

# LLM Trainer: Automated Robotic Data Generating via Demonstration Augmentation using LLMs

Abraham George<sup>1</sup> and Amir Barati Farimani<sup>1</sup>

**Abstract**—We present *LLM Trainer*, a fully automated pipeline that leverages the world knowledge of Large Language Models (LLMs) to transform a small number of human demonstrations (as few as one) into a large robot dataset for imitation learning. Our approach decomposes demonstration generation into two steps: (1) offline demonstration annotation that extracts keyframes, salient objects, and pose-object relations; and (2) online keypose retargeting that adapts those keyframes to a new scene, given an initial observation. Using these modified keypoints, our system warps the original demonstration to generate a new trajectory, which is then executed, and the resulting demo, if successful, is saved. Because the annotation is reusable across scenes, we use Thompson sampling to optimize the annotation, significantly improving generation success rate. We evaluate our method on a range of tasks, and find that our data annotation method consistently outperforms expert-engineered baselines. We further show an ensemble policy that combines the optimized LLM feed-forward plan with a learned feedback imitation learning controller. Finally, we demonstrate hardware feasibility on a Franka Emika Panda robot. For additional materials and demonstration videos, please see the project website: <https://sites.google.com/andrew.cmu.edu/llm-trainer>

## I. INTRODUCTION

Recent advances in Large Language Models (LLMs) have revolutionized the field of robot learning, with applications ranging from task planning [1], to tool use in long horizon tasks [2], to deformable object manipulation [3]. At the core of these works is the LLM’s broad base of world knowledge, gathered from training on internet-scale data, which allows these agents to be extremely generalizable. In this work, we seek to leverage the world knowledge of LLMs to fully automate demonstration generation through human demo augmentation. To do this, we employ a similar pipeline as [4] and [5] for data generation: first, identify key robot poses in a demonstration, then generate a new environment and modify the key poses based on an initial observation, and finally, use the new key poses to warp the demonstration trajectory, resulting in a new trajectory, which is rolled out in the new environment. However, unlike prior works which rely on human annotation and hard-coded methods to identify key poses and modify them in response to the new environments [4]–[6], our system seeks to fully automate this process by leveraging large language models (LLMs). An outline of our method can be seen in Fig. 1.

Our method for LLM-based data generation has two main steps: First, the LLM annotates the human demonstration, identifying keyframes (timesteps that are important inflection points for the task), listing relevant objects at each keyframe,

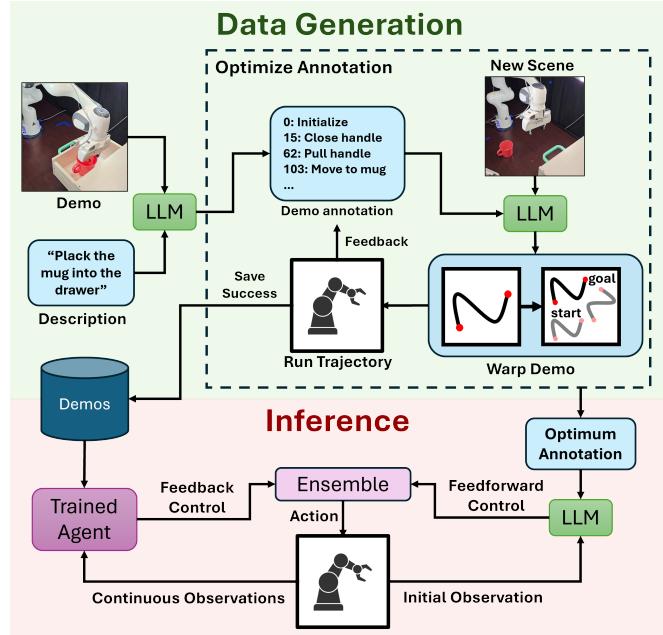


Fig. 1: Diagram of our approach. First, the LLM annotates the demonstration, noting key points from the task and any relevant objects. A new instance of the task environment is initialized, and the LLM uses the initial observation and the annotated demo to create a new set of keypoints. These keypoints are used to warp the demonstration trajectory, and the warped trajectory is rolled out in the new environment. If successful, the generated data is saved for later use in training an IL agent. The binary feedback from the rollout (success or failure) is used to optimize the demo annotation.

and explaining the relationship between the robot and these objects. Second, the LLM uses this annotation, along with an initial observation of a newly initialized scene, to determine how the robot’s pose should be adjusted at each keypoint. Because the first step of this process does not require information from the new scene, we can reuse these annotations, saving compute cost and opening the door for optimization. By employing a multi-armed bandit-based method, we are able to optimize the demo annotation step, improving data generation success rate by 2-3 times.

Once the data generation process is complete, we can use the generated data to train imitation learning agents. Additionally, thanks to our annotation optimization process, we develop a highly effective LLM-based feed-forward policy during data generation. In addition to serving as a viable agent on its own, we show that this feed-forward

<sup>1</sup>With the Department of Mechanical Engineering, Carnegie Mellon University aigeorge@andrew.cmu.edu, barati@cmu.edu

policy, when combined with the feedback agent, can form an effective ensembled policy, combining the long-horizon planning and generalizability of LLMs with the feedback control of imitation learning.

This work has three main contributions:

- An LLM-based data generation method which can autonomously generate data using only a single, unannotated demonstration and a short (one sentence) description of the task.
- A multi-armed bandit-based optimization method which significantly improves demo generation success rate, allowing our method to outperform baselines that rely on expert annotations.
- An ensembling strategy to combine a learned IL policy with the optimized LLM-based feedforward controller developed during data collection.

## II. RELATED WORKS

### A. Automated Robotic Demonstration Generation

Demonstration generation has emerged as a critical tool for addressing the large data requirements of robot learning. [4] first proposed using manually labeled anchor points tied to object locations to programmatically warp recorded trajectories to generate novel demonstrations, and used this method to assist in training an RL agent, both through replay buffer spiking [7] and through curriculum learning [8]. MimicGen [6] employed a similar approach to programmatically adapt a set of recorded demonstrations, using human annotations to separate each demonstration into object-specific sub-trajectories, then warp each sub-trajectory based on the change in the relevant object’s pose. MimicGen evaluated their method using evaluation tasks and imitation learning benchmarks from RoboMimic [9]. Similar to MimicGen, OneACTPlay also used data generation (the method from [4]) to train imitation learning agents using only a single human demonstration. Additionally, this work developed a novel uncertainty-based addition to Action Chunking Transformer’s (ACT) [10] temporal ensembling method to better handle out-of-distribution states. Additional follow-up works to MimicGen have extended this methodology to bimanual manipulation [11] and have incorporated hybrid control, using motion planners to assist in the generation of new trajectories [12]. However, as our work does not explore bimanual manipulation, and we seek to avoid human annotation, hard-coding, and embodiment-specific methods, we use [5] and [6] as our baselines.

