Extended Abstract

Motivation Training robust, generalist robotics policies has long been driving goal behind of much of robotics research. Several state-of-the art policies have been trained with imitation learning using demonstrations collected by humans operating real robots. Due to the cost and scalability issues of collecting data with real hardware, simulation has become a popular alternative method for data collection. OmniGibson, a high-fidelty simulation gym for robots in household environments, is one such example. Released recently, OmniGibson has not yet been used to train actual policies. This project therefore attempts to train a policy designed for real-world robots in the OmniGibson simulation environment, as a benchmark of the system.

Method This project consists of several key phases: task selection, data collection, data processing, policy training, and evaluation. A relatively short-horizon task called `picking_up_trash` was first selected from the BEHAVIOR-1k dataset, a collection of 1,000 common household tasks. A 3D scene containing this task, along with a simulated version of a wheeled-base bimanual robot called the R1 Pro, were then instantiated in the OmniGibson environment. 100 successful demonstrations were collected for this task, with actions provided by a human expert using a custom teleoperation controller called *JoyLo 7DOF*. These simulated demonstrations were then processed to extract actions (robot joint angles) and observations (colored point clouds and proprioceptive data) at each timestep. This data was used to train the Whole-Body Visuomotor Attention Policy (WB-VIMA), a recently published deep diffusion policy specifically optimized for the heirarchical kinematics of the R1 robot family. The policy was trained using a diffusion variant of behavior coloning (BC), with the loss being equal to the L2 distance between the predicted and actual denoising vectors. Training was conducted in the cloud on an AWS GPU instance. The trained policy was then rolled out in the OmniGibson environment for 100 episodes, with rewards measured using a sparse indicator of task completion, provided by the simulation.

Results The policy was unfortunately not successful, never achieving a reward greater than zero. Qualitatively, the agent seems to get stuck in its starting position, and when manually forced out, it behaves chaotically with highly noisy actions. Hypotheses for this behavior including inadequate training, data issues, and potential bugs in the pipeline are proposed and mittigations are attempted and analyzed.

Discussion The data clearly shows that the methods presented in this project are insufficient to train an effective policy in the OmniGibson environment: this provides some guidance for future efforts. In spite of this, the project still serves as a useful validation for OmniGibson as a data collection platform, one of its primary purposes. Based on the limitations identified, several avenues for future work are described.

Conclusion This project serves as a proof-of-concept for the OmniGibson ecosystem in some respects but not in others. Future authors looking to train policies in OmniGibson and in simulation more generally should keep the limitations identified in this project in mind and use this as a negative benchmark.

Training Robotics Policies With Imitation Learning from Simulated Teleoperation: A Proof of Concept for the BEHAVIOR-1k Project

Niklas Vainio
Department of Computer Science
Stanford University
niklasv@stanford.edu

Abstract

Imitation learning has proven to be a successful method for training performant robotics policies, but collecting data with real robots is costly and challenging to scale. Simulation environments, such as OmniGibson, have been proposed as alternative ways to gather data and train policies. To benchmark OmniGibson, this project attempts to train WB-VIMA, a diffusion policy designed for a real-world mobile bimanual robot, using behavior cloning from demonstrations collected by a human expert in simulation. The policy is evaluated in the OmniGibson environment and is unable to complete a relatively short-horizon task, suggesting that these methods are inadequate for training effective policies. Limitations are investigated and analyzed, and the resulting avenues for future inquiry are presented.

1 Introduction

Training robust, generalist robotics policies which are capable of performing a wide array of household tasks has, for a long time, been a driving goal behind of much of robotics research. A wide variety of methods for training performant robotics policies have been attempted, with techniques including model-based and model-free reinforcement learning, imitation learning, as well as more classical control algorithms.

Many of the most recent successful attempts at training robotics policies have relied on imitation learning from human demonstrations in some form: examples include π_0 [1] and *Open VLA* [4], two recently released generalist robotics policies trained largely on human-collected demonstration data. One common feature of works in this category is that they collect data and evaluate their policies on actual robots in the real world. While real-world hardware seems like the obvious choice for robotics data collection, and has the clear advantage that the training and deployment environments are identical, it also has some severe limitations. In particular, as the size of the dataset being collected grows, using real robots for data collection becomes incredibly expensive due to the physical hardware needed: state-of-the art quadruped and humanoid robots typically cost tens to hundreds of thousands of dollars each, for example. Real-world training environments are also difficult to precisely replicate in different physical locations, and resetting typically requires manual human intervention. It can also be challenging to automatically determine if a task has been completed, especially if the task is long-horizon or not rigorously defined (examples would include folding clothes or preparing a meal).

Despite these challenges, many relatively large-scale robotics datasets have been collected using physical robots: examples include DROID (Distributed Robot Interaction Dataset) [3], which collected over 700,000 trajectories for a variety of robot embodiments and RoboTurk [7], an early attempt to collect robotics data by crowd-sourcing teleoperation of a physical robot. However, attempts like these typically contain relatively little data for whole-body or humanoid embodiments, and also remain relatively limited in size, especially in comparison to the quantities data that would be required to train a true robotics foundation model.

