Scaling Up and Distilling Down: Language-Guided Robot Skill Acquisition

Anonymous Author(s)
Affiliation
Address
email

Abstract: We present a framework for robot skill acquisition, which 1) efficiently scale up data generation of language-labelled robot data and 2) effectively distills this data down into a robust multi-task language-conditioned visuo-motor policy. For (1), we use a large language model (LLM) to guide high-level planning, and sampling-based robot planners (*e.g.* motion or grasp samplers) for generating diverse and rich manipulation trajectories. To robustify this data-collection process, the LLM also infers a code-snippet for the success condition of each task, simultaneously enabling the data-collection process to detect failure and retry as well as the automatic labeling of trajectories with success/failure. For (2), we extend the diffusion policy single-task behavior-cloning approach to multi-task settings with language conditioning. Finally, we propose a new multi-task benchmark with 18 tasks across five domains to test long-horizon behavior, common-sense reasoning, tool-use, and intuitive physics. We find that our distilled policy successfully learned the robust retrying behavior in its data collection procedure, while improving absolute success rates by 33.2% on average across five domains. All code, data, and qualitative policy results are available at this anonymized website.

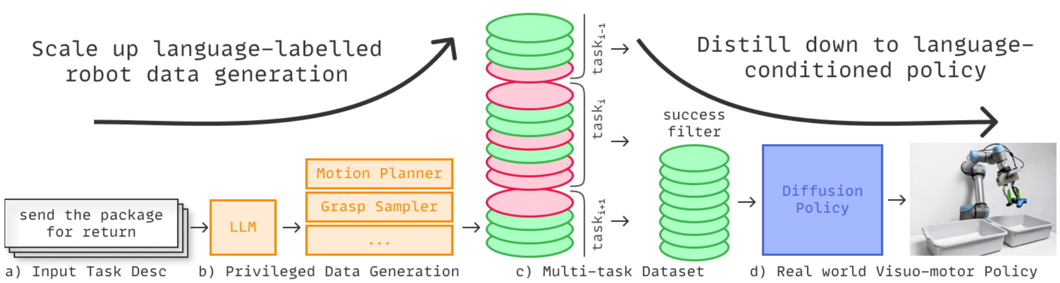

Figure 1: **Language-guided Skill Acquisition** enables scalable robot learning. In the data generation stage, a LLM takes as input task descriptions (a) and uses sampling-based robotic planners and privileged simulation information (b) to perform task-directed exploration. This enables the scaling up of language and task-success labeled dataset generation (c). In the second stage, the dataset is filtered for success and distilled down into a closed-loop language-conditioned visuomotor policy for real world deployment (d).

1 Introduction

How can we scalably acquire robust, reusable, real-world manipulation skills? This question has been the driving force behind extensive research in robot learning. Attempts in the field have focused on two primary aspects: First, how to **scale up the data collection** for a diverse range of manipulation skills, which involves efforts such as improving the hardware [1, 2] and software [3, 4] which support demonstration collection, utilization of non-robotics datasets [5, 6], or trial-and-error explorations [7]. The second aspect of this question concerns **effective learning** from the collected data, which delves into exploring effective action representations [8–10] and policy formulations [11, 12] that can robustly model the training data and generalize to novel scenarios.

This paper proposes a new framework that provides a comprehensive solution for both aspects by leveraging language guidance, while using no expert demonstrations or reward specification/engineering. We contribute two key components with our framework:

- **Scaling Up Language-Guided Data Generation:** Our data-collection policy is a large language model (LLM) which has access to a suite of 6DoF exploration primitives (*i.e.*, sampling-based robot planners and utilities). Given an input task description, this policy first **simplifies** the task by recursively decomposing it into subtasks, resulting in a hierarchical plan (*i.e.*, task tree). Next, this plan is **grounded** into a sequence

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

of 6DoF exploration primitives, which generates diverse robot trajectories for the task. Finally, the data collection policy **verifies** the trajectories' success with an inferred success function and **retries** the task until it succeeds. This verify & retry step not only improves the data-collection policy's success, but also adds robot experience on how to recover from failure, an important trait for downstream policy distillation. This data generation approach is scalable, enabling significantly more efficient autonomous task-directed exploration than unguided alternatives (*i.e*., reinforcement learning) while not being limited by the lack of low-level understanding of the LLM-only solution.

- **Distilling Down to Language-Conditioned Visuomotor Policy:** We distill these robot experiences into a visuo-linguo-motor policy that infers control sequences from visual observations and a natural language task description. To enable effective learning of high entropy, diverse robot trajectories, we extend the diffusion policy [12] to handle language-based conditioning for multi-task learning. This allows the learned policy to be reused and recomposed through language-based planners. We found that our distilled policy successfully learned the robust retrying behavior from its data collection policy, while improving upon its absolute success rate across five domains by 33.2%. Further, we demonstrate that our policy directly transfers to the real-world without fine-tuning using domain randomization.

Our framework combines these two components to get the best of both worlds – leverage LLM's common-sense reasoning abilities for efficient exploration while learning robust and re-usable 6DoF skills for real-world deployment. In summary, the key contribution of this paper is a new framework for visuo-linguo-motor policy learning that is enabled by three novel components:

- A new language-guided data collection framework that combines language-based task planner with 6DoF robot utilities (*e.g*. motion planning, grasp sampling).
- New formulation of diffusion-based policy that effectively learns multi-task language-conditioned closed-loop control policies.
- In addition to our algorithmic contributions, we also contribute a new multi-task benchmark that includes 18 tasks across five domains, requiring long-horizon (≈ 800 control cycles), common sense, tool-use, and intuitive physics understanding – capabilities lacking in existing manipulation benchmarks.

2 Related Works

Scaling visuo-linguo-motor data. In learning vision-and-language-conditioned motor policies for real-world deployment [9, 10, 13–18], one of the most important questions is how to scale up "robot-complete data" – data that has robot sensory inputs (*e.g*. vision), action labels (*e.g*. target end-effector & gripper commands), and task labels (*e.g*. language description, success). The most prevalent paradigm is to use humans to annotate both actions (*e.g*. teleoperation) and language [9, 10, 13–18]. When providing action labels, humans can either provide task-specific [9, 10, 15, 18], or task-agnostic ("play") data [13, 14, 16, 19]. A primary limitation, however, is that data scalability is human-limited.

Other prior works have proposed strategies to enable more-autonomously-scalable data. To scale language annotation, prior works study using visual-language models [20, 21], or procedurally post-hoc provided in simulation [19]. To scale action labels, methods study how to use *autonomous sub-optimal policies* from random [7] to learned [22] policies. Human egocentric videos [6, 23, 24] has also been shown to be relevant to robot learning [5, 25], but *is not robot-complete* (lacks action labels), and requires cross-embodiment transfer. Towards unsupervised exploration, prior works have also investigated evolving environments [26, 27] and embodiments [28], automatic task generation [29], leveraging language guidance [30, 31] and world-model error [32], but have not been demonstrated to scale to 6 DoF robotic skill learning. While these approaches reduce human efforts, they are still limited in optimality, generality, and/or completeness of robot data labels.

Another option for the autonomous data collection policy is to use a model-based policy, *e.g*. task and motion planning (TAMP) [33]. Our approach extends such methods in terms of flexibility and task generality by leveraging LLM's common-sense knowledge. However, in contrast to recent works which use LLMs as the *final* policy [34–40], we use the LLM-based planner as a suboptimal *data-collection* policy. We then distill only successful trajectories into an observable-information [41–43] policy, allowing the distilled policy to improve upon its LLM data collection policy's performance.

Policy Representations and Multi-task Policy Distillation. One primary question in visuo-motor learning [44] has been how to represent the policy for effective learning, i.e. to enable high precision, multi-modal robot behavior [2, 11, 12, 45, 46]. Another related question has been how to best train multi-task policies [47, 48], including those conditioned on language [9, 10, 13, 15, 16, 18]. Our work presents the novel formulation of bringing diffusion-based [49, 50] policies [12] into the language-conditioned [51, 52] visuomotor domain. Additionally, prior works in multi-task language-conditioning typically focus on cloning policies from experts, meanwhile we study distilling data from a success-filtered suboptimal policy. Success-filtering [11, 53] can be viewed as the simplest form of offline RL [54].

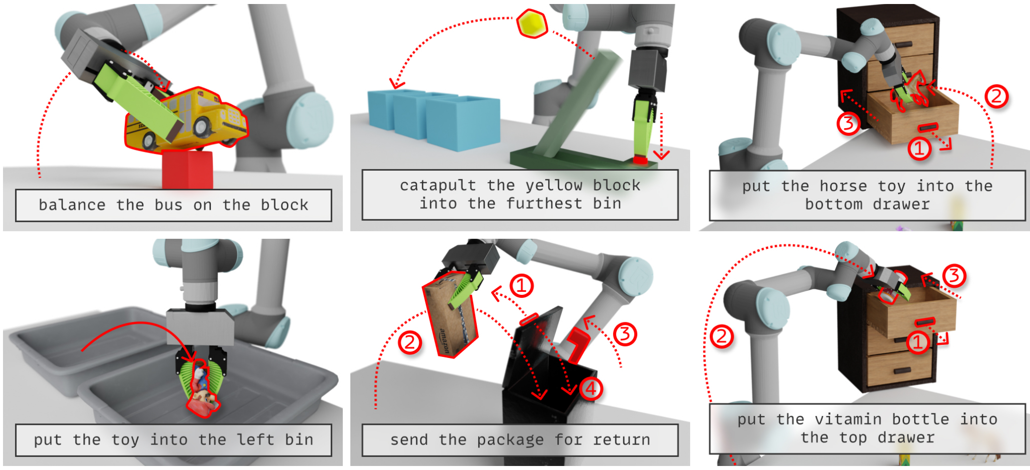

Figure 2: **Benchmark.** We validate our approach on a new multi-task benchmark addressing challenging long-horizon tasks (*i.e.*, 800 control cycles) requiring language understanding (e.g., put [object] to [top] drawer), common sense knowledge (e.g., send a package for return requires raising the mailbox flag), tool-use (e.g., catapult), and intuitive physics (e.g., balance the bus). The tasks are best viewed on our this anonymized website.