### B. LLMs for Robot Control and Data Generation

Recent advances in Large Language Models (LLMs) have enabled transformative progress in many aspects of robot control [13]–[15]. Early work, such as SayCan [1], combined LLM-based reasoning with robot affordance grounding to connect abstract language instructions to predefined robot skills. More recent works have extended LLM-based control to a wide range of methodologies, ranging from code-based control [16] to path and task planning [17], [18]. In addition to direct robot control, LLMs have been used to

assist in policy training, with a key avenue being through improved data generation. [19] used LLMs to automate the generation of 3D assets, task descriptions, and reward functions in simulation, creating novel tasks to train RL agents. Similarly, [20] uses LLMs to iteratively generate novel simulated tasks, modifying an existing environment either to adapt it to meet a desired task description or to explore novel tasks. [21] extends this work to handle long-horizon tasks with articulated objects. Likewise, [22] uses LLMs to automate task generation for humanoid robots. As with [19], these works utilize their generated environments to train robotic agents, using reinforcement learning, pre-trained solvers, motion planning, or LLM-generated expert code. However, none of these methods explicitly leverage human demonstrations or the keypoint-based demonstration trajectory transformations used by MimicGen and its related family of works. In our paper, we seek to close this gap by combining LLMs with demo augmentation-based data generation.

## III. METHODS

### A. Key Assumptions

As with [5], [6], we assume that we have a set of demonstrations,  $\mathcal{D} = \{D_1, D_2, \dots, D_N\}$ , each consisting of a set of observations  $O = \{o_t\}_{t=1}^T$ , robot poses  $P = \{p_t\}_{t=1}^T$ , and actions  $A = \{a_t\}_{t=1}^T$ . Additionally, we assume that the actions are in the form of goal positions and the observations consist of image observations of the scene, the pose of key objects in the scene, and the robot’s pose. We also assume that we have access to an initial observation from each new scene,  $o_1^{\text{new}}$ , including both image observations and object poses. Because this observation only applies to the initial observation of the new scene, we can extract this observation data using annotation tools that would not be practical during operation (i.e., LLM Description + SAM [23] + Grounding Dino [24]). Finally, we assume a short (one or two sentence) description of the task,  $s$ .

### B. Identifying and Modifying Key Poses with LLMs

We can view key point identification and modification as a functional mapping of a demonstration, task description, and new observation to a modified set of key points  $\mathcal{F} : (D, s, o_1^{\text{new}}) \mapsto K'$ .

We can decompose this function into two subfunctions, one which extracts keypoints and a separate function that modifies these keypoints  $\mathcal{F} : (D, s) \mapsto K$ ;  $\mathcal{G} : (D, s, o_1^{\text{new}}, K) \mapsto K'$ . Since only  $\mathcal{G}$  requires information on the new scene, we only need to run  $\mathcal{F}$  once during data generation. We can take further advantage of this by having  $\mathcal{F}$  return processed demonstration information  $\hat{D}$ , which can replace the original demonstration  $D$ , thereby shifting this processing from  $\mathcal{O}(T)$  to  $\mathcal{O}(1)$ . Combining these two modifications, we define the following composite functions:  $\mathcal{F} : (D, s) \mapsto K, \hat{D}$ ;  $\mathcal{G} : (\hat{D}, s, o_1^{\text{new}}, K) \mapsto K'$ .

We leverage the world knowledge of LLMs for keypoint identification and mapping, using LLMs to perform  $\mathcal{F}$  and  $\mathcal{G}$ . A detailed diagram of our approach can be seen in Fig. 2. To

perform these functions using an LLM, we must convert the demonstration into a compatible form (vision and text) and convert the LLMs output (text) to keypoints. Although demo conversion is straightforward (the visual observations can be presented as images, and the robot and object poses for each timestep can be converted to text), the resulting LLM input would be far too large (especially the images), as each demo has hundreds of timesteps. Instead, we can have the LLM choose which timesteps are worth viewing. We provide the LLM with the initial and final observations from a single scene overview camera and the pose of the robot and the objects throughout the demo.

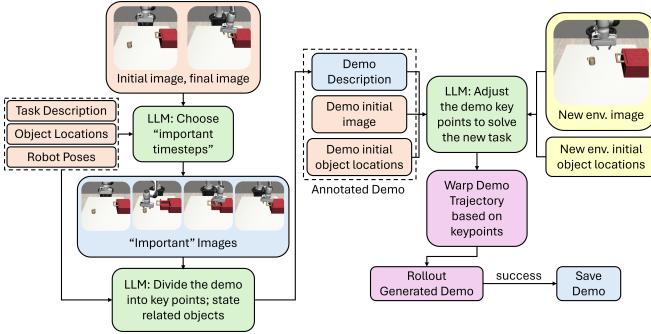


Fig. 2: Diagram of the LLM-based demo augmentation method. First, the LLM is used to annotate the demonstration, noting key points and the relationship between these points and objects in the scene. Next, the LLM uses this annotation to adapt the key points to a new scene and uses the new key points to warp the recorded trajectory. The recorded trajectory is rolled out, and if successful, saved.

To reduce the amount of text the LLM has to parse, we only provide the object poses every 5 timesteps (approximately). We add a small amount of noise to this sampling process, so the list of timesteps is slightly different for each run to keep the LLM from attaching to specific timestep numbers. All poses are presented to the LLM as a translation (mm) and a rotation expressed in Euler angles (deg). Additionally, the end-effector’s rotation is presented as the relative rotation (in the world coordinate frame) of the end-effector from its home pose, which we define as the first pose of the first demonstration in  $\mathcal{D}$ .

$$\mathbf{R}_{\text{LLM},t} = \mathbf{R}_{\text{home}}^{-1} \mathbf{R}_t \quad (1)$$

We found that the LLM struggles with reasoning about rotation, and using relative rotation mitigates the impact of unintuitive default rotations.