In an effort to address the limitations of real-world robotics data collection, a variety of robot simulation environments have been developed. From a theoretical perspective, simulations offer a number of advantages: they can be scaled much more efficiently, do not require manual resets, and can extract extra information from the environment (such as ground truth object positions, or collision information) which can help analytically determine whether a task has been completed. Simulations are, of course, different to the real world so policies trained in simulation will require an additional transfer learning step to be truly useful: the hope is that with a large enough model, the efficiency improvements from performing initial pre-training in simulation will offset the additional complexity of performing the final transfer learning step.

One example of a robotics simulation environment is OmniGibson, developed by the Stanford Vision and Learning Lab and released in early 2024 [5]. OmniGibson comes bundled with a wide array of household environments and digital replicas of many common robots (such as various Franka arms, Fetch, and Galaxea.ai's R1 and R1 Pro). The environment exposes simple `step` and `get_obs` methods, enabling it to interface with any controller or policy. OmniGibson also allows instantiating tasks from the BEHAVIOR-1k dataset, a collection of over 1,000 household tasks with completion criteria defined rigorously in first-order logic over scene objects and predicates [5] (this process is described in more detail below).

OmniGibson is ultimately intended to be used to train robotics models, however given its relatively recent release, there are currently no baselines for what performance can be achieved when training a policy using data collected in the environment. **The goal of my project is therefore to provide a baseline for the performance of the system in the simplest case: training a model on a single task using vanilla behavior cloning (BC)**. The recent 'BEHAVIOR Robot Suite' paper from Jiang et al. has demonstrated that a particular policy architecture called WB-VIMA (described in more detail below) can achieve success rates of over 50% on tasks similar to the BEHAVIOR-1k dataset when trained on a real-world robot using simple BC [2] – attempting to train this same policy in the OmniGibson environment will therefore be good test of the environment and data pipeline. The success, or lack thereof, of this approach will hopefully provide useful future guidance on how to most effectively leverage OmniGibson and simulation in general to train robotics policies.

1.1 Mathematical Formulation

A task in the OmniGibson environment can be formulated mathematically as a Partially-Observable Markov Decision Process (POMDP):

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \Omega, T, O, r)$$

The state space, $\mathcal{S}$, represents the possible internal simulation states (the poses and velocities of all objects), while the action space, $\mathcal{A}$, represents the set of vectors of joint positions for the robot being operated. The observation space, Ω, is the set of all possible RGB and depth images and proprioceptive observations from the robot. The transition function, T, observation function, O, and reward function, r, are provided by the simulation environment: in this case, transitions and observations are deterministic but highly complex. In the case of OmniGibson, the reward function is a sparse indicator that outputs the change in the percentage of task predicates which are true – for example, if a task involves placing three objects inside a container, the reward will be $+0.33$ on timesteps where a new object enters the container, and 0 on others.

The objective in this context is to train a policy, $\pi_\theta(\mathbf{a}_t|\mathbf{o}_{t:t-T_h})$, that takes in a history of observations and outputs actions that maximize the sum of rewards over time by successfully completing the task.

2 Related Work

This section describes the prior work that this project is based off in more technical detail, providing technical grounding and motivations for the methods attempted in this paper.

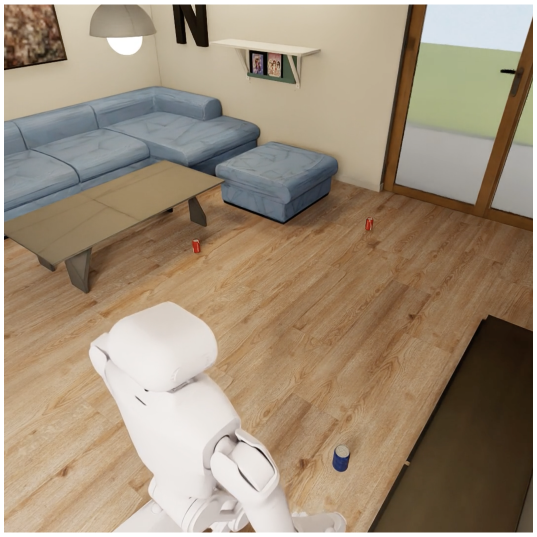


Figure 1: Example rendering of a robot in the `house_double_floor_lower` OmniGibson environment

2.1 OmniGibson and BEHAVIOR-1k

As described above, OmniGibson is a high-fidelity robotics simulation environment released by the Stanford Vision and Learning Lab in early 2024, and described in detail by Li et al. [5]. The environment is built around NVIDIA's Omniverse and Isaac Sim packages, two popular utilities for simulating real-world physical environments. OmniGibson aims to simulate realistic physical interactions of rigid and articulated bodies, fluids, and fabrics, and is also able to export rendered camera feeds in a variety of modalities [5]. By default, OmniGibson runs its physics simulation at 30 Hz [5].

