quaternion.

**get_empty_gripper_map**(): Returns a default gripper map initialized with current gripper action, where 1 indicates “closed” and 0 indicates “open”.

**get_empty_velocity_map**(): Returns a default affordance map initialized with 1, where the number represents scale factor (e.g., 0.5 for half of the default velocity).

**reset_to_default_pose**(): Reset to robot rest pose.
A.3 Real-World Environment Setup

We use a Franka Emika Panda robot with a tabletop setup. We use Operational Space Controller with impedance from Deoxys [128]. We mount two RGB-D cameras (Azure Kinect) at two opposite ends of the table: bottom right and top left from the top down view. At the start of each rollout, both cameras start recording and return the real-time RGB-D observations at 20 Hz.

For each task, we evaluate each method on two settings: without and with disturbances. For tasks with disturbances, we apply three kinds of disturbances to the environment, which we pre-select a sequence of them at the start of the evaluation: 1) random forces applied to the robot, 2) random displacement of task-relevant and distractor objects, and 3) reverting task progress (e.g., pull drawer open while it’s being closed by the robot). We only apply the third disturbances to tasks where “entity of interest” is an object or object part.

We compare to a variant of Code as Policies [75] as a baseline that uses an LLM with action primitives. The primitives include: move_to_pos, rotate_by_quat, set_vel, open_gripper, close_gripper. We do not provide primitives such as pick-and-place as they would be tailored for a particular suite of tasks that we do not constrain to in our study (similar to the control APIs for VoxPoser specified in Sec. A.2).

A.3.1 Tasks

Move & Avoid: “Move to the top of [obj1] while staying away from [obj2]”, where [obj1] and [obj2] are randomized everyday objects selected from the list: apple, banana, yellow bowl, headphones, mug, wood block.

Set Up Table: “Please set up the table by placing utensils for my pasta”.

Close Drawer: “Close the [deixis] drawer”, where [deixis] can be “top” or “bottom”.

Open Bottle: “Turn open the vitamin bottle”.

Sweep Trash: “Please sweep the paper trash into the blue dustpan”.


A.4 Simulated Environment Setup

We implement a tabletop manipulation environment with a Franka Emika Panda robot in SAPIEN [112]. The controller takes as input a desired end-effector 6-DoF pose, calculates a sequence of interpolated waypoints using inverse kinematics, and finally follows the waypoints using a PD controller. We use a set of 10 colored blocks and 10 colored lines in addition to an articulated cabinet with 3 drawers. They are initialized differently depending on the specific task. The lines are used as visual landmarks and are not interactable. For perception, a total of 4 RGB-D cameras are mounted at each end of the table pointing at the center of the workspace.

A.4.1 Tasks

We create a custom suite of 13 tasks shown in Table 4. Each task comes with a templated instruction (shown in Table 4) where there may be one or multiple attributes randomized from the pre-defined list below. At reset time, a number of objects are selected (depending on the specific task) and are randomized across the workspace while making sure that task is not completed at reset and that task completion is feasible. A complete list of attributes can be found below, divided into “seen” and “unseen” categories:

Seen Attributes:

- [pos]: [“back left corner of the table”, “front right corner of the table”, “right side of the table”, “back side of the table”]
- [obj]: [“blue block”, “green block”, “yellow block”, “pink block”, “brown block”]
- [preposition]: [“left of”, “front side of”, “top of”]
- [deixis]: [“topmost”, “second to the bottom”]
- [dist]: [3, 5, 7, 9, 11]
- [region]: [“right side of the table”, “back side of the table”]
- [velocity]: [“faster speed”, “a quarter of the speed”]
- [line]: [“blue line”, “green line”, “yellow line”, “pink line”, “brown line”]

Unseen Attributes:

- [pos]: [“back right corner of the table”, “front left corner of the table”, “left side of the table”, “front side of the table”]
- [obj]: [“red block”, “orange block”, “purple block”, “cyan block”, “gray block”]
- [preposition]: [“right of”, “back side of”]
- [deixis]: [“bottommost”, “second to the top”]
- [dist]: [4, 6, 8, 10]
- [region]: [“left side of the table”, “front side of the table”]
- [velocity]: [“slower speed”, “3x speed”]
- [line]: [“red line”, “orange line”, “purple line”, “cyan line”, “gray line”]
### A.4.2 Full Results on Simulated Environments