Given this demonstration summary and the task description, the LLM responds with a list of timesteps it views as important. Images and object poses for these timesteps, along with the full robot trajectory and the summary for the task, are passed to the LLM (a new instance, separate from the prior query), which returns a description of the demonstration, including a list of key poses (both the timestep and the pose), a list of important objects for each key pose, and instructions on how the key poses should be modified in

response to alterations in these objects’ poses. Occasionally, the LLM will not have perfect recall and incorrectly list the robot’s pose at a key point. In these cases, we replace the listed key pose with the recorded robot pose at the specified timestep (both in the extracted list of key poses and the text of the demo description). Additionally, we add the initial and final poses of the demonstration as key poses if the LLM did not list these points. If the LLM includes any poses with a timestep outside of the range of the demonstration, fails to match the response formatting criteria, or some other error occurs during processing, we discard the LLM responses and restart the annotation generation process. The final key point list,  $K$ , and processed demo description text are combined with the original demonstration information to form an *annotated demo* and saved.

Next, we use the annotated demo to perform  $\mathcal{G}$  using an LLM. First, we initialize a new environment and capture the initial observation  $o_1^{\text{new}}$ . From this observation, we extract an image of the new scene and the initial poses of the robot and the objects in the scene. We then combine the initial robot pose, initial object poses, initial image, final image, and demo description text (which includes the key points) from the demo annotation to form  $\hat{\mathcal{D}}$ .  $\hat{\mathcal{D}}$ , the task description,  $s$ , and the processed initial observation,  $o_1^{\text{new}}$ , are combined and passed into the LLM, which responds with a list of new keypoints,  $K'$ , adjusted based on the new observation in accordance with the instruction included in  $\hat{\mathcal{D}}$ . These new keypoints are then used to warp the demonstration trajectory, and the warped trajectory is executed in the new environment to generate a novel demonstration.

### C. Keypose-Based Trajectory Warping

We use the same warping mechanism proposed in [5], with a minor addition to handle end-effector rotation. Given a demonstration trajectory and a new start and end position,  $\mathbf{p}_0^{\text{new}}$  and  $\mathbf{p}_T^{\text{new}}$ , we generate the rigid transform,  $\mathbf{T}_{\text{warp}}$ , which, when applied to the demonstration trajectory, aligns the demonstration start and endpoints,  $\mathbf{p}_0^{\text{old}}$  and  $\mathbf{p}_T^{\text{old}}$ , with the new start and end positions. This transform has an extra degree of freedom (rotation about the axis from the start point to the end point), which we use to ensure that the transformed z-axis is aligned with the world’s z-axis,  $\hat{\mathbf{z}}$ .

$$\mathbf{T}_{\text{warp}} = \begin{bmatrix} \mathbf{R}_{\text{warp}} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \quad \text{s.t.} \quad \begin{cases} \mathbf{p}_0^{\text{new}} = \mathbf{T}_{\text{warp}} \mathbf{p}_0^{\text{old}} \\ \mathbf{p}_T^{\text{new}} = \mathbf{T}_{\text{warp}} \mathbf{p}_T^{\text{old}} \\ \max \hat{\mathbf{z}}^\top \mathbf{R}_{\text{warp}} \hat{\mathbf{z}} \end{cases} \quad (2)$$

where  $\mathbf{p}$  is a position in homogeneous coordinates.

This transformation is then applied to the demonstration trajectory to form a new trajectory:

$$\mathbf{p}_t^{\text{new}} = \mathbf{T}_{\text{warp}} \mathbf{p}_t^{\text{old}} \quad (3)$$

When working with robot poses (position + rotation), we apply a similar methodology to warp the robot’s positions. For rotations, we calculate the delta rotation between the

demonstration trajectory and the new trajectory for the start and end points:

$$\Delta \mathbf{R}_0 = \mathbf{R}_0^{\text{new}} (\mathbf{R}_0^{\text{demo}})^{-1} \quad \Delta \mathbf{R}_T = \mathbf{R}_T^{\text{new}} (\mathbf{R}_T^{\text{demo}})^{-1} \quad (4)$$

then linearly interpolate between them to form a delta rotation for each timestep in the trajectory segment  $\Delta \mathbf{R}_t$ . Finally, these delta rotations are applied to the demonstration rotation to form the warped rotation:

$$\mathbf{R}_t^{\text{new}} = \Delta \mathbf{R}_t \mathbf{R}_t^{\text{demo}} \quad (5)$$

#### D. Data Collection via Multi-Armed Bandit Optimization

To create a dataset to train a robotic agent, we repeat the above demo augmentation method many times, rolling out each generated trajectory and saving successful runs as novel demonstrations. As discussed above, the demo generation method consists of two steps: first, we annotate the given demonstration, then we generate a new demonstration using this annotation and the initial observation from the new environment. Because this annotation includes both the key poses to be warped and the instructions for how to warp them, the success of the second step relies heavily on a high-quality annotation. As such, if we wish to optimize the data generation process, we must generate, identify, and use a high-quality annotation. With this in mind, we cast the data generation process as a modified multi-armed bandit problem. Here, each "arm" is a demo annotation, and our goal is to determine which arm (annotation) we should pull (use to generate a demo) to maximize overall reward (total number of successful demos), balancing exploration (experimenting to determine which annotation is best) with exploitation (using the best annotation). In this process, we are optimizing the success rate of our generated demo (ie. minimizing the number of rollouts required to generate a set number of successful demonstrations). This ignores the cost of running the LLM. We believe that this is the right choice for our analysis, as our emphasis is on robot demo generation, reducing the number of robot rollouts will (generally) reduce the amount of LLM compute, and equating LLM compute to robot rollouts in a general case is infeasible.

Because this multi-armed bandit problem has a binary reward (each rollout either succeeds or fails), we can employ Thompson Sampling [25]. This method models each arm as a beta distribution, where the success rate of each arm is treated as a Bernoulli process. We assume that the prior distribution is uniform, resulting in a beta distribution for each arm of the form:

$$P(\text{arm}_i) \sim \text{Beta}(n_{\text{suc},i} + 1, n_{\text{fail},i} + 1)$$

where  $n_{\text{suc},i}$  is the number of successes and  $n_{\text{fail},i}$  is the number of failures recorded for arm  $i$ .

At each iteration, we sample a success probability from each annotation's beta distribution and select the annotation with the highest sampled value to generate the next demonstration. This approach naturally balances exploration and exploitation: underexplored annotations with high uncertainty have a greater chance of being selected early on,

while consistently successful annotations are chosen more frequently over all. As more demonstrations are collected, the probability estimates converge, leading to more reliable selections and an improved overall success rate in data generation.