OmniGibson comes bundled with 3D models for a variety of common scenes, inducing the inside and outside of houses, shops, and restaurants. It also includes a dataset of common objects, with each including a 3D model derived from real physical scans, a set of physical properties, and a classification into semantically meaningful categories called 'synsets' (examples include simple categories like `bed` and `table` and more complex ones such as `fixed_window` and `articulating_window`). A variety of robots can also be instantiated, with their simulated kinematics mirroring those in the real world.

In the same paper, Li et al. also present the BEHAVIOR-1k dataset, which contains definitions for over 1,000 household tasks in a domain-specific language called BDDL (Behavior Domain Definition Language) [5]. A BDDL description for a particular task lists the scene in which that task takes place, the 'synsets' of the objects involved, and a definition of the goal state. The goal definitions are provided in standard first-order logic, quantifying over objects in the simulation environment and using basic geometric predicates like `inside` and `on_top`, which can be automatically evaluated by the simulation [5]. As an example, the goal definition of the `picking_up_trash` task used in this paper is shown below in figure 2: it defines success as all soda can objects in the scene lying inside the trash can.

A single BDDL task can have an infinite number of possible 'realizations' in actual 3D scenes: the `picking_up_trash` task can, for example, occur in any room with cans of soda and a trash can. OmniGibson provides a tool to 'sample' a task from its description, creating a full 3D scene in which that task can be completed [5].

More information about OmniGibson is available on its dedicated documentation page.

2.2 BEHAVIOR Robot Suite

A recent paper by Jiang et al. introduces a collection of utilities called the 'BEHAVIOR Robot Suite': two of these have particular relevance to this paper, the Whole-Body VisuoMotor Attention Policy

```
(:goal
    (and
        (forall
            (?pad.n.01 - pad.n.01)
            (inside ?pad.n.01 ?ashcan.n.01_1)
        )
        (forall
            (?can__of__soda.n.01 - can__of__soda.n.01)
            (inside ?can__of__soda.n.01 ?ashcan.n.01_1)
        )
    )
)
```

Figure 2: Example Behavior Domain Description Language (BDDL) goal specification for a BEHAVIOR-1k task called `picking_up_trash`

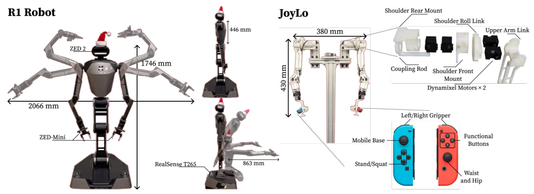


Figure 3: Images of the R1 robot and *JoyLo* interface. Sourced from Jiang et al. [2]

(WB-VIMA) architecture and *JoyLo* teleoperation interface [2]. These are used to train Galaxea.ai's R1 robot, a mobile bi-manual manipulation robot with a wheeled base, articulated torso, and two 6 degree-of-freedom (DOF) arms with parallel grippers. Data collection and policy rollouts are performed in the real world, and the model is trained on 100 demonstrations using simple BC loss. [2]

The WB-VIMA architecture is a diffusion policy designed specifically to control the R1 robot. At each timestep, the policy takes in a colored egocentric point cloud derived from stereoscopic cameras on the robot, as well as a proprioceptive state containing joint positions and and velocities. The model also takes in a noisy action chunk $\tilde{\mathbf{A}}_k$ equal to the true action chunk $\mathbf{A}_k$ plus a noise vector ϵ_k and outputs its estimate of the noise. During inference time, action chunks are sampled from pure noise, and through K denoising steps, the model produces a final estimated action chunk. This final denoising step is done using three networks: one for the base, one for the torso, and one for the arms. At each diffusion step, each network is conditioned on the output of the previous one: this accounts for the hierarchical nature of the R1's kinematics, whereby small motions of the base cause large displacements in the downstream links. [2]. The model architecture and training process are described in more mathematical detail in the 'methods' section below.

JoyLo is a custom teleoperation interface used to collect real-world teleoperation data, which Jiang et al. use to train the WB-VIMA policy [2]. *JoyLo* consists of two 3D-printed 6DOF arms attached to a frame, with each containing a Nintendo Switch JoyCon controller – the kinematics of each arm mimics the kinematics of the real R1 robot. The joint positions of the arms are measured using encoders and transmitted to the joints of the R1 in real time. The joysticks in each hand controller enable control of the mobile base and torso. The system is low-cost (with a total price of around $500) and was rated by participants as preferable to traditional spatial hand-controllers [2].

Overall, Jiang et al. demonstrate success rates of over 50% on two different tasks similar to those in the BEHAVIOR-1k dataset [2]. **This demonstrates the theoretical viability of using this model and training method to control the R1 robot. As mentioned above, all training in this paper is conducted in the real world, so I will attempt to train this policy using data from the OmniGibson simulation environment, which has not previously been attempted.**

3 Method

As described above, the main goal of this project is to replicate the prior results from Jiang et al. [2] in the OmniGibson simulation environment, as a proof-of-concept of the system. In addition to training in simulation instead of the real world, another significant change was made, namely using a newer version of the R1 called the R1 Pro, which is available in the OmniGibson environment. The

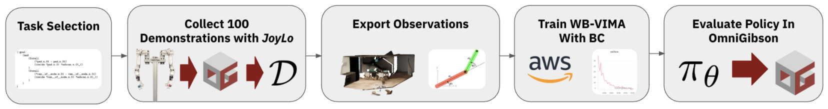


Figure 4: Method Overview.

main difference between these robots is that the R1 Pro has 7 degrees of freedom per arm instead of 6. This should, in theory, enable more versatile manipulation but also requires changes to the policy and data collection methods, which are described below.