3 Approach

We propose a new framework for robot learning that performs automatic data collection and policy learning from only a task description. Our design is grounded on four key observations:

- We recognize the importance of random exploration in reinforcement learning, but aim to not be constrained by its inefficiency for long-horizon, sparse reward tasks.
- We acknowledge the usefulness of LLM's common-sense and zero-shot capabilities, but believe language *is not by itself* the ideal representation for robust, rich, and precise robotic manipulation.
- We are inspired by the effectiveness of robotic planning methods, e.g. TAMP, but wish to be flexible to novel tasks and domains and non-reliant on ground truth state during policy inference.
- We aim to achieve the simplicity and effectiveness of behavior cloning in distilling collected robot experience into a policy for real-world deployment, while side-stepping the requirement for costly human demonstrations or play data collection.

Using no human demonstration or manually specified reward, our framework combines the strengths of these four areas into a unified framework for both efficient task-directed exploration and multi-task visuo-linguo-motor policy learning.

Method Overview. In the data generation phase, we use an LLM to recursively decompose (§3.1) tasks into a hierachical plan (*i.e.*, task tree) for exploration and ground the plan into sampling-based robot utilities and motion primitives (§3.2). Next, the LLM infers success-detection functions for each task in the plan (§3.3), providing success-labeling. This autonomous data generation process outputs a replay buffer of task-directed exploration experience, labeled with language descriptions and success labels. In the training phase (§3.4), we filter this data for success according to the LLM inferred success condition and distill it into a multi-task vision-and-language-conditioned diffusion policy [12].

3.1 Simplify: Task Planning and Decomposition

Given a task description, the first step is to generate a high-level task plan. To improve the flexibility to work with any tasks and 3D assets, we opted for an LLM-based planner to leverage their common-sense and zero-shot reasoning skills. Unlike classical TAMP planners, our framework does not require domain-specific engineering and transition function design to work with new tasks.

Concretely, our recursive LLM planner takes as input the task description, the simulation state, and outputs a plan in the form of a task tree (Fig. 3a). To do so, the LLM first checks whether the task description involves the robot interacting with multiple or only one object. For instance, "move the package into the mailbox" involves opening the mailbox before picking up the package and putting the mailbox in, and should be considered a multi-object task. Meanwhile, "with the mailbox opened, move the package into the mailbox" should be a single-object task. For the base case of single-object tasks, we prompt the LLM to which object part name to to interact. For the case of multi-object tasks, we prompt the LLM to decompose the task into subtasks, and recurse down each subtask.

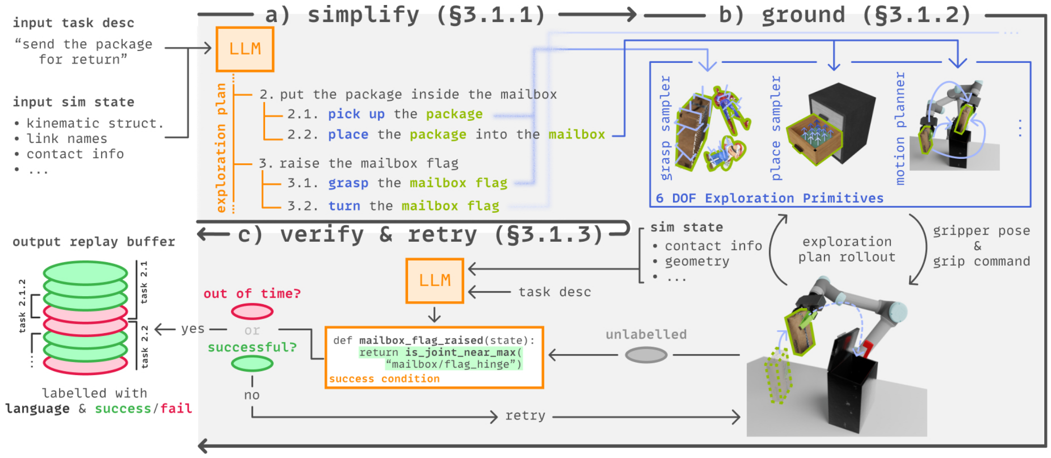

Figure 3: **Language-Driven Robot Data Generation** takes as input the task description and simulation state, and outputs a replay buffer, labelled with language descriptions and success. It starts by using an LLM to simplify tasks recursively (a) until the task involves only one object, resulting in a hierarchical exploration plan. Next, the plan is grounded (b) into a sequence of 6 DOF exploration primitives (*e.g.* grasp samplers, motion planners, etc.) and rolled out in simulation to give an unlabelled robot trajectory. Finally, an LLM infers a success function code-snippet, and uses it to verify (c) and label it with succeeded or failed. If the trajectory failed, the LLM retries the exploration plan with a different random seed (*e.g.* a different grasp pose from the grasp sampler). If the robot succeeds or run out of time, the labeled trajectory is returned.

3.2 Ground: Compiling a Plan into Robot Utilities

With the generated task tree §3.1, the next step is to ground the high-level plan into physical actions. Here, the choice of the *low-level robot API* critically defines the system's capability and, therefore, becomes a key differentiating factor between different systems. In principle, there are three desired properties we want to see in the action space design:

- **Flexibility.** Planar actions [10, 37] aren't flexible enough to manipulate prismatic and revolute joints.
- **Scalable.** Namely, actions should not require human demonstrations to acquire [9, 10, 13–16, 35].
- **Language-friendly.** While joint sequences can encode any action, it is not language-friendly.

We propose to ground the LLM's plan with API calls into a set of robot utility functions, which include a sampling-based motion planner, a geometry-based grasp and placement sampler, and motion primitives for articulated manipulation. We refer to these utilities as 6 DOF Exploration Primitives (Fig 3b) because, by virtue of being *pseudo-random*, the sampling-based utilities generate *diverse* robot trajectories, enabling effective exploration for rich 6 DoF manipulation settings. For instance, our grasp and placement samplers samples uniformly amongst all points in the object part's point cloud to find good grasps and placements poses, respectively, which are used as input into a rapidly-exploring random trees [55] motion planner that samples uniformly in joint space. This results in diverse grasps, placements, and motion trajectories connecting grasps and placements.

For each leaf node in the inferred task tree (§ 3.1), the grounding process takes as input the node's task description (*e.g.* "open the mailbox"), its associated object part name (*e.g.* "mailbox lid"), and the simulation state, and outputs a sequence of 6 DoF Exploration Primitive API calls. Using the object part name, we can parse the object's kinematic structure from the simulation state and handle articulated and non-articulated (*i.e.*, rigid, deformable) objects separately. For non-articulated objects, the LLM is prompted to choose the pick & place object names, used to sample grasp and placement pose candidates. For articulated objects (with either revolute or prismatic joints), the leaf node's associated object part name is used to sample a grasp candidate followed by a rotation or translation primitive conditioned on its joint parameters (i.e., joint type, axis, and origin).

Exploration Plan Rollout. Each node in the exploration plan is grounded only when it is being executed, where the order of execution follows a pre-order tree traversal. By keeping track of the subtask's state, sub-segments of robot trajectory can be labelled with the subtask's description, thereby providing **dense and automatic text labels** for the trajectory. For instance, all actions taken during the inferred subtask "open the mailbox" can be labeled with both the subtask's description "open the mailbox" and the root task description "move the package into the mailbox".

Since grounding happens only when a task node is visited, each node's grounding process is independent of the other leaf nodes, depending only on the simulation state when it is evaluated. While this simplifies planning significantly, it also means that failed execution can occur. For instance, a grasp candidate may render all placement candidates infeasible.

3.3 Verify & Retry: Robustifying the Data Collection Policy

Recall, the planning and grounding step can fail, especially when we consider long-horizon tasks. To address this, we propose a verify & retry (Fig. 3c) scheme, which uses environment feedback to detect failed execution.

Verify. For each task, the LLM infers **a success function code snippet** given the task description, simulation state, and API functions to for query simulation state (e.g., checking contact or joint values, etc). This amounts to prompting the LLM to complete a task success function definition that outputs a boolean value, indicating task success. For instance, given the task "raise the mailbox flag", the LLM's inferred code snippet should check whether the mailbox's flag hinge is raised (Fig. 3c, highlighted green).

Retry. When a trajectory is labeled failed, the robot retries the same sequence of robot utilities with a different random seed (*i.e.*, for the sampling-based robotic utilities) without resetting the simulation state until the task succeeds. For instance, in the bus balance task (Fig. 2, top left), the robot would repeatedly try different grasp and place candidates until the bus is balanced. In the tree traversal process § 3.2, nodes only yield execution to its parent task when the node's inferred success condition returns true. This design not only leads to higher success rates in data generation but also provides useful demonstrations on **how to recover from failure**. In the output replay buffer, the only failed trajectories are ones which timed-out or led to invalid states (*e.g.* object dropped on the floor).

3.4 Language-conditioned Policy Distillation

We extend diffusion policy [12], a state-of-the-art approach for single-task behavior cloning, to the multi-task domain by adding language-conditioning. This policy takes as input a task description CLIP [56] feature, proprioception history, and visual observations, and outputs a sequence of end effector control commands. Following Robomimic [4]'s findings, we use a wrist-mounted view in addition to a global (workspace) view to help with tasks requiring precise manipulation. We use their ResNet18-based [57] vision encoders, one for each view. We found that using only the latest visual observation along with the full observation horizon of proprioception maintains the policy's high performance while reducing training time. When used in conjunction with the DDIM [58] noise scheduler, we found that we could use a $10\times$ shorter diffusion process at inference (5 timesteps at inference, 50 timesteps at training) while retaining a comparable performance. Quantitatively, when using a 10 dimensional action space*, our policy can be run at $\approx 35Hz$ on an NVIDIA RTX3080.