However, our problem differs from the standard multi-armed bandit formulation in that, at each step, we have an additional option - we can attempt to create a new arm (generate a new demo annotation). To create a new arm, we sample a random demonstration from  $\mathcal{D}$ , and use it to generate a new demo annotation. We then immediately "pull" this new arm (generate a new trajectory using the annotation, roll it out, and see if it is a success). If the new arm results in a success, we add it to the bandit. If not, we discard the arm. To determine whether we should pull an existing arm or generate a new arm, we must determine the expected value of each operation, keeping in mind the effect of including a new arm on all future Thompson Sampling steps. If we are going to sample the bandit  $T$  more times, the expected value of adding a new arm is:

$$\mathbb{E}_{\text{add}} = P_{\text{add}} \cdot (1 + \mathbb{E}_{T-1}(\mathcal{P} \cup \{p_{\text{new}}\})) + (1 - P_{\text{add}}) \cdot \mathbb{E}_{T-1}(\mathcal{P})$$

where  $P_{\text{add}}$  is the probability of successfully adding an arm (getting a success on the first pull of a new arm), which we estimate from previous attempts to create a new arm, and  $\mathbb{E}_T(\mathcal{P})$  is the expected value of running Thompson sampling for  $T$  iterations using arms with success probabilities  $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$ .

Therefore, we should try sampling a new arm if and only if  $\mathbb{E}_{\text{add}} > \mathbb{E}_T(\mathcal{P})$ . To evaluate this, we must find a suitable method to estimate the new arm's success rate  $p_{\text{new}}$ , and determine the estimated value of a Thompson Sampling rollout. We model the distribution of arm success rates as a beta distribution. However, directly fitting a beta distribution to the observed arm success rates is challenging, as we only have a small number of arms (as few as two). Instead, we leverage the fact that we have an estimated success rate distribution for each arm, and fit the new arm prior to these existing success rate distributions. Formally, this means finding the beta distribution parameters  $\hat{\alpha}$  and  $\hat{\beta}$  that satisfy:

$$(\hat{\alpha}, \hat{\beta}) = \arg \max_{\alpha, \beta} \prod_{i=1}^n \mathbb{E}_{x \sim \text{Beta}(\alpha_i, \beta_i)} \left[ \frac{x^{\alpha-1} (1-x)^{\beta-1}}{B(\alpha, \beta)} \right] \quad (6)$$

where  $\alpha_i$  and  $\beta_i$  are the beta distribution parameters for arm  $i$ .

Practically, we evaluate this by sampling a large number ( $m$ ) of success rates from each of the  $n$  arms, then fitting a Beta distribution to the resulting  $n \cdot m$  samples.

To evaluate the estimated value of a Thompson sampling rollout  $\mathbb{E}_T(\mathcal{P})$ , we use a brute-force large-number approximation. We begin by sampling  $k$  sets of discrete probabilities from the arm success rate distributions (the beta distributions for each arm used for Thompson Sampling). For each of these  $k$  sets, we run a virtual Thompson Sampling rollout for  $T$  iterations, using the  $k$  sampled probability sets as ground-

truth success rates. If we are evaluating a newly generated arm, then the success probability for the arm is sampled from the arm prior distribution (equation 6), and the arm is initialized with 1 success and 0 failures (since for a newly generated arm to be used, its first pull must be successful). When determining if a new arm should be added, we need to run three of these rollouts:  $\mathbb{E}_T(\mathcal{P})$ ,  $\mathbb{E}_{T-1}(\mathcal{P})$ , and  $\mathbb{E}_{T-1}(\mathcal{P} \cup \{p_{\text{new}}\})$ . To reduce the impact of the sampling method, we use the same  $k$  samples of  $\mathcal{P}$  for all three calculations and choose a large value for  $k$  (1000). This method requires an estimate of the total number of Thompson Sampling steps that will be performed,  $T$ . However, we typically run the data generation system until we reach a total number of successful demonstrations. In this case, we approximate  $T$  as  $T \approx (n_{\text{suc, goal}} - n_{\text{suc, cur}})/\max(\mathcal{P})$ .

### E. Ensembling

Due to the multi-armed bandit optimization method, our LLM feedforward control scheme serves as a capable agent after data collection, reaching success rates on par with and even surpassing those of trained IL agents. Despite their strong performance, these policies have a fatal flaw - as a pure feed-forward method, they cannot respond to disturbances in the environment. The trained IL agent solves this problem, but struggles with longer horizon tasks and generalizing to out-of-distribution observations - two tasks that the LLM-based feed-forward policy excels at. We take advantage of the complementary strengths of these two policies by ensembling them together. An outline of our ensembling method can be seen in Fig. 3. Since the feedforward policy excels at high-level planning, the ensembled agent begins by executing the LLM feedforward trajectory. This continues until the ensembled agent detects that the feedforward policy has made an error, at which point the agent switches to the IL policy to correct the error. Once the error is fixed, the ensemble policy can "reattach" to the feedforward trajectory at an appropriate point and continue executing the pre-planned trajectory until another error is detected. A simple heuristic for determining when the feedforward policy has made an error is the similarity of the action predicted by the feedforward and feedback policies. If the two policies are in substantial prolonged disagreement, then it stands to reason that an error has occurred and the feedback policy should be used. To measure similarity, we use a magnitude-aware cosine similarity between normalized actions (normalization allows us to combine different action modalities).

$$\text{sim} = \frac{2 \min(\|\mathbf{a}_{LLM}\|_1, \|\mathbf{a}_{IL}\|_1)}{\|\mathbf{a}_{LLM}\|_1 + \|\mathbf{a}_{IL}\|_1} \cdot \frac{\mathbf{a}_{LLM} \cdot \mathbf{a}_{IL}}{\|\mathbf{a}_{LLM}\|_2 \|\mathbf{a}_{IL}\|_2} \quad (7)$$

After the IL policy recovers from an error, the agent attempts to rejoin the feedforward trajectory. To do this, it evaluates each future point on the recorded trajectory as a potential reattachment target. For each point, we compute (i) a reattach action—the action needed to move from the agent’s current pose to the candidate goal pose—and (ii) the recorded action that the feedforward agent would take at that

timestep (based on the recorded trajectory). We then select the first point that satisfies two conditions: (1) the reattach action is similar to the IL policy’s current action, and (2) the recorded action is also similar to the IL policy’s current action. Among all such points, we choose the one with the highest reattach action similarity. To prevent rapid switching, we enforce a short cooldown (5 timesteps) after each policy change before another can occur.

$$t^* = \arg \max_t \text{sim}(\mathbf{a}_t^{\text{att}}, \mathbf{a}^{\text{IL}}) \text{ s.t. } \begin{cases} \text{sim}(\mathbf{a}_t^{\text{att}}, \mathbf{a}^{\text{IL}}) > \tau \\ \text{sim}(\mathbf{a}_t^{\text{rec}}, \mathbf{a}^{\text{IL}}) > \tau \end{cases} \quad (8)$$

where  $t^*$  is the reattach timestep,  $\mathbf{a}^{\text{IL}}$  is the action predicted by the imitation learning agent,  $\mathbf{a}_t^{\text{att}}$  is the reattach action,  $\mathbf{a}_t^{\text{rec}}$  is the recorded action, and  $\tau$  is a similarity threshold.