The method for this project consists of the following key stages:

1. Selecting and sampling a BEHAVIOR-1k task
2. Collecting 100 demonstration trajectories for this task in the OmniGibson environment, using a modified version of *JoyLo*
3. Processing the data from these demonstrations into a format compatible with the WB-VIMA architecture
4. Training the WB-VIMA policy on the collected demonstrations using behavior cloning
5. Evaluating the performance of the policy in the OmniGibson environment

The first four of these steps described in turn below, while evaluation is described in detail in the next section.

3.1 Task Selection

The first step of the project was to select a task from the BEHAVIOR-1k dataset to use. I decided to use the `picking_up_trash` task, whose BDDL goal specification is shown in figure 2. The task requires the robot to drive around a living room, picking up three cans of soda (in fixed locations) and placing them in a single trash can near the starting location.

The main reason for choosing this task was that the 3D scene and objects had already been 'sampled' and validated by other members of the lab, making the process easier. The task is also an appropriate level of difficulty, requiring some manipulation and navigation but with simple geometries and a relatively short time horizon, making demonstrations quick to collect. The task is also a good test of whole-body coordination, as the robot must bend its torso over to pick up the cans and use the base to transport them: this should, in theory, play to the WB-VIMA architecture's strengths.

3.2 Data Collection

After selecting my task, I began preparing for data collection. As mentioned above, I planned to collect data for the R1 Pro instead of the R1, meaning that I would need a new version of *JoyLo* which was compatible with the different kinematics of the R1 Pro. This version, which we called *JoyLo-7DOF* (because the R1 Pro has 7 degrees of freedom per arm) was designed by my lab, and I 3D printed and assembled my own version of it. I then created and ran a script to compensate for different joint orientations and offsets resulting from slight differences in the construction process.

I then began the actual data collection: I piped the outputs of my *JoyLo-7DOF* set into the OmniGibson environment and collected 100 successful episodes of demonstration data. This number was chosen as a baseline because 100 demos was sufficient to train the WB-VIMA policy in the paper by Jiang et al. [2]. To reduce variance in the data, all soda cans were collected using the right gripper, though the order they were picked up in was varied. Omnigibson automatically records the simulation state and actions at each timestep to a `.hdf5` file.

Demonstration data was collected at roughly 15 Hz with each demo being lasting roughly 2000-3000 timesteps, or around 2-3 minutes of real time. This led to a total data collection time of around 5 hours, which was split up over multiple days.

(a) Before (b) After

Figure 5: Illustration of a Point Cloud before and after Down-Sampling.

3.3 Data Export and Processing

In order to use the collected data to train the WB-VIMA model, observations must be exported from the simulation in a format compatible with the policy. I used a custom script, which exported observations and actions from all data collection sessions and merged them into a single `.hdf5` file.

The action at each timestep is simply a 23-dimensional vector containing the base velocity (3 DOF), torso joint positions (4 DOF), and joint and gripper states for the left and right arms $2(7+1) = 16$ DOF): $\mathbf{a}_t \in \mathbb{R}^{23}$. The actions are normalized using the robot's joint limits so that each entry ranges from -1 to 1.

The easiest observations to export were proprioceptive data: at each timestep, OmniGibson automatically provides a `proprio` (proprioceptive) observation vector for the robot containing the sines and cosines of the joint angles, end effector poses, gripper states, trunk and base velocities, and the robot base link position and orientation. This vector is 68-dimensional, so at each timestep we have $\mathbf{o}_t^{\text{prop}} \in \mathbb{R}^{68}$.

Exporting the point clouds is more involved. The WB-VIMA architecture is designed to take in a single egocentric point cloud containing a fixed number of points, but the simulation only provides RGB and depth images, captured at each timestep by rendering the simulated environment with cameras at the robot's head, left wrist, and right wrist. We use these images to reconstruct a point cloud – for each pixel location (u, v) in the RGB image of each camera, we use the depth value d and the camera intrinsic matrix K (available analytically from OmniGibson) to back-project the pixel into 3D space:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = d \cdot K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{1}$$

Doing this for all pixels of each camera gives three point clouds, $\mathbf{o}_t^{\text{pcd},R}$, $\mathbf{o}_t^{\text{pcd},L}$, and $\mathbf{o}_t^{\text{pcd},H}$ each in the frame of their respective camera. To merge these point clouds, we use poses from the simulation to transform them into the robot base frame so we can take the union:

$$\mathbf{o}_t^{\text{merged}} = T_{B\leftarrow L}\mathbf{o}_t^{\text{pcd},L} \cup T_{B\leftarrow R}\mathbf{o}_t^{\text{pcd},R} \cup T_{B\leftarrow H}\mathbf{o}_t^{\text{pcd},H} \tag{2}$$

Finally, following Jiang et al. [2], we use an algorithm called 'furthest-point down-sampling' (implemented in the `fpsample` python library following this paper by Li et al. [6]) to reduce the point cloud to a fixed number of points (I chose 4096, as in the paper). This gives the final observation, $\mathbf{o}_t^{\text{pcd}} \in \mathbb{R}^{(4096\times 6)}$. Visualizations of the point clouds before and after down-sampling are shown in figure 5.

