A Appendix

A.1 APIs for VoxPoser

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

`detect(*obj_names)`: Takes in a (list of) object name(s) 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 "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 functionals that can be invoked to reflect the latest environment observation.

`index2cm(index, direction)`: Takes in an integer 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 "radious_cm" from "voxel_xyz".

`get_empty_affordance_map()`: Returns a default affordance map initialized with 0.

`get_empty_avoidance_map()`: Returns a default affordance map initialized with 0.

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

`get_empty_gripper_map()`: Returns a default affordance map initialized with current gripper action.

`get_empty_velocity_map()`: Returns a default affordance map initialized with 1.

`reset_to_default_pose()`: Reset to rest pose.

A.2 Real-World Environment Setup

We use a Franka Emika Panda robot with a tabletop setup. We use Operational Space Controller with impedance from Deoxys [114]. 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.

We compare to a variant of Code as Policies [71] as a baseline that 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).

A.2.1 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.3 Simulated Environment Setup

We implement a tabletop manipulation environment with a Franka Emika Panda robot in SAPIEN [115]. 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.3.1 Tasks

We create a custom suite of 13 tasks shown in Table 4. Each task comes with a templated instruction 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.3.2 Full Results on Simulated Environments

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

Table 4: Full experimental results in simulation. "SA" indicates seen attributes and "UA" indicates unseen attributes.

A.4 Prompts

`planner`: Takes in a user instruction $\mathcal{L}$ and generates a sequence of sub-tasks ℓ_i which is fed into "composer".

simulation: voxposer-anon.github.io/sim_planner_prompt.txt.

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

`composer`: Takes in sub-task instruction ℓ_i and invokes necessary value map LMPs to compose affordance maps and constraint maps.

simulation: voxposer-anon.github.io/sim_composer_prompt.txt.

real-world: voxposer-anon.github.io/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.

simulation: voxposer-anon.github.io/sim_parse_query_obj_prompt.txt.

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

`get_target_map`: Takes in natural language parametrization from composer and returns a NumPy array for task affordances.

simulation: voxposer-anon.github.io/sim_get_target_map_prompt.txt.

real-world: voxposer-anon.github.io/real_get_target_map_prompt.txt.

`get_avoidance_map`: Takes in natural language parametrization from composer and returns a NumPy array for end-effector rotation.

simulation: voxposer-anon.github.io/sim_get_avoidance_map_prompt.txt.

real-world: voxposer-anon.github.io/real_get_avoidance_map_prompt.txt.

`get_rotation_map`: Takes in natural language parametrization from composer and returns a NumPy array for end-effector rotation.

simulation: voxposer-anon.github.io/sim_get_rotation_map_prompt.txt.

real-world: voxposer-anon.github.io/real_get_rotation_map_prompt.txt.

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

simulation: voxposer-anon.github.io/sim_get_gripper_map_prompt.txt.

real-world: voxposer-anon.github.io/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.

simulation: voxposer-anon.github.io/sim_get_velocity_map_prompt.txt.

real-world: voxposer-anon.github.io/real_get_velocity_map_prompt.txt.