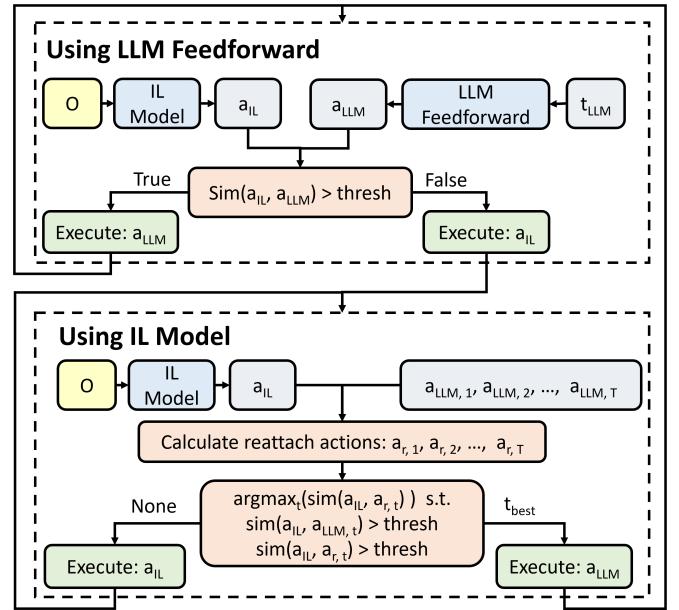


Fig. 3: Outline of the ensembling method. The agent begins by running the LLM-based feedforward policy. When the agent has made an error (as determined by increased disagreement between the two policies), the IL agent takes control. Once the agent returns the feedforward path, the feed-forward agent retakes control.

### F. Hardware Implementation

As described in section III-A, our system assumes that the observations,  $O$ , in our set of human demonstrations  $\mathcal{D}$ , along with the initial observation,  $o^{new}$ , of the new scene, contain the poses of the objects in the environment. Although these observations are easy to access in simulation, if we wish to extend our method to hardware, we need some way to calculate these poses. Furthermore, since the goal of this work is to develop a task-agnostic method for data generation that does not require manual human annotation (aside from a brief textual description of the task), our method for gathering  $O$  must also be task-agnostic and fully automated. To achieve this, we implement a system similar to one used

in [2], using a combination of LLM descriptions, SAM [23], and Grounding DINO [24] to extract object locations from RGB-D (color + depth) images of the scene.

We assume that at least some portion of the image observations from each timestep,  $\{i_t^d\} \in o_t$ , are RGB-D images (color + depth), and that these RGB-D images are sufficient to observe the full scene. As with the simulation experiments, we also assume one of the images provides an overview of the full scene,  $i_t^o \in o_t$  (this image does not have to contain depth information). First, we use the LLM to identify the relevant objects for the task, requesting both the name and color of the desired objects. With this list of relevant objects, we can use SAM [23] and Grounding DINO [24] to extract a segmentation mask of each object from the RGB-D images,  $i_t^d$ , which we then use to create a point cloud of the objects.

For our method, we need to calculate the pose of each object for each observation in the human demonstrations and for the first observation of each new scene. To do this, we first define a reference point cloud for each object. Any observation from a human demo,  $o_t$ , can be used for this, so long as the relevant object is visible (detected by Grounding DINO). For our implementation, we default to the first observation in the demo where the object is visible. We define the pose of the reference point cloud, setting its rotation to zero and its position to the center of the point cloud. To determine the pose of an object in a new observation, we generate its point cloud, then use RANSAC [26] and ICP [27] to determine the transformation from the reference point cloud to the observed point cloud, and apply this transformation to the reference point-cloud's pose. This method helps to ensure consistency in pose measurements across observations.

$$P_{\text{obs}} = T_{\text{obs}}^{\text{ref}} P_{\text{ref}}, \quad P_{\text{ref}} = \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{c}_{\text{ref}} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \quad (9)$$

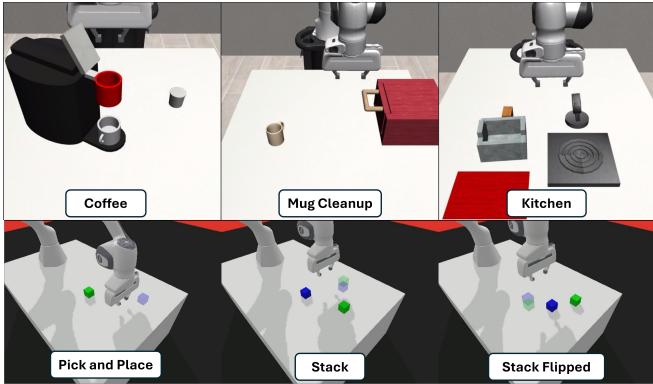


Fig. 4: Simulation evaluation tasks. Top: RoboMimic tasks from the MimicGen benchmark. Bottom: PandaGym tasks from the OneACTPlay benchmark

#### IV. RESULTS

To evaluate our proposed method, we use three of the RoboMimic [9] simulation environments from MimicGen

[6], and the block pick-and-place and block stacking PandaGym [28] tasks from OneACTPlay (we did not use the slide task as it is just a simplified case of the pick-and-place task). Additionally, we included two variations of the block stacking task: block stack flipped and block stack walking. In the “flipped” version of the stack task, the order in which the blocks must be stacked is flipped for 50% of runs. The only way this change is conveyed to the agent is the switching of the colors of the transparent goal location blocks in the agent’s visual observation - there is no explicit text or flag of any kind informing the agent of the change. This task was included to illustrate the reasoning capabilities of the LLM. In the walking version of the stack task, the blocks undergo a random walk, moving 0.4mm in a random direction each timestep while on the ground. Once they are picked up by the robot, they stop moving. This task was included to see how well the trained agents handled time-varying tasks, and was used to evaluate agents trained on the regular stack task. A diagram showing the evaluation tasks can be seen in Fig. 4. For these tasks, we used the same human demonstrations used in [6] and [5]. Therefore, for the RobotMimic tasks, we used 10 human demonstrations, and for the OneACTPlay block manipulation tasks, we used a single human demonstration.