During the export process, 10 demonstrations were excluded due to recording issues, and data from the remaining 90 demonstrations was exported into a single `.hdf5` data file approximately 29 GB in size.

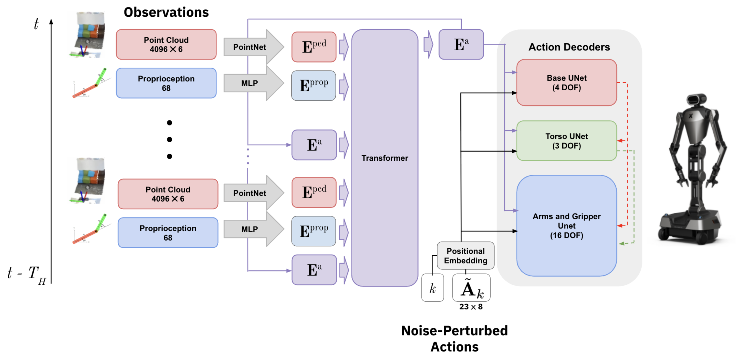


Figure 6: Diagram of the WB-VIMA architecture

3.4 Model Architecture and Training

As described above, the WB-VIMA policy takes in a colored point cloud ($\mathbf{o}_t^{\text{pcd}} \in \mathbb{R}^{(4096\times6)}$) and proprioceptive state ($\mathbf{o}_t^{\text{prop}} \in \mathbb{R}^{68}$) at each timestep. These observations are encoded through a *PointNet* [8] and standard neural network respectively to produce tokens $\mathbf{E}^{\text{pcd}}$ and $\mathbf{E}^{\text{prop}}$.

The tokens corresponding to the the T_H most recent timesteps (following Jiang et al., I used $T_H = 2$ [2]) are fed into a transformer, which performs causal self-attention and outputs a single token $\mathbf{E}^{\text{a}}$, which is fed into the decoding stage of the network.

The decoder stage takes in this token, a positional encoding of the current diffusion step k, and a noisy action chunk $\tilde{\mathbf{A}}_k \in \mathbb{R}^{(23\times T_A)}$, consisting of the actions for the last T_A timesteps (I used $T_A = 8$) with noise ϵ_k added. The noise is sampled from a Gaussian: $\epsilon_k \sim \mathcal{N}(\mu_k, \sigma_k^2)$ with μ_k and σ_k specified by the `DDIMScheduler` module from Huggingface. The decoder outputs a matrix $\hat{\epsilon}_k \in \mathbb{R}^{(23\times T_A)}$ representing its best guess of the noise vector that was added.

As mentioned above, the decoding stage consists of three cascaded U-Nets, one for the base, one for the torso, and one for the arms. Each is conditioned on the previous one as follows:

$$\hat{\epsilon}_k^{\text{base}} \sim U_{\phi_1}^{\text{base}}(\cdot|\mathbf{E}^a, \tilde{\mathbf{A}}_k, k) \qquad \hat{\epsilon}_{\text{base}} \in \mathbb{R}^{(3\times T_A)} \tag{3}$$

$$\hat{\epsilon}_k^{\text{torso}} \sim U_{\phi_2}^{\text{torso}}(\cdot|\mathbf{E}^a, \tilde{\mathbf{A}}_k, k, \hat{\epsilon}_k^{\text{base}}) \qquad \hat{\epsilon}_{\text{torso}} \in \mathbb{R}^{(4\times T_A)} \tag{4}$$

$$\hat{\epsilon}_k^{\text{arms}} \sim U_{\phi_3}^{\text{arms}}(\cdot|\mathbf{E}^a, \tilde{\mathbf{A}}_k, k, \hat{\epsilon}_k^{\text{base}}, \hat{\epsilon}_k^{\text{torso}}) \qquad \hat{\epsilon}_{\text{arms}} \in \mathbb{R}^{(16\times T_A)} \tag{5}$$

The model is trained to minimize a diffusion version of BC loss, which attempts to reconstruct the noise that was originally applied, thereby matching the action distribution to the dataset (here, $\hat{\epsilon}_\theta$ represents the entire model):

$$\mathcal{L}(\theta) = \mathbb{E}_{\substack{(\mathbf{o},\mathbf{a})\sim\mathcal{D} \\ k\sim U[0,K]}} \left[\mathbb{E}_{\epsilon_k} \left[\left|\left| \epsilon_k - \hat{\epsilon}_\theta(\mathbf{o}_{t-T_H:t}, \tilde{\mathbf{A}}_{k,t:t+T_A}, k) \right|\right|^2 \right] \right] \tag{6}$$