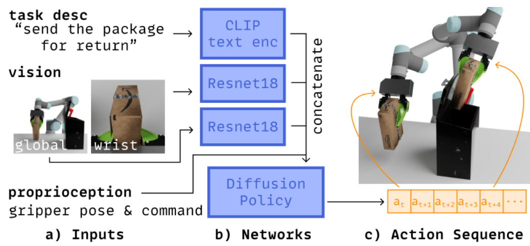

Figure 4: **Language-Conditioned Policy Distillation.** The policy takes as input a task description, two RGB camera views, and gripper proprioception data, and outputs a sequence of gripper poses and closing command.

4 Evaluation

Our experiments try to validate two questions: 1) Can our data generation approach efficiently perform task-directed exploration? 2) Can our policy learning approach effectively distill a multi-modal, multi-task dataset into a generalizable and robust visuo-linguo-motor policy?

Domain	Complex geometry	Articulation	Common sense	Tool use	Multi-task	Long horizon
Balance	✗	✗	✗	✗	✗	✗
Catapult	✗	✓	✓	✓	✓	✗
Transport	✓	✗	✗	✗	✗	✗
Mailbox	✗	✓	✓	✗	✗	✓
Drawer	✓	✓	✗	✗	✓	✓

Table 1: **Benchmark Suite.**

Our Benchmark contains 18 tasks across 5 domains (Fig. 2 Tab. 1), with the following properties:

- **6DoF & articulated manipulation**, for deadling with complex object geometry and articulation.
- **Geometry Generalization.** In our bin transport domain, the robot must generalize its bin transport skill to unseen object instances, with novel shapes, sizes, and colors.
- **Intuitive physics.** Robots should understand the physical properties of the world and use this knowledge to perform tasks. In the bus balance domain, the robot needs to learn the precise grasping and placement to balance a large bus toy on a small block. In the catapult domain, where the block is placed along a catapult arm determines how far the block will be launched, and, thus, which bin (if any) the block will land in.
- **Common-sense reasoning & Tool-use.** Natural language task description is user-friendly but often under-specifies the task. Common-sense can help to fill in the gaps. In the mailbox domain, given the task "send the package for return", the robot should understand that it not only needs put the package inside, but also raise the mailbox flag to indicate that the package is ready for pickup. In the catapult domain, the robot needs to understand that pressing the catapult's button will activate the catapult, and that the block needs to be placed on the catapult arm to be launched.

*3 for position, 6 for rotation using the upper rows of the rotation matrix, and a gripper close command

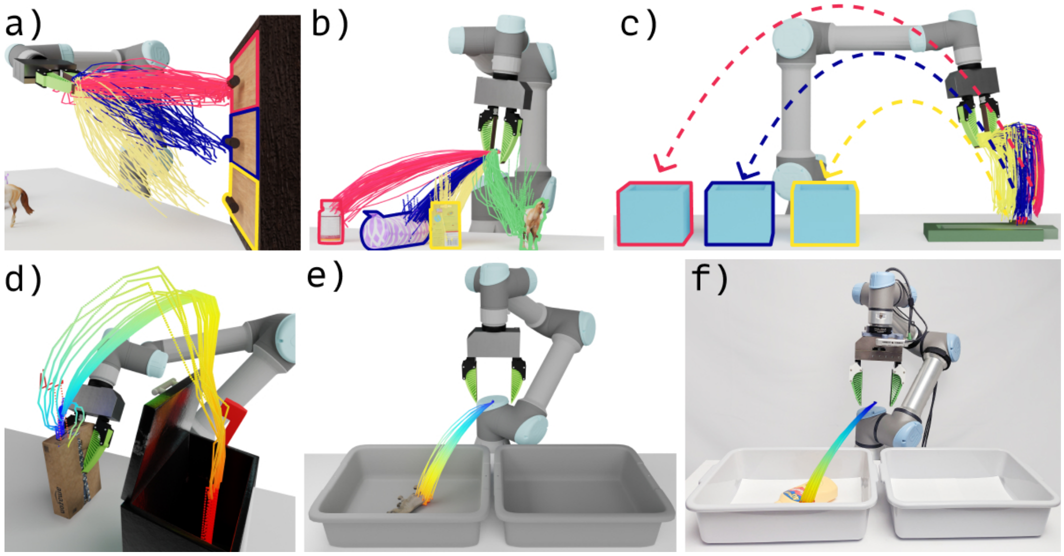

Figure 5: **High Entropy yet Precise Language-Guided Action Sequences.** Running the pseudorandom language-conditioned diffusion process with different seeds on the same observations yields language-consistent (a-c, different colors for different task descriptions), high entropy actions when possible (a-f, object grasping, transports, & placements) and precise actions when necessary (d, narrow mailbox with large package). Further, domain randomization enables a simulation trained policy (e) to generalize to the real world (f).

- **Multi-task conditioning.** Given the same visual observations but different task description, the robot should perform different and task-relevant actions. The catapult domain has 3 tasks for three target bins, and the drawer domain has 12 tasks.
- **Long horizon behaviour.** Our longest horizon domain, mailbox, takes at least 4 subtasks to complete (open the mailbox, put the package in the mailbox while its opened, close the mailbox, then raise the mailbox flag) which can require up to 800 control cycles. In the drawer domain, the robot needs to open the drawer, move the object into the drawer, then close it, which takes about 300 control cycles.

The benchmark is built on top of the MuJoCo [3] simulator, using assets from the Google Scanned dataset [59, 60]. We use a table-top manipulation set-up with a 6DoF robot arm. The task success in evaluation is a manually designed function, instead of LLM generated function used for data collection.

Metrics. We report the success rates (%) averaged over 200 episodes in Table 2, a task completion efficiency plot in Fig. 6, and qualitative results in Fig. 5. If a domain has multiple tasks then we report the average performance of all tasks. We also compare different LLMs in Table 4 (10 samples per task) and investigate the sources of error in our system for the mailbox domain in Table 3 (200 trials per execution).

Data Generation Baselines. Code-as-Policy [37] is a state-of-the-art approach for using an LLM directly as a robot policy by making state (*e.g.* query present objects) and action primitive API calls to a robot. Given an LLM-inferred code string, they execute the snippet in an open-loop fashion. Crucially, in their table top manipulation setting, they assume access to planar action primitives. Thus, we introduce the following baselines, which build on top of Code-as-Policy and each other as follows:

- **LLM-as-Policy (2D)**: Similar to code-as-policy using planar pick-and-place, but we use ground truth object segmentation instead of their off-the-shelf object detectors [61, 62].
- **(+) 6 DOF robot utils**: Builds on top of the previous baseline by adding access to 6 DOF robot utilities for grasping, placement, motion planning, and articulated manipulation.
- **(+) Verify & Retry**: Adding to the previous baselines, this baseline uses the LLM's predicted success condition to label trajectories and retry failed ones. Since the robot utilities involve pseudo-random samplers (*e.g.* RRT, grasp sampling), retrying the task means running these samplers again using the pseudo-random state and environment state from where failed trajectory left it. Since we use this approach as our data generation policy, it also serves as an ablation of our approach.

Policy Distillation Ablations. We compare against BC-Z [15]'s single-task policies which does not use FiLM conditioning (used in their bin emptying and door opening tasks). To understand the effects of our policy learning design decisions in the single-task regime, we fix training time and dataset size (2 days using at least 500 successful trajectories), and provide the following ablations:

- **Action Generation**: Instead of using diffusion processes conditioned on the policy input embedding to decode actions, it is typical use multi-layer perceptrons. Following Jang et al. [15], we use one **MLP** with two hidden layers and ReLU activations for end effector position, one for the orientation, and another for gripper command. This standard policy architecture is deterministic, and is trained with mean-squared error loss for pose and binary cross entropy loss for gripper command.

- **Action Space**: Besides our absolute end effector pose action space, **Delta-Action** and velocity control spaces is another popular action space choice [4, 15, 63–65]. We also ablate BC-Z's execution action horizon (Exec) while keeping their original prediction horizon (Pred).
- **Observation Encoder**: All approaches encode images using a ResNet18 [57] architecture. Although the original architecture was designed with an average pooling layer, its typical for robotic policies to use a spatial softmax pooling [44] layer instead.
- **Data usage**: **No-Retry** trains on successful trajectories generated from the data generation approach without Verify & Retry, so it does not observe any recovery behavior.

4.1 Data Collection Policy Evaluation

6DoF exploration is critical. First, we verify different approach's ability to perform and explore in 6DoF, which is crucial for general manipulation. When 6DoF exploration is introduced, we first observe a drop in the average success rate for simple tasks that could be accomplished with planar actions (Balance, Transport, Tab. 2). However, this ability is critical for exploring complex tasks, providing data to improve upon in the later distilling stage. In particular, we observed that 6DoF actions are important for grasping diverse objects with complex geometry (Transport, Tab. 2), and manipulating articulated objects (Drawer, Mailbox, Tab. 2).

Approach	Planar			6DoF		Average
	Balance	Catapult	Transport	Mailbox	Drawer	
LLM-as-Policy (2D)	28.0	**33.3**	21.5	0.0	0.0	27.6
(+) 6DoF Robot Utils	5.5	2.5	35.0	0.0	1.3	8.8
(+) Verify & Retry	**45.0**	7.3	**82.0**	**3.0**	**31.8**	**33.8**
Distill No Retry	67.5	38.5	32.5	0.0	22.7	32.2
Distill Ours	**79.0**	**58.3**	**80.0**	**62.0**	**55.8**	**67.0**

Table 2: **Success Rates (%)** for data generation (top) and distillation approaches (bottom) over 200 trials.