Using our proposed method, we generate 1000 successful demos for each MimicGen task, and 400 successful demonstrations for each OneACTPlay task (these values were chosen to match the baselines). We report the generation success rate when using the best annotation found during the multi-armed bandit exploration, the average generation success rate when using a new annotation (this would be the success rate of our method if we did not employ RL optimization), and the overall success rate for the full data generation process, which reflects both the exploration and exploitation steps of the optimization process. Additionally, we used these generated demonstrations to train imitation learning agents and reported the trained agent’s performance on each task for a variety of different numbers of demonstrations. For evaluation, we calculate the success rate using 50 evaluation rollouts and report the mean and standard deviation across 5 trials (the policy is trained separately for each trial). Our results, and the comparison to baseline, can be seen in Table II. For the MimicGen-trained agent success rates, we follow the advice from their repository and retrain and re-evaluate the agents using their data, rather than use the success rate from the paper, due to discrepancies in the simulator. For OneACTPlay, we found an error in the stack task simulation, which gave occasional false positives. As such, we tightened the success cutoff for this task and reran the evaluations.

*1) Data Generation:* The simulation results show that our LLM-based data generation method significantly outperformed the MimicGen and OneACTPlay baselines, with the optimized LLM achieving a higher success rate on all tasks. Additionally, the total success rate, which includes the exploration phase of the optimization method, also beat the baseline in all tasks except for the Stack task. Importantly, our method achieved this improvement over the previous

TABLE I: Task descriptions used in our experiments.

Task	Description
Mug Cleanup	Open the drawer, pick up the mug, place it in the drawer, then close the drawer.
Kitchen	Turn on the stove, place the pot on the stove, add the bread to the pot, move the pot from the stove to the serving area, then turn off the stove.
Coffee	Pick up the coffee pod, place it into the coffee machine, and close the coffee machine lid.
Stack	Place the two blocks in the highlighted region, stacking the green block on top of the blue block.
Stack Color Flip	Place the two blocks in the highlighted region, stacking them on top of each other.
Pick & Place	Pick up the green block and place it in the blue highlighted region.

TABLE II: Data generation success rates (%) on MimicGen and OneACTPlay tasks.

Task	No RL	Best Annotation	Total	Baseline
Mug Cleanup	12%	44%	36.1%	29.5%
Kitchen	20%	60%	50.5%	42.7%
Coffee	39%	89%	82.6%	78.2%
Pick & Place	65%	92%	84.5%	81.7%
Stack	34%	89%	79.1%	80.0%
Stack Flipped	14%	58%	40.9%	N/A

methods despite the prior methods using a human expert to manually annotate the trajectories. In contrast, our method required **no** human input other than a short (one sentence) description of the task. Despite this major limitation, our system outperformed the baselines, showing that through our optimization process, LLM Trainer can outperform human experts.

Our success rate results also illustrate the importance of our optimization method. For all tasks, we found that using the best annotation found using our optimization method led to a 2-3× increase in generation success rate.

Finally, the alternating stack task illustrates how the reasoning of the LLM allows our method to be more generalizable. Unlike the baseline methods, which rely on hardcoded object-centered transforms, our method is able to adapt based on purely visual cues. The LLM is able to reason that if the shaded green block appears on top of the shaded blue block, then the order in which the blocks are stacked must be flipped. Our system is then able to dynamically adjust its choice of waypoints accordingly - a level of flexibility that prior methods lacked.

2) *Trained Model Performance:* Comparing the performance of imitation learning agents trained on data generated by our method to that of agents trained on data generated by the baseline methods, we see that our method performs roughly the same as MimicGen, and moderately outperforms OneACTPlay. These results show that our method generates high-quality data useful for training imitation learning policies. Somewhat unexpectedly, models trained using our data consistently outperformed identical models trained using data generated using OneACTPlay’s method. We think this

improvement is due to OneACTPlay’s reliance on a single demonstration. Although our method also only uses a single human demonstration, the natural stochasticity of the LLM generation process increases the variation of the generated dataset, thereby making it more useful for training. We believe we didn’t see a similar result for MimicGen due to that method’s use of 10 human demonstrations, which adds additional variety to the baseline.

3) *Ensembling Performance:* Examining the success rate of our ensembling method, we see that ensembling significantly improved the performance of IL agents in low data regimes, but occasionally hindered them in high data regimes where the trained IL agent was able to solve the task on its own. Interestingly, this decrease in performance did not apply to the block manipulation tasks. We believe this discrepancy stems from the fact that block manipulation failures are recoverable, while many failures in the MimicGen tasks are unrecoverable. For example, if the robot drops a block, it can easily pick it back up, while if the robot drops a mug and the mug lands on its side, the robot will be unable to retrieve it. Therefore, on the MimicGen tasks, the ensembled policy can be limited by the performance of the feedforward agent - once the ensembled agent realizes the feedforward agent has made a mistake, it may already be too late.

Comparing the ensembled agent’s performance to that of a pure feedforward controller (best annotation from the data generation process), we see that the ensembled agent often underperforms. This is in part due to the static nature of our evaluation tasks, which makes them well suited for feed-forward agents. If the scene is dynamic, however, the lack of feedback control quickly becomes detrimental. To explore this, we evaluated the LLM feedforward agent on the walking stack task. In this environment, the LLM feedforward agent achieved a success rate of only 14%, illustrating the brittleness of these feedforward policies.

#### A. Hardware Experiments

To validate that our data generation method works on hardware, we evaluated it using a Franka Emika Panda robot, completing a mug-cleanup task. In this task, the robot must open a drawer, pick up the mug from the scene, place it in the drawer, and then close the drawer. The location of the mug in the scene was randomized (placed in a 20cm × 30cm region in front of the robot), as was the rotation of the mug ( $\pm 45$  deg). A diagram of this task can be seen in Fig. 5.