I set up and trained WB-VIMA on an AWS `g5.2xlarge` GPU instance. I had to modify parameters to get the architecture from Jiang et al. [2] to accept differently-sized state and action spaces, and also conducted several smaller training runs with different hyperparameters (primarily learning rate schedules). The model has roughly 7.5 million total trainable parameters, and was trained with a batch size of 192 (found to be the largest before the GPU started running out of memory). In my final training run, the loss stopped decreasing after around 26 hours of training and I terminated training.

3.5 Policy Inference

During inference, observations are extracted from the simulation at each timestep and processed in the manner described above. The policy is given the observations and an action chunk, $\tilde{\mathbf{A}}_K$, consisting of pure noise and the action distribution is the output of the denoising process:

$$\begin{aligned} \pi_\theta(\cdot|\mathbf{o}_{t:t-T_H}) &= \tilde{\mathbf{A}}_0 \\ \text{where} \quad \tilde{\mathbf{A}}_{k-1} &= \tilde{\mathbf{A}}_k - \hat{\epsilon}_\theta(\mathbf{o}_{t-T_H:t}, \tilde{\mathbf{A}}_k, k) \\ \text{for} \quad k &= K, K-1, \cdots, 1 \end{aligned} \tag{7}$$

The T_A (in my case, 8) actions in this chunk are un-normalized and fed one-by-one to OmniGibson. After all actions have been executed, inference is run again to generate the actions for the next T_A timesteps.

4 Experimental Setup

The evaluation plan is relatively simple: I will roll out the policy in the OmniGibson environment using the method described in equation 7. Actions were computed every 8 timesteps, as described above.

I allowed the policy to run in the environment for 100 episodes: each episode terminates after either 5,000 timesteps or successful task completion. Success will be measured using the total task reward (equal to the number of cans in the trash divided by 3), and if applicable, I will also collect data on how many timesteps the policy took to complete the task, with fewer being better. **There is no baseline model, as this project itself is indented to provide a baseline for the simulation suite**.

Inference was carried out on my lab workstation (the same computer used for data collection), which is equipped with an NVIDIA RTX 2080 GPU. The rollouts ran at approximately 15 timesteps per second, so the full procedure took several hours.

5 Results

After running all 100 evaluation episodes, I unfortunately observed that the agent never received a reward higher than 0. **This implies that it never successfully managed to pick up a soda can and place it in the trash**.

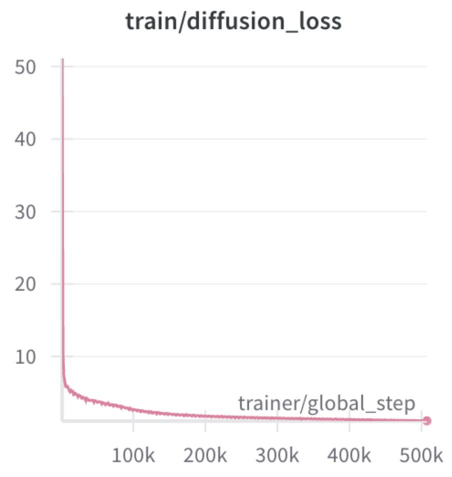
Qualitatively, the agent appears to get stuck near its starting position, applying very small velocities to the mobile base and making applying seemingly random displacements to the arms.

5.1 Analysis

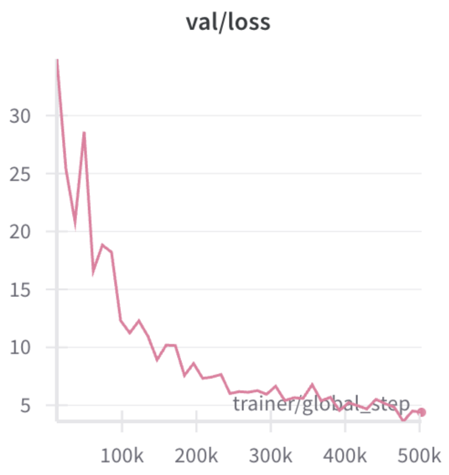
This result is, of course, disappointing and several possible explanations for the agent's poor performance are described below, as well as a few mittigation strategies that were attempted.

Static Starting States in Training Data: When starting to record a demonstration episode, it takes a few timesteps for the expert to physically grab the controllers and start inputting actions. As a result, at the start of every training episode, there are a few timesteps where the robot is in its starting location and the joints are all still in their default positions. Since the starting position of the robot is the same in every episode, a large number of these identical starting states will be present in the training data, potentially causing the policy to learn to do nothing when near the starting position. This was my initial hypothesis for why the robot did not seem to be moving, and to mitigate this, I tried manually forcing the base to move forward at its maximum speed for the first 100 timesteps to force the policy into a different part of the state space. While this did solve the problem of the robot doing nothing, the robot instead started taking noisy, chaotic-looking actions and was still unable to complete the task successfully.