<table>
<thead>
<tr>
<th>Tasks</th>
<th>U-Net + MP</th>
<th>LLM + Prim.</th>
<th>VoxPoser</th>
</tr>
</thead>
<tbody>
<tr>
<td>move to the [preposition] the [obj]</td>
<td>95.0% 0.0%</td>
<td>85.0% 60.0%</td>
<td>90.0% 55.0%</td>
</tr>
<tr>
<td>move to the [pos] while staying on the [preposition] the [obj]</td>
<td>100.0% 10.0%</td>
<td>80.0% 30.0%</td>
<td>95.0% 50.0%</td>
</tr>
<tr>
<td>move to the [pos] while moving at [velocity] when within [dist]cm from the obj</td>
<td>80.0% 0.0%</td>
<td>10.0% 0.0%</td>
<td>100.0% 95.0%</td>
</tr>
<tr>
<td>close the [deixis] drawer by pushing</td>
<td>0.0% 0.0%</td>
<td>60.0% 60.0%</td>
<td>80.0% 80.0%</td>
</tr>
<tr>
<td>push the [obj] along the [line]</td>
<td>0.0% 0.0%</td>
<td>0.0% 0.0%</td>
<td>65.0% 30.0%</td>
</tr>
<tr>
<td>grasp the [obj] from the table at [velocity]</td>
<td>35.0% 0.0%</td>
<td>75.0% 70.0%</td>
<td>65.0% 40.0%</td>
</tr>
<tr>
<td>drop the [obj] to the [pos]</td>
<td>70.0% 10.0%</td>
<td>60.0% 100.0%</td>
<td>60.0% 100.0%</td>
</tr>
<tr>
<td>push the [obj] while letting it stay on [region]</td>
<td>0.0% 5.0%</td>
<td>10.0% 0.0%</td>
<td>50.0% 50.0%</td>
</tr>
<tr>
<td>move to the [region]</td>
<td>5.0% 0.0%</td>
<td>100.0% 95.0%</td>
<td>100.0% 100.0%</td>
</tr>
<tr>
<td>move to the [pos] while staying at least [dist]cm from the [obj]</td>
<td>0.0% 0.0%</td>
<td>15.0% 20.0%</td>
<td>85.0% 90.0%</td>
</tr>
<tr>
<td>move to the [pos] while moving at [velocity] in the [region]</td>
<td>0.0% 0.0%</td>
<td>90.0% 45.0%</td>
<td>85.0% 85.0%</td>
</tr>
<tr>
<td>push the [obj] to the [pos] while staying from [obstacle]</td>
<td>0.0% 0.0%</td>
<td>0.0% 10.0%</td>
<td>45.0% 55.0%</td>
</tr>
<tr>
<td>push the [obj] to the [pos]</td>
<td>0.0% 0.0%</td>
<td>20.0% 25.0%</td>
<td>80.0% 75.0%</td>
</tr>
</tbody>
</table>

**Table 4:** Full experimental results in simulation on seen tasks and unseen tasks. “SA” indicates seen attributes and “UA” indicates unseen attributes. Each entry represents success rate averaged across 20 episodes.
A.5 Prompts

Prompts used in Sec. 4.1 and Sec. 4.2 can be found below.

**planner**: Takes in a user instruction $\mathcal{L}$ and generates a sequence of sub-tasks $\ell_i$ which is fed into "composer" (Note that planner is not used in simulation as the evaluated tasks consist of a single manipulation phase).

real-world: voxposer.github.io/prompts/real_planner_prompt.txt.

**composer**: Takes in sub-task instruction $\ell_i$ and invokes necessary value map LMPs to compose affordance maps and constraint maps.

simulation: voxposer.github.io/prompts/sim_composer_prompt.txt.
real-world: voxposer.github.io/prompts/real_composer_prompt.txt.

**parse_query_obj**: Takes in a text query of object/part name and returns a list of dictionaries, where each dictionary corresponds to one instance of the matching object containing center position, occupancy grid, and mean normal vector.

real-world: voxposer.github.io/prompts/real_parse_query_obj_prompt.txt.

**get_affordance_map**: Takes in natural language parametrization from composer and returns a NumPy array for task affordance map.

simulation: voxposer.github.io/prompts/sim_get_affordance_map_prompt.txt.
real-world: voxposer.github.io/prompts/real_get_affordance_map_prompt.txt.

**get_avoidance_map**: Takes in natural language parametrization from composer and returns a NumPy array for task avoidance map.

simulation: voxposer.github.io/prompts/sim_get_avoidance_map_prompt.txt.
real-world: voxposer.github.io/prompts/real_get_avoidance_map_prompt.txt.

**get_rotation_map**: Takes in natural language parametrization from composer and returns a NumPy rotation map.

simulation: voxposer.github.io/prompts/sim_get_rotation_map_prompt.txt.
real-world: voxposer.github.io/prompts/real_get_rotation_map_prompt.txt.

**get_gripper_map**: Takes in natural language parametrization from composer and returns a NumPy array for gripper action map.

simulation: voxposer.github.io/prompts/sim_get_gripper_map_prompt.txt.
real-world: voxposer.github.io/prompts/real_get_gripper_map_prompt.txt.

**get_velocity_map**: Takes in natural language parametrization from composer and returns a NumPy array for end-effector velocity map.

simulation: voxposer.github.io/prompts/sim_get_velocity_map_prompt.txt.
real-world: voxposer.github.io/prompts/real_get_velocity_map_prompt.txt.