TABLE III: Task success rates (%). Values are mean  $\pm$  standard deviation

Task	50 demos			100 demos			200 demos			500 demos			1000 demos		
	MimicGen	Ours IL	Ours Ens	MimicGen	Ours IL	Ours Ens	MimicGen	Ours IL	Ours Ens	MimicGen	Ours IL	Ours Ens	MimicGen	Ours IL	Ours Ens
Mug Cleanup	21.2 $\pm$ 5.7	24.8 $\pm$ 6.9	33.6 $\pm$ 7.1	37.2 $\pm$ 6.9	42.0 $\pm$ 6.1	38.0 $\pm$ 4.0	50.0 $\pm$ 5.2	52.4 $\pm$ 5.3	42.8 $\pm$ 5.7	61.6 $\pm$ 7.8	60.4 $\pm$ 10.1	46.0 $\pm$ 2.2	65.6 $\pm$ 5.9	65.6 $\pm$ 3.2	58.0 $\pm$ 5.2
Kitchen	92.0 $\pm$ 5.1	88.8 $\pm$ 9.0	56.4 $\pm$ 5.0	98.4 $\pm$ 2.3	88.8 $\pm$ 1.6	60.0 $\pm$ 2.8	99.6 $\pm$ 0.8	94.4 $\pm$ 4.1	62.8 $\pm$ 2.0	98.8 $\pm$ 1.6	96.0 $\pm$ 4.2	63.2 $\pm$ 2.0	100.0 $\pm$ 0.0	92.4 $\pm$ 4.1	63.6 $\pm$ 3.4
Coffee	84.0 $\pm$ 5.2	86.4 $\pm$ 5.4	93.6 $\pm$ 2.3	92.4 $\pm$ 3.2	89.6 $\pm$ 2.7	93.2 $\pm$ 3.2	92.4 $\pm$ 4.5	96.0 $\pm$ 2.8	96.4 $\pm$ 0.8	95.6 $\pm$ 2.9	95.2 $\pm$ 4.1	96.4 $\pm$ 2.0	96.4 $\pm$ 2.3	99.2 $\pm$ 1.6	98.8 $\pm$ 1.0
Task	25 demos			50 demos			100 demos			200 demos			400 demos		
	ACT	Ours IL	Ours Ens	ACT	Ours IL	Ours Ens	ACT	Ours IL	Ours Ens	ACT	Ours IL	Ours Ens	ACT	Ours IL	Ours Ens
Stack	0.0 $\pm$ 0.0	0.8 $\pm$ 1.0	10.4 $\pm$ 9.4	2.5 $\pm$ 0.9	0.4 $\pm$ 0.8	32.4 $\pm$ 10.4	5.6 $\pm$ 3.4	20.8 $\pm$ 5.5	71.2 $\pm$ 8.0	24.4 $\pm$ 6.7	30.4 $\pm$ 14.7	86.0 $\pm$ 5.4	46.0 $\pm$ 5.9	87.2 $\pm$ 2.7	96.4 $\pm$ 0.8
Pick & Place	17.6 $\pm$ 5.0	19.2 $\pm$ 7.4	50.0 $\pm$ 6.3	54.4 $\pm$ 7.2	56.4 $\pm$ 7.3	76.0 $\pm$ 7.9	73.2 $\pm$ 11.6	78.0 $\pm$ 6.1	94.8 $\pm$ 3.7	88.0 $\pm$ 4.2	88.8 $\pm$ 4.3	98.0 $\pm$ 1.3	99.6 $\pm$ 0.8	98.0 $\pm$ 1.3	98.8 $\pm$ 1.6

TABLE IV: Task success rates (%) on additional tasks. Values are mean  $\pm$  standard deviation

Task	25 demos		50 demos		100 demos		200 demos		400 demos	
	IL	Ensembled	IL	Ensembled	IL	Ensembled	IL	Ensembled	IL	Ensembled
Stack Flipped	0.4 $\pm$ 0.8	7.2 $\pm$ 4.3	2.0 $\pm$ 1.3	20.8 $\pm$ 3.5	7.2 $\pm$ 2.7	37.6 $\pm$ 6.9	13.6 $\pm$ 10.3	43.2 $\pm$ 7.4	40.8 $\pm$ 6.5	56.8 $\pm$ 6.1
Stack Walking	2.0 $\pm$ 2.2	4.8 $\pm$ 3.9	2.0 $\pm$ 1.8	17.6 $\pm$ 4.6	22.0 $\pm$ 5.1	40.0 $\pm$ 5.5	35.6 $\pm$ 13.9	53.2 $\pm$ 9.4	80.4 $\pm$ 9.8	66.0 $\pm$ 3.8

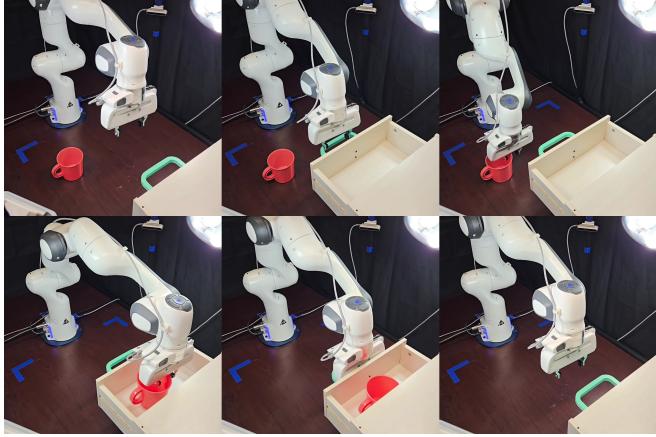


Fig. 5: Mug Clean-up task used for the hardware experiments. For this task, the robot must open the drawer, pick up the mug, place it into the drawer, and then close the drawer.

First, we used our method to generate 100 success demonstrations, generating 32 failures along the way, resulting in a total success rate of 75.8%. During this process, the best annotation (the annotation with the highest success rate) was used 95 times, generating 78 successful demonstrations - an average success rate of 82%. We also did a separate experiment to evaluate the success rate of our method without optimization, and found a success rate of 45%. We then trained an IL agent from [5] using the 100 successful demonstrations. We evaluated the IL agent, the feedforward LLM agent (using the best annotation), and the ensembled agent, running 20 trials for each. We found the IL agent had a success rate of 60%, the LLM feedforward agent had a success rate of 80%, and the ensembled agent had a success rate of 85%.

These results align with our findings from simulation, with the optimization method significantly improving the data generation success rate, and the ensemble agent performing slightly better than the LLM feedforward agent. Interestingly, the data generation worked better for our physical mug clean-up task than for the simulated version of this task. We think

this is likely because the location of the mug in simulation is not the geometric center of the mug, but rather the location of the handle. However, the location of the mug for the physical experiments was the centroid of the point cloud. We believe this more consistent location made understanding the mugs pose easier for the LLM.