Failure in Training Process: My other initial hypothesis was that the model simply failed to learn the patterns in the data during the training process, however looking at the training curves, this seems somewhat unlikely. As can be seen from figure 7, the training curves show both a decreasing diffusion loss, and a decreasing validation loss (computed as the L1 loss between the inferred and actual actions across a batch of the dataset) with a relatively low final validation loss of roughly 5 (summed across a 23-dimensional vector with entries ranging from -1 to 1). The loss curves also stop decreasing near the end of the training run, suggesting that more training time would not be an effective fix. Overfitting is a potential issue, however this would most likely cause the agent to stick closely to



(a) Diffusion Loss



(b) L1 Validation Loss

Figure 7: Training curves for the WB-VIMA model.

Table 1: Results Summary

Policy	Average Environment Reward (100 episodes)
WB-VIMA	0.0

demonstration trajectories and only perform poorly once it starts deviating, which is not exactly what we observe.

Training Data Issues: There are several potential issues with the demonstration data collected for this project, which could be contributing to poor policy performance. Due to both the human operator and physical inaccuracies in the *JoyLo* mechanism, the demonstraton data is highly noisy, potentially obscuring the signal of the actual action distribution. The training data also contains branching state trajectories, such as from demonstrations where I picked up the cans in different orders, which would prevent BC from learning an accurate action distribution for these branching states. Finally, post-processing operations on the observations, particularly the point cloud downsampling step, could mean that the observations do not enable the model to adequately perceive the environment – the soda cans are a particular concern, as they are critical to the task but are also very small, meaning that they typically only had a few points each in the point cloud. All these effects could make it very difficult for the policy to learn the correct action distribution of the dataset.

Bug in Data Preparation: A final possibility is some kind of bug in the training or evaluation pipeline causing data to appear in an unexpected format. During development, I did encounter and fix several issues in the training and evaluation pipelines (such as RGB and XYZ channels of the pointclouds being swapped, and incorrect color normalization) – although I thoroughly scrutinized the code and manually verified the correctness of the observations being passed to the model and the action post processing, it is still possible that some kind of bug remains.

Overall, I believe that the policy's poor performance is due to a combination of the high-dimensional state and action spaces, relatively limited demonstration coverage, and reliance on BC alone with no online training. These factors together likely mean that although the model seemed to train successfully, it did not learn the patterns in the expert distribution well enough to complete the task successfully. As we are running a diffusion policy, a lack of proper generalization would likely manifest as a very noisy, chaotic action distribution, which is exactly what we ibserve.

6 Discussion

The data clearly shows that the methods presented in this project are insufficient for training a policy to complete BEHAVIOR-1k tasks in the OmniGibson environment, most likely for the reasons detailed above. Although this project fails to act as a benchmark for OmniGibson in this regard, it still provides the important finding that more soiphisticated data processing, training methods, and policy architectures will likely be needed to enable OmniGibson to support the training of effective policies.

This project also acts as a validation of the data collection side of the OmniGibson ecosystem: using the simulation environment and new *JoyLo 7DOF* conrol interface, a relatively large amount of successful task demonstrations were able to be collected efficiently, and for the first time for the R1 Pro robot. The data from these demonstrations was also succcessfully able to be processed into a format compatible with a real-world robotics policy. This suggests that the pipeline in this project is, at the very least, theoretically capable of training state-of-the-art policies; the methods simply need to be improved and adjusted.

The limitations discussed in the previous section suggest several possible avenues for future work. First, the lack of state coverage and issues with static starting states suggest that incorporating online demonstration data could be helpful: this could be done with methods like DAgger, and would be particularly well-suited to a simulation like OmniGibson due to the ease of setting the environment to a particular state. Actual policy rollouts could also be inserted into the training phase to better monitor the model's performance and enable issues to be caught earlier during training. Various RL methods leveraging actual reward information could also be investigated: most common methods would require a more dense reward function to work effectively, and while this would be difficult to in general, a dense reward function could be engineered for a one off task like in this project. Alternative policy architectures could be explored, including task and motion planning (TAMP) methods, or fine-tuning a large generalist policy such as the aforementioned π_0 [1] or *Open VLA* [4] – these could even be trained across multiple OmniGibson environments and tasks, though this would require substantially more demonstration data than was collected in this project. Finally, data quality could be improved through methods such as temporal filtering to remove noise, and data generation methods like keyframe interpolation could be explored, potentially increasing the quantity of demonstration data available.

7 Conclusion

Overall, this project serves as an effective proof-of-concept for some aspects of the OmniGibson suite: it has been demonstrated that the environment can be used to collect demonstration data, and that this data can be processed into a format that can be accepted by state-of-the art robotics policies. This project has also shown that simple behavior cloning is insufficient for training policies to perform BEHAVIOR-1k tasks, suggesting that more sophisticated training methods will be required. Reasons for the poor performance have been thoroughly analyzed above, and future works should keep these limitations in mind.