Moreover, 6DoF exploration also helps in **diversifying** the data collection strategy, which provides the **possibility to improve upon** in the later distilling stage. For example in the catapult domain, LLM-as-Policy (2D) is only able to solve one of three possible goals (the closest bin) using a deterministic strategy. However, it provides no useful data for learning the other two goals, making it a poor data-collection policy. In contrast, incorporating 6 DOF robot utilities achieves lower but non-zero average success rates in all bins (16.3%, 3.3%, and 2.2%, full table in appendix), which provide much better exploration data for distillation.

Subtask	Planning	Verify	Execution
Open mailbox	100	100	43.5
Put package in mailbox	100	100	28.5
Raise mailbox flag	100	100	62.0
Close mailbox	100	100	94.2

Table 3: **Sources & Propagation of Error**. Accuracy (%) of planning, verification, and execution success rate (%) for each mailbox subtask.

Verify & Retry always helps. In the verify & retry step, the LLM retries all tasks until they are successful. This simple addition improves performance in all domains, with $2\times$, $3\times$, $8\times$, and $13\times$ in transport, catapult, balance, and drawer domains. Without this crucial step, we observe 0.0% success rate in the mailbox domain, underscoring the difficulty of flawlessly executing long sequences of 6 DOF actions, and the importance of recovery after failure.

Model	Size	Planning	Success
LLAMA2	7B	42.0	10.0
	13B	62.0	48.3
GPT3	175B	**82.0**	**91.1**

Table 4: **LLM Evaluation**.

Language Model Scaling. In addition to the final task success, we provide more detailed analysis of planning and success condition inference accuracy in Tab. 4. We evaluate on the proprietary GPT3 [66] (175B text-davinci-003) and the open LLAMA2 [67] (7B and 13B). We found that Llama models struggles in complex planning domains because they do not follow instructions provided in the prompts. For instance, in the drawer domain, both models fail to account for drawer opening and closing. However, we observe an upwards trend with respect to Llama model size, with the 13B model outperforming the 7B model by $+20.0\%$ and $+38.3\%$ in planning and success verification accuracy respectively.

4.2 Distilled Policy Evaluation

Robustness In, Robustness Out. By filtering trajectories with LLM's inferred success condition, distilled policies inherit the robustness of their data collection policies while improving upon success rates ($+23.4\%$ and $+33.2\%$ for no-retry and ours, Tab. 2). Since our distilled policy learned from a robust data collection policy, it also recovers from failures (*e.g.* failed grasps or placements) and continuously retries a task until it succeeds. Meanwhile, since the no-retry distilled policy learned from a data collection policy which did not retry upon failure, it is sensitive and brittle, leading to -34.8% lower average success rate across all domains compared to ours (Tab. 2).

High Performance From Diverse Retry Attempts. Plotting how long policies take to solve the balance task (Fig. 6), we observed that our policy and its data collection policy continuously tries a diverse set of grasps and placements after each failed attempt until it succeeds. This results in higher success rates as the policy is given more time, and is reflected in their monotonically increasing success rates. In contrast, baselines plateau after their first grasp/placement attempts. This highlights the synergy of two design decisions. First, the verify & retry step (§ 3.3) is crucial for demonstrating retrying behavior, but is by itself *insufficient* if each retrying action is the identical as the previous one. Instead, opting for a diffusion policy (§ 3.4) for learning from and generating high-entropy, diverse retry attempts (Fig 5) is also essential for high performance.

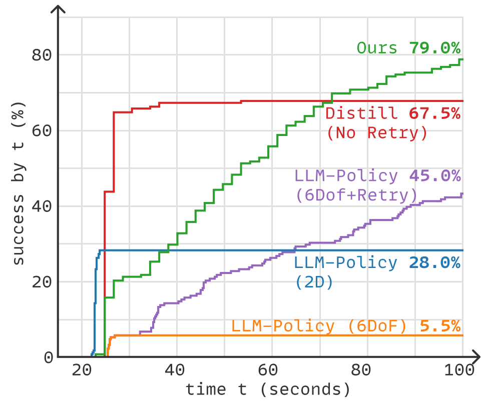

Figure 6: **Distilled Robustness**. **Our policy** inherits robust recovery from failure behavior from **its data collection policy**, while improving upon success rate.

Policy Learning Baselines. We investigate policy learning design decisions on the single-task balance domain, and remove language conditioning. While BC-Z found spatial softmax hurt their performance and opted for a mean pool, we observed using spatial softmax improved performance by +5.0%. Further, we found that switching from delta to absolute action spaces improved success rates $+6.5\%$ and $+9.5\%$ when using the MLP action decoder and our diffusion action decoder, respectively, confirming Chi et al. [12]'s findings. Lastly, we find that using our pseudo-random diffusion-based action encoder consistently outperforms a deterministic MLP action mappings, regardless of other design decisions.

Sim2Real Transfer. We evaluated a policy trained on domain randomized synthetic data in a real world transport task with five novel objects (Fig. 5e). Averaging across ten episodes per object, our policy achieved 76% success rate, demonstrating the effectiveness of our approach in Sim2Real transfer.

Method	Output				Input		Success
	Generation	Rep.	Exec	Pred	Pool	Proprio	(%)
BC-Z	FeedForward	Delta	1	10	Avg	✗	0.0
	FeedForward	Delta	4	10	Avg	✗	15.0
	FeedForward	Delta	8	10	Avg	✗	18.5
Ours	FeedForward	Delta	8	16	Spatial	✓	29.0
	FeedForward	Abs	8	16	Spatial	✓	35.5
	Diffusion	Delta	8	16	Spatial	✓	69.5
	Diffusion	Abs	8	16	Avg	✓	76.5
	Diffusion	Abs	8	16	Spatial	✓	**79.0**

Table 5: **Policy Learning Ablations**. Action generation using diffusion models [50] robustly outperforms feed-forward models across other policy design decisions.

4.3 Limitations

By using priviledged simulation state information, the LLM can infer success conditions which uses ground truth contact, joint information, and object poses. This means our implementation of the data generation phase is limited to simulation environments, and our policy requires sim2real transfer. Further, Our data generation method relies on existing 3D assets and environments, which presents a further opportunity for scaling up with assets from 3D generative models or procedural generation. Finally, while our approach's dataset contains text labels and success labels for all subtasks, we have only evaluated its effectiveness in learning the root task. Learning from all subtasks and growing a robot's set of learned, reusable sub-skills over time to enable compositional generalization is left for future work.

5 Conclusion

We proposed "Scaling Up and Distilling Down", a framework that combines the strengths of LLMs, sampling-based planners, and policy learning into a single system that automatically generates, labels, and distills diverse robot-complete exploration experience into a multi-task visuo-linguo-motor policy. The distilled policy inherits long-horizon behaviour, rich low-level manipulation skills, and robustness from its data collection policy while improving upon performance beyond its training distribution. We believe that this integrated approach is a step towards putting robotics on the same scaling trend as that of LLM development while not compromising on the rich low-level control.

References

[1] S. Song, A. Zeng, J. Lee, and T. Funkhouser. Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters*, 5(3):4978–4985, 2020.

[2] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[3] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi:10.1109/IROS.2012.6386109.

[4] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[5] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

[6] K. Grauman, A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012, 2022.

[7] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[8] J. Wu, X. Sun, A. Zeng, S. Song, J. Lee, S. Rusinkiewicz, and T. Funkhouser. Spatial action maps for mobile manipulation. *arXiv preprint arXiv:2004.09141*, 2020.

[9] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[10] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

[11] P. Florence, C. Lynch, A. Zeng, O. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. *Conference on Robot Learning (CoRL)*, November 2021.

[12] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[13] C. Lynch and P. Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

[14] S. Stepputtis, J. Campbell, M. Phielipp, S. Lee, C. Baral, and H. Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *Advances in Neural Information Processing Systems*, 33:13139–13150, 2020.

[15] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 991–1002. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/jang22a.html.

[16] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence. Interactive language: Talking to robots in real time. *arXiv preprint arXiv:2210.06407*, 2022.

[17] O. Mees, L. Hermann, and W. Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters*, 7(4):11205–11212, 2022.

[18] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[19] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3): 7327–7334, 2022.

[20] T. Xiao, H. Chan, P. Sermanet, A. Wahid, A. Brohan, K. Hausman, S. Levine, and J. Tompson. Robotic skill acquisition via instruction augmentation with vision-language models. *arXiv preprint arXiv:2211.11736*, 2022.

[21] J. Zhang, K. Pertsch, J. Zhang, and J. J. Lim. Sprint: Scalable policy pre-training via language instruction relabeling. *arXiv preprint arXiv:2306.11886*, 2023.

[22] S. Nair, E. Mitchell, K. Chen, S. Savarese, C. Finn, et al. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR, 2022.

[23] R. Goyal, S. Ebrahimi Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, et al. The" something something" video database for learning and evaluating visual common sense. In *Proceedings of the IEEE international conference on computer vision*, pages 5842–5850, 2017.

[24] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, et al. Scaling egocentric vision: The epic-kitchens dataset. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 720–736, 2018.

[25] A. S. Chen, S. Nair, and C. Finn. Learning generalizable robotic reward functions from" in-the-wild" human videos. *arXiv preprint arXiv:2103.16817*, 2021.

[26] R. Wang, J. Lehman, J. Clune, and K. O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

[27] M. Jiang, M. Dennis, J. Parker-Holder, J. Foerster, E. Grefenstette, and T. Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021.

[28] J.-B. Mouret and J. Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[29] K. Fang, T. Migimatsu, A. Mandlekar, L. Fei-Fei, and J. Bohg. Active task randomization: Learning visuomotor skills for sequential manipulation by proposing feasible and novel tasks. *arXiv preprint arXiv:2211.06134*, 2022.