## V. CONCLUSION

We introduced *LLM Trainer*, an autonomous demonstration generation system that leverages LLM world knowledge to adapt a small number of human demonstrations (as few as one) to new scenes, enabling automated collection of robot training data without manual annotation or task-specific hard-coding. In addition to removing the reliance on human input, the inclusion of LLMs enables the generation system to use world knowledge to dynamically adapt demonstrations based on context, providing our system with flexibility that human annotation-based baselines lack. Using Thompson sampling to optimize reusable LLM annotations, our method significantly increases generation success rates, improving 2 $\times$ –3 $\times$  over naïve annotations, and allowing our system to surpass expert-engineered baselines across diverse simulated manipulation tasks. The optimized LLM feed-forward policy is also a capable controller in its own right, and when ensembled with a feedback imitation learning agent, significantly boosts success rates in low-data regimes.

Crucially, we also show that the demonstrations produced by LLM Trainer are useful for robot learning. Imitation learning agents trained on the generated data achieve performance on par with baseline methods, confirming that our pipeline produces high-quality trajectories for policy learning. Finally, we validate end-to-end feasibility on hardware, demonstrating reliable data generation (75.8% average generation success rate) on a long-horizon mug clean-up task using a Franka Emika Panda Robot. Using this data, we were able to train an imitation learning agent to complete the task, and used this trained agent to validate our ensembling approach, achieving a success rate of 85%.

**Limitations and future work.** The main limitation of our work is its reliance on robot rollouts to validate proposed

trajectories and record demonstrations. While straightforward in simulation, on hardware this process is time-consuming and may require human supervision. Automated resetting and pre-execution validation of the proposed trajectory could mitigate this concern, but we leave this task to future work. Another limitation of our work is that our system is optimized to reach a target number of successful demonstrations with the fewest rollouts, ignoring LLM prompting cost and the downstream utility of an optimized feed-forward policy. Future work may seek to incorporate these factors into a more holistic optimization objective, tailored to their specific use case.

In summary, *LLM Trainer* provides a robust, task-agnostic, generalizable solution for demonstration generation, capable of producing high-quality data for robot learning from a single, unannotated human demonstration and a short description of the task.

## REFERENCES

- [1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, “Do As I Can, Not As I Say: Grounding Language in Robotic Affordances,” *arXiv preprint arXiv:2204.01691*, 2022.
- [2] A. Car, S. S. Yarlagadda, A. Bartsch, A. George, and A. B. Farimani, “Plato: Planning with llms and affordances for tool manipulation,” *arXiv preprint arXiv:2409.11580*, 2024.
- [3] A. Bartsch and A. B. Farimani, “Llm-craft: Robotic crafting of elasto-plastic objects with large language models,” *arXiv preprint arXiv:2406.08648*, 2024.
- [4] A. George, A. Bartsch, and A. B. Farimani, “Minimizing human assistance: Augmenting a single demonstration for deep reinforcement learning,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5027–5033.
- [5] A. George and A. B. Farimani, “One act play: Single demonstration behavior cloning with action chunking transformers,” *arXiv preprint arXiv:2309.10175*, 2023.
- [6] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox, “Mimicgen: A data generation system for scalable robot learning using human demonstrations,” *arXiv preprint arXiv:2310.17596*, 2023.
- [7] Z. C. Lipton, J. Gao, L. Li, X. Li, F. Ahmed, and L. Deng, “Efficient exploration for dialog policy learning with deep bbq networks\& replay buffer spiking,” *CoRR abs/1608.05081*, 2016.
- [8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 41–48. [Online]. Available: <https://doi.org/10.1145/1553374.1553380>
- [9] 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.
- [10] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, “Learning fine-grained bimanual manipulation with low-cost hardware,” *arXiv preprint arXiv:2304.13705*, 07 2023.
- [11] Z. Jiang, Y. Xie, K. Lin, Z. Xu, W. Wan, A. Mandlekar, L. J. Fan, and Y. Zhu, “Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning,” in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 16 923–16 930.
- [12] C. Garrett, A. Mandlekar, B. Wen, and D. Fox, “Skillmimicgen: Automated demonstration generation for efficient skill learning and deployment,” *arXiv preprint arXiv:2410.18907*, 2024.
- [13] P. Li, Z. An, S. Abrar, and L. Zhou, “Large language models for multi-robot systems: A survey,” *arXiv preprint arXiv:2502.03814*, 2025.
- [14] Y. Kim, D. Kim, J. Choi, J. Park, N. Oh, and D. Park, “A survey on integration of large language models with intelligent robots,” *Intelligent Service Robotics*, vol. 17, no. 5, pp. 1091–1107, 2024.
- [15] J. Wang, E. Shi, H. Hu, C. Ma, Y. Liu, X. Wang, Y. Yao, X. Liu, B. Ge, and S. Zhang, “Large language models for robotics: Opportunities, challenges, and perspectives,” *Journal of Automation and Intelligence*, vol. 4, no. 1, pp. 52–64, 2025.
- [16] 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,” *arXiv preprint arXiv:2209.07753*, 2022.
- [17] C. Merrill, A. Raman, A. George, and A. Barati Farimani, “Llmdrone: aerial additive manufacturing with drones planned using large language models,” *Construction Robotics*, vol. 9, no. 2, pp. 1–14, 2025.
- [18] J. Barkley, A. George, and A. B. Farimani, “Semantic intelligence: Integrating gpt-4 with a planning in low-cost robotics,” *arXiv preprint arXiv:2505.01931*, 2025.
- [19] P. Katara, Z. Xian, and K. Fragkiadaki, “Gen2sim: Scaling up robot learning in simulation with generative models,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 6672–6679.
- [20] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang, “Gensim: Generating robotic simulation tasks via large language models,” *arXiv preprint arXiv:2310.01361*, 2023.
- [21] P. Hua, M. Liu, A. Macaluso, Y. Lin, W. Zhang, H. Xu, and L. Wang, “Gensim2: Scaling robot data generation with multi-modal and reasoning llms,” *arXiv preprint arXiv:2410.03645*, 2024.
- [22] Z. Jing, S. Yang, J. Ao, T. Xiao, Y. Jiang, and C. Bai, “Humanoidgen: Data generation for bimanual dexterous manipulation via llm reasoning,” *arXiv preprint arXiv:2507.00833*, 2025.
- [23] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, “Segment anything,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4015–4026.
- [24] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu *et al.*, “Grounding dino: Marrying dino with grounded pre-training for open-set object detection,” *arXiv preprint arXiv:2303.05499*, 2023.
- [25] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen *et al.*, “A tutorial on thompson sampling,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.
- [26] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [27] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. Spie, 1992, pp. 586–606.
- [28] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen, “Multi-goal reinforcement learning environments for simulated franka emika panda robot,” *arXiv preprint arXiv:2106.13687*, 2021.