8 Team Contributions

This was an individual project, so the substantive work was conducted entirely by me. This project was done in conjunction with my ongoing research with the Stanford Vision Lab, so I received technical mentorship and support from lab members and the BEHAVIOR team, as well as relying some of their existing code. In the project repository, all code outside the `external` directory was written or significantly adapted by me.

Changes from Proposal This project has changed quite significantly since the proposal. I decided to down-scope from three tasks to just one to simplify data collection, and also decided to focus on just one model due to time constraints.

References

[1] Kevin Black et al. "π_0: A Vision-Language-Action Flow Model for General Robot Control". In: *arXiv preprint arXiv:2410.24164* (2024).

[2] Yunfan Jiang et al. *BEHAVIOR Robot Suite: Streamlining Real-World Whole-Body Manipulation for Everyday Household Activities*. 2025. arXiv: 2503.05652 [cs.RO]. URL: https://arxiv.org/abs/2503.05652.

[3] Alexander Khazatsky et al. "Droid: A large-scale in-the-wild robot manipulation dataset". In: *arXiv preprint arXiv:2403.12945* (2024).

[4] Moo Jin Kim et al. "Openvla: An open-source vision-language-action model". In: *arXiv preprint arXiv:2406.09246* (2024).

[5] Chengshu Li et al. *BEHAVIOR-1K: A Human-Centered, Embodied AI Benchmark with 1,000 Everyday Activities and Realistic Simulation*. 2024. arXiv: 2403.09227 [cs.RO]. URL: https://arxiv.org/abs/2403.09227.

[6] Jingtao Li et al. "An Adjustable Farthest Point Sampling Method for Approximately-sorted Point Cloud Data". In: *2022 IEEE Workshop on Signal Processing Systems (SiPS)*. 2022, pp. 1–6. DOI: 10.1109/SiPS55645.2022.9919246.

[7] Ajay Mandlekar et al. "Roboturk: A crowdsourcing platform for robotic skill learning through imitation". In: *Conference on Robot Learning*. PMLR. 2018, pp. 879–893.

[8] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

A Policy Hyperparameters

A list of key hyperparameters used to train the WB-VIMA policy is provided in table 2 (many of these were taken from the original paper by Jiang et al. [2]).

Table 2: Policy Hyperparameters

Hyperparameter	Value
Point Cloud Dimensions	4096×6
Proprioception Dimension	68
Action Dimension	23 (3 base, 4 torso, 16 arms)
Observation Time Horizon (T_H)	2
Action Prediction Time Horizon (T_A)	8
PointNet Depth	2
PointNet Layer Size	256
Proprioception MLP Depth	2
Proprioception MLP Layer Size	256
Transformer Embedding Dimension	256
Number of Transformer Layers	2
# Attention Heads	8
Transformer Dropout Rate	0.1
Transformer Activation Function	GeGLU
Decoder position embedding dimension	128
Decoder UNet Intermediate dimensions	128, 64
Decoder UNet kernel Size	5
Number of Diffusion Steps, Training (K_{train})	100
Number of Diffusion Steps, Inference ($K_{\text{inference}}$)	16
Diffusion Beta Schedule	0.0001 to 0.02 using `squaredcos_cap_v2`
Initial learning rate	7×10^{-4}
Learning rate cosine schedule duration	10^6 steps
Batch Size	192
# Training Epochs	413
Training GPU	NVIDIA A10G
Total Trainable Parameters	approx. 7.5 million

B Task Details

The full BDDL definition of the `picking_up_trash` task used in this project is as follows:

```
(define (problem picking_up_trash-0)
    (:domain omnigibson)

    (:objects
        ashcan.n.01_1 - ashcan.n.01
        pad.n.01_1 pad.n.01_2 pad.n.01_3 - pad.n.01
        can__of__soda.n.01_1 can__of__soda.n.01_2 - can__of__soda.n.01
        floor.n.01_1 floor.n.01_2 - floor.n.01
        agent.n.01_1 - agent.n.01
    )

    (:init
        (ontop ashcan.n.01_1 floor.n.01_2)
        (ontop pad.n.01_1 floor.n.01_2)
        (ontop pad.n.01_2 floor.n.01_2)
        (ontop pad.n.01_3 floor.n.01_1)
        (ontop can__of__soda.n.01_1 floor.n.01_1)
        (ontop can__of__soda.n.01_2 floor.n.01_1)
        (inroom floor.n.01_2 kitchen)
        (inroom floor.n.01_1 living_room)
        (ontop agent.n.01_1 floor.n.01_2)
    )

    (:goal
        (and
            (forall
                (?pad.n.01 - pad.n.01)
                (inside ?pad.n.01 ?ashcan.n.01_1)
            )
            (forall
                (?can__of__soda.n.01 - can__of__soda.n.01)
                (inside ?can__of__soda.n.01 ?ashcan.n.01_1)
            )
        )
    )
)
```

C Project Source Code

Project source code is available at on Github at `https://github.com/niklas-vainio/cs224r-final-project`