[30] Y. Du, O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas. Guiding pretraining in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*, 2023.

[31] S. Mirchandani, S. Karamcheti, and D. Sadigh. Ella: Exploration through learned language abstraction. *Advances in Neural Information Processing Systems*, 34:29529–29540, 2021.

[32] R. Mendonca, S. Bahl, and D. Pathak. Alan: Autonomously exploring robotic agents in the real world. *arXiv preprint arXiv:2302.06604*, 2023.

[33] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[34] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[35] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[36] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *6th Annual Conference on Robot Learning*.

[37] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *arXiv preprint arXiv:2209.07753*, 2022.

[38] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. Palm-e: An embodied multimodal language model. *ICML*, 2023.

[39] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

[40] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[41] A. Agarwal, A. Kumar, J. Malik, and D. Pathak. Legged locomotion in challenging terrains using egocentric vision, 2022.

[42] D. Seita, A. Ganapathi, R. Hoque, M. Hwang, E. Cen, A. K. Tanwani, A. Balakrishna, B. Thananjeyan, J. Ichnowski, N. Jamali, K. Yamane, S. Iba, J. F. Canny, and K. Goldberg. Deep imitation learning of sequential fabric smoothing policies. *CoRR*, abs/1910.04854, 2019. URL http://arxiv.org/abs/1910.04854.

[43] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[44] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[45] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, and J. J. Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. *Advances in neural information processing systems*, 30, 2017.

[46] N. M. M. Shafiullah, Z. J. Cui, A. Altanzaya, and L. Pinto. Behavior transformers: Cloning k modes with one stone, 2022.

[47] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[48] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

[49] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[50] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[51] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

[52] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[53] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[54] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[55] S. M. LaValle et al. Rapidly-exploring random trees: A new tool for path planning.

[56] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[57] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[58] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*.

[59] L. Downs, A. Francis, N. Koenig, B. Kinman, R. Hickman, K. Reymann, T. B. McHugh, and V. Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items, 2022. URL https://arxiv.org/abs/2204.11918.

[60] K. Zakka. Scanned Objects MuJoCo Models, 7 2022. URL https://github.com/kevinzakka/mujoco_scanned_objects.

[61] A. Kamath, M. Singh, Y. LeCun, G. Synnaeve, I. Misra, and N. Carion. Mdetr-modulated detection for end-to-end multi-modal understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1780–1790, 2021.

[62] X. Gu, T.-Y. Lin, W. Kuo, and Y. Cui. Open-vocabulary object detection via vision and language knowledge distillation. *arXiv preprint arXiv:2104.13921*, 2021.

[63] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[64] P. Florence, L. Manuelli, and R. Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 5(2):492–499, 2019.

[65] A. Mandlekar, F. Ramos, B. Boots, S. Savarese, L. Fei-Fei, A. Garg, and D. Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4414–4420. IEEE, 2020.

[66] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[67] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

A Policy Rollout Visualizations

Our policy's 6DoF manipulation behavior is best visualized through videos. Please visit this anonymized website to view the videos.

B LLM Prompts

Below, we include all prompts used in our approach. We use the same LLM pipeline and prompts in all domains and tasks. We first outline the rationale behind our design of the LLM pipeline (§ B.1). Next, we describe in detail the LLM modules and how they are used in the data generation stage (§ B.2), summarize the general prompt structure (§ B.3), and outline the API supplied to the LLM for success condition inference (§ B.4). Finally, we show some examples of LLM completions (§ B.5).

In all of our experiments, we use GPT3 (text-davinci-003) with temperature 0.0.

B.1 LLM Pipeline Design

Our LLM pipeline is factorized into multiple LLM modules, allowing each module's prompt to speciallize in a small reasoning skill (*e.g.* one set of prompts for deciding whether a task involves a single or multiple objects). We found that this not only improves the LLM's performance, but also makes designing and maintaining prompts easy. For instance, during development, if the LLM outputs an unexpected task tree, the error could be traced back to a single module, and only that module's prompt needs to be updated. Another convenient feature of this approach is that it also saves on token usage. Since each module's task is small (*e.g.* answer only "one" or "multiple"), the amount of completion tokens is significantly smaller than a monolithic prompt. Further, when a module's prompt is updated, only that module's outputs needs to be updated, allowing cost-effective approaches to cache-ing LLM's completions.

B.2 LLM Pipeline

The recursive LLM-based planner starts with an ambiguous task description handler (Listing 1), which transforms ambiguous task descriptions such as "move the block onto the catapult then shoot the block into the furthest bin" into more specific task descriptions like "move the block onto the catapult then shoot the block into the furthest bin by pressing the catapult's button". While this handler's task can occasionally overlap with the LLM planner's task, we found that it was more effective to keep them separate.

Next, given a un-ambiguous task description, the LLM planner first decides whether the planning step is necessary by checking whether the task involves touching only a single object or requiring further decomposition (Listing 2). If the task involves multiple objects, it proceeds with planning (explained in the next paragraph). If the task involves only one object part, an LLM identifies which object part name it should interact with (Listing 3). If the object part name is a single-link rigid object, the LLM is asked for which object it should move (the pick object part) and where (the place object part) using the prompt in Listing 4.

In the planning step, the LLM planner outputs a list of subtasks (Listing 5). Given the recursive nature of this planning module, parent tasks also need to keep track of and propagate the current state of the environment to child tasks. For instance, the "open the fridge" subtask should be followed with "with the fridge door opened, move the eggs from the fridge ...", such that the recursive call for moving the eggs knows it does not need to open the fridge door again.

After it has inferred the full task tree, the LLM also infers a success condition for every task in the task tree (Listing 6) in the form of a code-snippet. Similar to [37], we inform the LLM which state API utilities are available for its usage by including import statements at the top of the file and demonstrating how they are used in the examples.

B.3 Prompt Structure

All prompts start with instructions to explain to the LLM what the task is (*e.g.* "given an input task description, the goal is to output a list of subtasks .."), followed by a few "shots" of examples, separated by a "#" symbol (in text-based prompts) or a multi-line comment (in code-based prompts). Each shot starts with a structured text encoding of the scene's object's and their parts' names in the form of a bullet list. In the planning, success condition inference, single-or-multiple , pick-and-place, and ambiguous task description LLM tasks, we found that it was helpful to encourage the LLM to output its reasoning (either with an explicit "reasoning:" field or through in-line code comments). In contrast, we found the object part identifier task to be more effective without this explicit reasoning field.

B.4 APIs for Success Condition Code Generation

All functions take as the first argument the simulation state, which contains information on object and part names, kinematic structure, contact, all degrees of freedom, and collision meshes.

Contact. This function takes as input two object (part) names, and returns whether they (or any of their parts) are in contact.

Activation. A pair of functions, `check_activated` and `check_deactivated`, take as input an object part name and checks whether the revolute/prismatic joint connecting the object part to its parent link are near their maximum or minimum values, respectively. This is useful for checking whether a lid is opened/closed or a button is pressed/released.

Spatial Relations. We provide two spatial relations, `check_on_top_of` and `check_inside`, which takes two object (part) names and returns whether the first object (part) is on top of the second object (part) or inside the second object (part), respectively. An object is on top of another if they are in contact and the contact normal's dot product with the up direction is greater than 0.99. An object is inside a container if that the intersection of that object's axis-aligned bounding boxes with the container's axis-aligned bounding boxes is at least 75% of the object's axis-aligned bounding box's volume. This axis-aligned bounding box information can be parsed from the collision checker of most physics simulators.

Listing 1: Ambiguous task description handler's prompts

```
instructions:
given an input task description, the goal is rephrase the task such that it is not ambiguous.
if the task is already specific enough, just return the original task description.
below are some examples:
#
task: stack the blocks on top of each other
scene:
 - navy block
 - maroon block
 - violet block
reasoning: the block stacking order is ambiguous. we can specify which block should be placed
     on which, in which order.
answer: move the maroon block onto the navy block, and the violet block on the maroon block.
#
task: move the lilac block onto the brown block
scene:
 - brown block
 - lilac block
 - yellow block
reasoning: the blocks to interact with are fully specified, so just return the original task
     description.
answer: move the lilac block onto the brown block.
#
task: sort the blocks based on their color's temperature onto corresponding plates
scene:
 - red block
 - orange block
 - blue block
 - purple block
 - red plate
 - blue plate
reasoning: which blocks and plates belong to the same color temperature group are ambiguous.
     we can specify exactly which blocks should be placed on which plate.
answer: move the red and orange blocks onto the red plate, and the purple and blue blocks onto
      the blue plate.
#
task: open the jar
scene:
 - jar
    + jar lid
reasoning: opening a jar is a primitive action and is fully specified, so just return the
     original task description.
answer: open the jar.
#
task: close the second drawer
scene:
 - drawer
    + first drawer
       + first drawer handle
    + second drawer
       + second drawer handle
    + third drawer
       + third drawer handle
reasoning: closing the second drawer is a primitive action towards a specific drawer, so just
     return the original task description.
```

```
answer: close the second drawer.
#
task: move the ingredients for the omelette onto the kitchen counter
scene:
 - kitchen counter
    + cupboard
       + cupboard door
          + cupboard door handle
       + salt
       + pepper
 - fridge
    + fridge door
       + fridge door handle
    + fridge top shelf
       + eggs
       + butter
       + cheese
       + milk
    + fridge bottom shelf
       + mushrooms
       + broccoli
    + freezer
       + lamb shank
       + trader joe's dumplings
       + tilapia fillet
reasoning: which ingredients belong to the omelette is ambiguous. we can specify exactly which
      items to take out of the fridge.
answer: move the eggs, butter, cheese, and mushrooms onto the kitchen counter and the salt and
      pepper onto the kitchen counter.
#
task: open the fridge, move the cheese onto the kitchen counter, and then close the fridge.
scene:
 - kitchen counter
    + cupboard
       + cupboard door
          + cupboard door handle
       + salt
       + pepper
 - fridge
    + fridge door
       + fridge door handle
    + fridge top shelf
       + eggs
       + butter
       + cheese
       + milk
    + fridge bottom shelf
       + mushrooms
       + broccoli
    + freezer
       + lamb shank
       + trader joe's dumplings
       + tilapia fillet
reasoning: which actions to perform and in which order is fully specified, so just return the
     original task description.
answer: open the fridge, move the cheese onto the kitchen counter, and then close the fridge.
```

Listing 2: One-or-Multiple module's prompts

```
instructions:
given an input task description, the goal is to classify whether performing the task will
     involve touching only "one" object or "multiple" objects.
all objects start in a de-activated state (e.g., doors, drawers, cabinets, cupboards, and
     other objects with doors are closed, lights are off, etc.) unless specified otherwise (e.
     g., with the door opened).
after performing the task, objects should be reset to their de-activated state if relevant.
below are some examples:
#
task: move the blue block onto the plate
scene:
 - green block
 - blue block
 - red block
 - plate
reasoning: "moving the blue block onto the plate" involves two objects, the blue block and the
      plate. moving the blue block requires touching it. the plate does not have any
     activation state, so does not need to be touched.
answer: one.
#
task: stack the blocks on the plate
scene:
 - green block
 - plate
 - red block
```

```
 - blue block
reasoning: "stack the blocks" can be decomposed into moving the red block onto the plate,
     moving the green block onto the red block, and moving the blue block onto the green block
     . performing these steps involve touching multiple blocks.
answer: multiple.
#
task: with the red block on the plate and the orange block on the red block, move the green
     block onto the pink block
scene:
 - orange block
 - pink block
 - plate
 - green block
 - red block
reasoning: "moving the green block onto the pink block" involves two objects, the green block
     and the pink block. moving the green block requires touching it. the pink block does not
     have any activation state, so does not need to be touched.
answer: one.
#
task: move the lasagna into the microwave
scene:
 - microwave
    + microwave door
       + microwave door handle
 - kitchen counter
 - fridge
    + fridge door
       + fridge door handle
 - lasagna
reasoning: "moving the pasta into the microwave" involves only two objects, the lasagna and
     the microwave. however, it is not a primitive task because the microwave has a door (
     activation state), but it starts off being closed (de-activated). opening the microwave
     involves touching the microwave.
answer: multiple.
#
task: with the microwave opened, move the pasta into the microwave
scene:
 - microwave
    + microwave door
       + microwave door handle
 - kitchen counter
 - fridge
    + fridge door
       + fridge door handle
 - pasta
reasoning: "moving the pasta into the microwave" involves two objects, the pasta and the
     microwave. the microwave's door needs to be opened (activation state), but it is already
     opened. since the task asserts that the microwave is opened, it also does not need to be
     closed afterwards. this means performing the task does not involve touching the microwave
     .
answer: multiple.
#
task: open the microwave
scene:
 - fridge
    + fridge door
       + fridge door handle
 - dumplings
 - microwave
    + microwave door
       + microwave door handle
 - kitchen counter
reasoning: "opening the microwave" is a primitive task. it involves only one object, the
     microwave.
answer: one.
#
task: with the microwave opened and the sandwich in the microwave, close the microwave
scene:
 - fridge
    + fridge door
       + fridge door handle
 - sandwich
 - microwave
    + microwave door
       + microwave door handle
 - kitchen counter
reasoning: "closing the microwave" is a primitive task. it involves only one object, the
     microwave.
answer: one.
```

Listing 3: Object part identifier's prompts

```
instructions: given an input task description, the goal is to identify which object part from
     the scene to interact with.
```

```

below are some examples:
#
task: stack the blue block on the plate
scene:
 - red block
 - blue block
 - green block
 - plate
answer: blue block.
#
task: with the red block on the plate, stack the green block on the red block
scene:
 - red block
 - blue block
 - green block
 - plate
answer: green block.
#
task: turn on the lights
scene:
 - light switch
 - ceiling light
 - wall
answer: light switch.
#
task: open the microwave
scene:
 - microwave
    + microwave door
       + microwave door handle
    + microwave start button
    + microwave plate
 - kitchen counter
    + cupboard
       + cupboard door
          + cupboard door handle
answer: microwave door handle.
#
task: with microwave opened and the lasagna on the kitchen counter, move the lasagna into the microwave
scene:
 - kitchen counter
    + cupboard
       + cupboard door
          + cupboard door handle
 - fridge
    + fridge door
       + fridge door handle
    + fridge top shelf
    + fridge bottom shelf
    + freezer
 - lasagna
 - microwave
    + microwave door
       + microwave door handle
    + microwave start button
    + microwave plate
answer: lasagna.
#
task: with the fridge door opened, open the cupboard
scene:
 - microwave
    + microwave door
       + microwave door handle
    + microwave start button
    + microwave plate
 - kitchen counter
    + cupboard
       + cupboard door
          + cupboard door handle
 - fridge
    + fridge door
       + fridge door handle
    + fridge top shelf
    + fridge bottom shelf
    + freezer
 - lasagna
answer: cupboard door handle.
```

Listing 4: Pick & place handler's prompts

```
instructions: given an input pick and place description, the goal is to identify which object to pick and where to place among the objects listed in the scene.
```

```
below are some examples:
#
task: move the blue block on the plate
scene:
 - red block
 - blue block
 - green block
 - plate
pick: blue block.
place: plate.
#
task: with the red block on the plate, move the green block to the top of the red block
scene:
 - red block
 - blue block
 - green block
 - plate
pick: green block.
place: red block.
#
task: with microwave opened and the lasagna on the kitchen counter, move the lasagna into the
     microwave
scene:
 - kitchen counter
    + cupboard
       + cupboard door
          + cupboard door handle
 - fridge
    + fridge door
       + fridge door handle
    + fridge top shelf
    + fridge bottom shelf
    + freezer
 - lasagna
 - microwave
    + microwave door
       + microwave door handle
    + microwave start button
    + microwave plate
pick: lasagna.
place: microwave plate.
```

Listing 5: Planning module's prompts

```
instructions: given a input task description, the goal is to output a list of subtasks, which,
      when performed in sequence would solve the input task. all objects start in a de-
     activated state (e.g., doors, drawers, cabinets, cupboards, and other objects with doors
     are closed, lights are off, etc.) unless specified otherwise (e.g., with the door opened)
     . after performing the task, objects should be reset to their de-activated state if
     possible. below are some examples:
#
task: move the red block onto the plate, the blue block onto the red block, and the green
     block on the blue block
scene:
 - red block
 - blue block
 - green block
 - plate
reasoning: no objects have activation states. the blocks can be directly placed onto the
     plates.
answer:
 - 1. move the red block onto the plate
 - 2. with the red block on the plate, move the blue block onto the red block
 - 3. with the red block on the plate and the blue block on the red block, move the green
      block onto the blue block
#
task: move the eggs, salt, and pepper onto the kitchen counter
scene:
 - kitchen counter
    + cupboard
       + cupboard door
          + cupboard door handle
       + salt
       + pepper
 - fridge
    + fridge door
       + fridge door handle
    + fridge top shelf
       + eggs
       + butter
       + cheese
       + milk
    + fridge bottom shelf
```

```
    + freezer door
       + freezer door handle
reasoning: the fridge and cupboard has doors (activation states) which start off closed (de-
     activated). they need to be opened before objects can be taken out of them. after the
     task is done, they need to be closed (reset).
answer:
 - 1. open the fridge
 - 2. with the fridge door opened, move the eggs from the fridge onto the kitchen counter
 - 3. with the eggs on the kitchen counter, close the fridge
 - 4. with the eggs on the kitchen counter, open the cupboard
 - 5. with the eggs on the kitchen counter and the cupboard door opened, move the salt onto
      the kitchen counter
 - 6. with the eggs and salt on the kitchen counter and the cupboard door opened, move the
      pepper onto the kitchen counter
 - 7. with the eggs, salt, and pepper on the kitchen counter, close the cupboard door
#
task: with the fridge door opened, move the eggs, salt, and pepper onto the kitchen counter
scene:
 - kitchen counter
    + cupboard
       + cupboard door
          + cupboard door handle
       + salt
       + pepper
 - fridge
    + fridge door
       + fridge door handle
    + fridge top shelf
       + eggs
       + butter
       + cheese
       + milk
    + fridge bottom shelf
    + freezer door
       + freezer door handle
reasoning: the fridge and cupboard has doors (activation states). the fridge's door is already
      opened (activated) and so don't need to be reset. the cupboard's door starts off closed
     (de-activated) but needs to be opened before objects can be taken out of it. after the
     task is done, the cupboard need to be closed (reset).
answer:
 - 1. with the fridge door opened, move the eggs from the fridge onto the kitchen counter
 - 2. with the fridge door opened and the eggs on the kitchen counter, open the cupboard
 - 3. with the fridge door opened, the eggs on the kitchen counter, and the cupboard door
      opened, move the salt onto the kitchen counter
 - 4. with the fridge door opened, the eggs and salt on the kitchen counter, and the cupboard
      door opened, move the pepper onto the kitchen counter
 - 5. with the fridge door opened, the eggs, salt, and pepper on the kitchen counter, close
      the cupboard door
```

Listing 6: Success Condition Inference module's prompts

```python
from utils import (
    check_contact,
    check_activated,
    check_deactivated,
    check_inside,
    check_on_top_of,
    EnvState,
)


"""
instructions:
given a input task description, the goal is to output the success condition for
that task. unless otherwise specified, all objects start in a de-activated state
(e.g., doors, drawers, cabinets, cupboards, and other containers are closed,
lights are off, etc.) unless specified otherwise (e.g., with the door opened).
after performing the task, objects should be reset to original state if possible.
"""


# robot task: touch the apple
# scene:
# - apple
#    + apple body
#    + apple stem
def touching_apple(init_state: EnvState, final_state: EnvState):
    return check_contact(
        final_state, "robotiq left finger", "apple body"
    ) and check_contact(final_state, "robotiq right finger", "apple body")


# robot task: release the cup
# scene:
```

```
# - cup
#    + cup body
#    + cup handle
def released_cup(init_state: EnvState, final_state: EnvState):
    finally_touching_cup = check_contact(
        final_state, "robotiq left finger", "cup handle"
    ) and check_contact(final_state, "robotiq right finger", "cup handle")
    finally_released_cup = (not finally_touching_cup) and (
        not final_state.gripper_command
    )
    return finally_released_cup


# robot task: move the milk carton into the shelf
# scene:
# - milk carton
# - coke can
# - shelf
def milk_carton_is_on_shelf(init_state: EnvState, final_state: EnvState):
    return check_on_top_of(final_state, "milk carton", "shelf")


# robot task: move the milk carton from the shelf
# scene:
# - milk carton
# - coke can
# - shelf
def milk_carton_is_not_on_shelf(init_state: EnvState, final_state: EnvState):
    return not check_on_top_of(final_state, "milk carton", "shelf")


# robot task: open the washing machine
# scene:
# - washing machine
#    + washing machine door
#       + washing machine door handle
#    + control panel
#       + on off button
def washing_machine_opened(init_state: EnvState, final_state: EnvState):
    return check_activated(final_state, "washing machine door")


# robot task: move the sock into the washing machine
# scene:
# - washing machine
#    + washing machine door
#       + washing machine door handle
#    + control panel
#       + on off button
# - sock
def sock_inside_washing_machine(init_state: EnvState, final_state: EnvState):
    # the washing machine can be opened (activated state) or closed (de-activated
    # state). since its activation state was not specified, the washing machine starts
    # off closed. therefore, it needs to be closed after the sock is moved inside.
    sock_inside_washing_machine = check_inside(final_state, "sock", "washing machine")
    washing_machine_door_closed = check_deactivated(final_state, "washing machine door")
    return sock_inside_washing_machine and washing_machine_door_closed


# robot task: with the washing machine opened, move the sock into the washing machine
# scene:
# - washing machine
#    + washing machine door
#       + washing machine door handle
#    + control panel
#       + on off button
# - sock
def sock_inside_washing_machine_with_washing_machine_opened(
    init_state: EnvState, final_state: EnvState
):
    # the washing machine can be opened (activated state) or closed (de-activated
    # state). the washing machine starts off opened, so it does not need to be closed
    # after the sock is moved inside.
    sock_inside_washing_machine = check_inside(final_state, "sock", "washing machine")
    return sock_inside_washing_machine
```

B.5 Example Completions

Most modules are designed such that their answers are multi-choice (*e.g.* single or multiple). We show examples of more interesting open-ended completions for the ambiguous task description handler, planning, and success condition inference modules. The LLM's outputs are highlighted in blue.

B.5.1 Ambiguous Task Description Handler

```
task:  move the block onto the catapult arm, then press the button to shoot the block into the
closest box
scene:
- catapult
    + button
    + catapult arm
- yellow block
- closest box
- furthest box
- middle box
- table
reasoning:  which block to move onto the catapult arm is ambiguous.  we can specify exactly
which block to move onto the catapult arm.
answer:  move the yellow block onto the catapult arm, then press the button to shoot the block
into the closest box.
```

```
task:  send the amazon package for return
scene:
- mailbox
   + mailbox lid
      + mailbox lid handle
   + mailbox flag
- amazon package
- table
reasoning:  which actions to perform and in which order is ambiguous.  we can specify exactly
 which actions to take.
answer:  open the mailbox lid, place the amazon package inside the mailbox, raise the mailbox
flag, and then close the mailbox lid.
```

B.5.2 Planning

```
task:  move the yellow block onto the catapult arm, then press the button to shoot the block
into the closest box
scene:
- catapult
   + button
   + catapult arm
- yellow block
- closest box
- furthest box
- middle box
- table
reasoning:  the catapult has a button (activation state) which starts off de-activated.  it
needs to be pressed to shoot the block.  after the task is done, the button should be reset to
its de-activated state.
answer:
 - 1.  move the yellow block onto the catapult arm
 - 2.  with the yellow block on the catapult arm, press the button to shoot the block into the
 closest box
 - 3.  with the yellow block in the closest box, reset the button to its de-activated state
```

```
task:  open the mailbox lid, place the amazon package inside the mailbox, raise the mailbox
flag, and then close the mailbox lid
scene:
- mailbox
   + mailbox lid
      + mailbox lid handle
   + mailbox flag
- amazon package
- table
reasoning:  the mailbox lid has an activation state (closed/de-activated).  it needs to be
opened before the package can be placed inside.  after the task is done, the lid needs to be
closed (reset).
answer:
- 1.  open the mailbox lid
- 2.  with the mailbox lid opened, move the amazon package from the table into the mailbox
- 3.  with the amazon package in the mailbox, raise the mailbox flag
- 4.  with the amazon package in the mailbox and the mailbox flag raised, close the mailbox lid
```

B.5.3 Success Condition Inference

```
# robot task: move the yellow block onto the catapult arm
# scene:
# - catapult
#    + button
#    + catapult arm
# - yellow block
# - closest box
# - furthest box
# - middle box
# - table
def yellow_block_on_catapult_arm(init_state: EnvState, final_state: EnvState):
return check_on_top_of(final_state, "yellow block", "catapult arm")
```

```
# robot task: open the mailbox lid
# scene:
# - mailbox
#    + mailbox lid
#       + mailbox lid handle
#    + mailbox flag
# - amazon package
# - table
def mailbox_lid_opened(init_state: EnvState, final_state: EnvState):
return check_activated(final_state, "mailbox lid")
```

C Training & Data Details.

C.1 Data Generation

Our data-collection policy uses the 6DoF Exploration Primitives with the Verify & Retry step. For each domain, we run data generation until we get at least 500 successful trajectories per task. Although this can be costly when tasks are long horizon with low success rates (the mailbox domain took 2 days on 256 CPU cores Intel Xeon Gold 6230R CPU @ 2.10GHz), data generation happens only once.

C.2 Network Architecture & Hyperparameters

We use the same network architecture and hyperparameters for all domains. Our task descriptions are encoded using CLIP B/32's text encoder [56], and projected into a 512-dimensional vector. For each of the two camera view, we learn a separate Resnet18-based [4] vision encoder, whose features are flattened, concatenated, and projected into a 512-dimensional vector. The Resnet18 architecture is pre-processed by replacing BatchNorm with GroupNorm and replacing the final average pool layer with a spatial softmax pooling [4, 12]. We use an image resolution of 160×240 for each view, processed with a 90% random crop to 144×216. Finally, the proprioception is concatenated with the vision and text encoder as the condition into the diffusion policy. We use the convolution network-based diffusion policy architecture [12]. The final network has 108 million parameters. All networks are optimized end-to-end with the AdamW optimizer, with 5e-5 learning rate and 1e-6 weight decay, and a cosine learning rate scheduler. For evaluation, we use an exponential moving average of all networks with a decay rate of 0.75.

C.3 Training

We train a separate multi-task policy for each domain using the same hyperparameters and network architecture. For domains with only a single task, this amounts to a single-task policy. All networks are trained for 2 days on a single NVIDIA A6000, and the best checkpoint's performance is reported. We found that performance typically saturates around 1 day into training.

D Utilities Implementation

For motion planning, we implemented rapidly-exploring random trees (RRT [55]) with grasped-object-aware collision checking, allowing the robot to motion plan with dynamic grasping constraints. The geometry-based grasp and placement sampler is implemented using point clouds created from depth maps, camera matrices, and segmentation maps from the simulator. While our grasp sampler uses only geometry, kinematics, and contact information, including other grasp quality metrics (*e.g.* stability analysis) can improve its performance. In the placement sampler, we sample candidate place positions at points whose estimated contact normal is aligned against the gravity direction. The revolute and prismatic joint motion primitives are implemented by

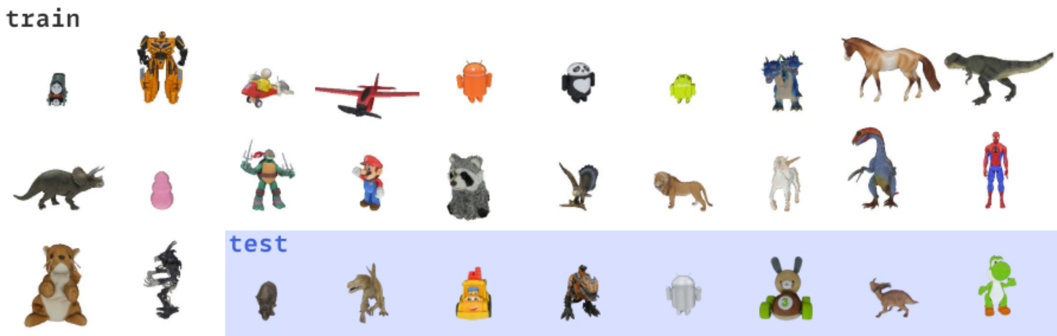

Figure 7: **Generalization to Novel Objects.** The Transport domain requires generalization to diverse and novel object shapes and colors. Trained to transport 22 toys, our distilled policy generalizes to 8 novel toys (in blue section). All objects rendered from a fixed camera to show diversity of object size.

checking the grasp pose relative to the joint (*e.g.* mailbox lid handle grasp relative to the mailbox lid hinge), then performing a circular motion around the joint axis or a linear motion along the joint axis, respectively.

E Benchmark

Our benchmark is built on top of the Mujoco [3] simulator, using assets from the Google Scanned Objects dataset [59, 60]. We use a table-top manipulation set-up, with a WSG50 gripper and Toyota Research Institute Finray fingers mounted on a UR5e, with a policy control rate of 4Hz. The workspace has two cameras, one front view, which observes the entire workspace and robot, and a wrist-mounted camera, which is used to help with fine-grained manipulation [4]. We end episodes when any object is dropped to the floor. Below, we clarify how we design the tasks for each domain.

E.1 Mailbox

To be considered successful, the mailbox needs to be closed with the package inside the mailbox, with the mailbox flag raised within 200 seconds (800 control cycles). During data generation and testing, the package's planar position is uniformly random in a planar bound of dimensions [10cm, 10cm]. At evaluation, the policy has to generalize to unseen package positions. The amazon has is a rigid object with 6DoF. The mailbox is a fixed rigid object, with one degree of freedom for each of its revolute joints, one for the mailbox lid, and one for the mailbox flag.

E.2 Transport

To be considered successful, the toy needs to be inside the left bin within 100 seconds. At the beginning of each episode, a random toy 3D asset is sampled. During data generation and testing, the toy's position is uniformly random inside the right bin, and orientation uniformly random along all three euler axes. On top of novel randomized poses, the policy also has to generalize to unseen object instances with novel geometry. We use 22 toys for data generation, and 8 for testing (Fig. 7). The toy is a rigid object with 6DoF, while the bins are fixed rigid objects with no DoF. The bin asset names corresponds with their spatial location (*e.g.* the left bin is called "left bin" when the scene is presented to the LLM).

E.3 Drawer

This is a multi-task domain with 12 tasks, where each task involves moving one of the four objects (vitamin bottle, pencil case, crayon box, horse) into one of the three drawers (top, middle, bottom). The task description follows the template "move the $\langle$*object*$\rangle$ into the $\langle$ *drawer*$\rangle$". To be considered successful, the specified object needs to be inside the specified drawer within 120 seconds. During data generation and testing, each of the four object's position is uniformly random within a planar bound of dimensions [10cm,10cm], centered around 4 evenly spaced locations along the table. At test time, the policy has to generalize to unseen object positions in the same distribution as its data generation.

All four objects are rigid objects with 6DoF. The drawer is a fixed articulated object with 3 DoF, one for each of the drawers.

Approach	Crayon			Horse			Pencilcase			Vitamin			Avg.
	B	M	T	B	M	T	B	M	T	B	M	T	
LLM-as-Policy (2D) 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(+) 6DoF Robot Utils	5.5	0.5	0.0	2.0	0.0	0.0	5.0	0.0	0.0	2.0	0.0	0.0	1.3
(+) Verify & Retry	48.5	39.5	33.0	45.5	32.0	24.5	46.0	27.0	20.0	27.0	18.5	20.5	31.8
Distill No Retry	19.0	19.0	17.5	13.0	34.0	22.5	27.5	41.0	39.5	13.5	12.5	13.5	22.7
Distill (Ours)	**57.5**	**63.0**	**50.0**	**62.5**	**59.0**	**51.5**	**59.5**	**72.5**	**61.5**	**46.0**	**39.5**	**46.5**	**55.8**

Table 6: **Drawer Quantitative Results (Success Rate %)** where B, M, T means bottom, middle, and top drawers. Averaged over 200 episodes.

Approach	Balance	Catapult			Transport		Mailbox
		Near	Mid	Far	Train	Test	
LLM-as-Policy (2D)	28.0	100.0	0.0	0.0	–	21.5	0.0
(+) 6DoF Robot Utils	5.5	7.0	1.0	0.0	–	35.0	0.0
(+) Verify & Retry	45.0	16.3	3.3	2.2	–	**82.0**	3.0
Distill No Retry	67.5	2.5	**56.5**	**56.5**	31.0	32.5	0.0
Distill (Ours)	**79.0**	**78.0**	52.0	45.0	**74.0**	80.0	**62.0**

Table 7: **Full Quantitative Results (Success Rate %).** Averaged over 200 episodes.

E.4 Catapult

This is a multi-task domain with 3 tasks, one for each of the three bins. The task description follows the template "move the block onto the catapult arm, then press the button to shoot the block into the ⟨*bin*⟩" where ⟨*bin*⟩ is either closest, middle, or furthest bin. The bin asset names corresponds with their spatial location (*e.g.* the furthest bin is called "furthest bin" when the scene is presented to the LLM).

In order to be considered successful, the block needs to be inside the specified bin within 60 seconds. This is a short amount of time, which prevents policies from retrying after failure. The block is a rigid object with 6DoF. The bins are fixed rigid objects with no degrees of freedom. The catapult has two degrees of freedom, one revolute joint for the catapult arm, and one prismatic joint for the button. This task is designed to study tool-use, and does not have any pose randomization. Thus, different seeds affect only the policy's pseudo random samplers or the diffusion process.

We implement the catapult with a special callback function which checks whether the button sliding joint is near its max value. If it is, then the constraint that holds the catapult arm down is disabled, releasing the spring loaded catapult arm hinge joint.

E.5 Bus Balance

In order to be considered successful, the bus needs to be fully balanced on top of the block within 100 seconds. On top of testing for intuitive physics, this high precision requirement of this task was also used to test the policy's precision and ability to recover from failure, which is why we allow a generous time budget. The task description is "balance the bus on the block".

The bus is a rigid object with 6DoF, dropped from a fixed location above the table with uniformly random orientation. This means when the bus drops, it lands in different positions and orientations. The block is fixed with no degrees of freedom.

F Full Results

We include the full results for all tasks in the drawer domain in Table 6, and all other domains in Table 7. We omit data generation baseline numbers on the train set in the transport domain, since they are non-learning approaches. All approaches are evaluated on 200 different seeds, which controls pose randomization, which asset is sampled, the pseudo-random robotic utility samplers, and the pseudo-random diffusion process. We make one exception in the catapult domain, where due to the low success rates of getting the block into the middle and far bin, we run evaluation until there are 500 successful trajectories per task, then report the average success rate. Since the time limit for the catapult is short, the data-collection policy will not have enough time to retry, leading to identical numbers with the baseline data-collection policy without verify & retry.

In the drawer domain, we observe that the task is more difficult for:

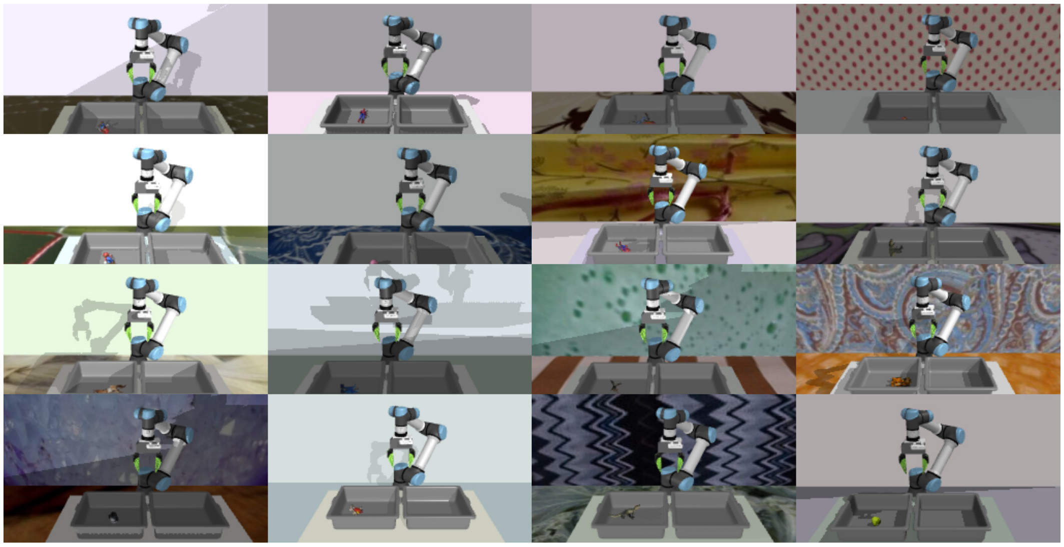

Figure 8: **Domain Randomization.** To facilitate Sim2Real transfer, we train our policy on lighting, texture, and camera pose randomized scenes.

1. **Larger objects:** The most challenging objects are the vitamin bottle and the horse toy, both of which are too large to fit the drawer if they are in an upright orientation. This means to be effective at this task, the robot should perform sideway grasps on these objects, such that downstream placement is easier. In contrast, the small crayon box is has the highest success rates amongst the data-collection policies.
2. **Top drawer:** We observe interacting with this drawer often brings the robot close to its kinematic reach range. This means slight imprecision in the policy's predicted actions or small shifts in the grasped object (which is unaccounted for during motion planning) in execution could lead to failure. For instance, while moving the objects inside the top drawer, the grasped object could collide with the drawer, causing the grasped object to drop or the drawer to close.
3. **Planar Action Primitives:** A top-down grasp on the drawer handle will typically be in collision with the drawer's body. Thus, in LLM-as-Policy (2D)'s first action to open the drawer, its call to the motion planner will fail due to an invalid goal configuration.

G Real World Evaluation

Figure 9: **Real World Objects**.

We train a separate policy for real-world transfer on domain randomized scenes (Fig. 8). We evaluate our policy on a real UR5e robot with a WSG50 gripper and Toyota Research Institute Finray fingers, matching our simulation set-up. We use five unseen objects (Fig. 9), ranging in shape, size, and visual appearance. Each object is evaluated on 10 episodes, with the object placed at a random pose on the right bin. We observe 70%, 80%, 60%, 80%, and 90% for the pear, monster, rubiks cube, fetch controller, and mustard bottle respectively, giving a mean success rate of 76%.