LEARNING AND PRACTICE OF PICK AND PLACE BASED ON IMITATION LEARNING STRATEGY

Anonymous authors
Paper under double-blind review

ABSTRACT

In this study, we explore imitation learning algorithms for robotic arms performing the Pick and Place task, evaluated in both a simulation environment (robosuite) and a real-world environment (Ufactory X-arm 7). We trained three widely-used imitation learning algorithms: Behavioral Cloning (BC), Generative Adversarial Imitation Learning (GAIL), and Action Chunking with Transformers (ACT). Our experimental results show that BC performed the worst, only being able to locate the object but failing to grasp it successfully. GAIL showed better performance, occasionally managing to grasp the object, while ACT demonstrated more consistent success, reliably grasping the object. However, ACT was unable to successfully place the object into the designated container. This study provides valuable experimental insights for the selection of imitation learning methods in robotic arm tasks.

1 INTRODUCTION

With the advancement of robotics, imitation learning has become a critical approach for training robotic arms in various tasks by mimicking expert human behavior. The Pick and Place task, a common manipulation task, is widely used to assess and compare the performance of different imitation learning algorithms. While progress has been made in simulation environments, real-world applications require robotic arms to handle more complex environments with higher precision, making the selection and optimization of imitation learning algorithms a crucial challenge.

This research compares three imitation learning algorithms: Behavioral Cloning (BC), Generative Adversarial Imitation Learning (GAIL), and Action Chunking with Transformers (ACT), in both simulation and real-world environments. We found that BC performed the poorest, managing to locate the object but unable to successfully grasp it. GAIL performed slightly better, occasionally succeeding in grasping the object. In contrast, ACT demonstrated more stable performance, consistently grasping the object. However, ACT was not able to successfully place the object into the designated container. Our findings provide valuable insights for selecting the most suitable imitation learning algorithm for robotic arm tasks and their real-world applications.

2 RELATED WORKS

In recent years, embodied intelligence and imitation learning (IL) for robotic arms have emerged as critical areas of research within the broader field of robotics. Imitation learning enables robots to acquire task strategies by mimicking expert demonstrations, offering an efficient alternative to traditional reinforcement learning (RL), especially for tasks where direct exploration is difficult or costly. Below, we provide an overview of significant advances in these domains.

2.1 FUNDAMENTAL METHODS OF IMITATION LEARNING

The foundational approach to imitation learning is **Behavior Cloning (BC)**, where robots learn tasks by supervised learning from expert-provided state-action pairs. Pomerleau (1988) introduced **ALVINN**, a system for autonomous driving that applied BC to learn navigation strategies. While BC proved effective in simpler environments, its performance struggled with high-dimensional sensory inputs, such as images. To overcome this limitation, Hester et al. (2017) improved BC for robotic control tasks by integrating deep learning techniques, allowing the system to process complex visual inputs and demonstrate better generalization in real-world environments.

2.2 Generative Adversarial Imitation Learning

To further enhance imitation learning, Ho & Ermon (2016) proposed **Generative Adversarial Imitation Learning (GAIL)**, which combines ideas from Generative Adversarial Networks (GANs). In GAIL, a generator imitates the expert's behavior while a discriminator distinguishes between the expert's actions and the robot's actions. This adversarial setup enables the robot to effectively learn policies without explicit reward signals, making GAIL particularly powerful in reinforcement learning tasks and robotic control. GAIL's ability to generalize across various tasks has made it a popular approach in modern imitation learning research.

2.3 Combining Reinforcement Learning and Imitation Learning

To address the limitations of traditional imitation learning, which struggles in data-scarce or complex environments, researchers have focused on combining **Reinforcement Learning (RL)** with imitation learning. Duan et al. (2017) introduced **One-Shot Imitation Learning**, which leverages RL to complement imitation learning, allowing robots to adapt to tasks with only limited expert demonstrations. In a similar vein, Rajeswaran et al. (2018) developed **On-Policy Imitation Learning**, combining RL-based policy optimization with imitation learning to improve performance in dynamic and unpredictable environments.

2.4 Multimodal Perception and Imitation Learning

The integration of multimodal perception—particularly combining vision and tactile feedback—has gained considerable attention in recent years. By incorporating multiple sensory modalities, robots can perform tasks with greater precision and robustness. Finn et al. (2016) introduced **Deep Visual Foresight**, a framework that predicts future states from visual inputs and assists robots in planning actions. Additionally, Yu et al. (2018) demonstrated how combining vision with force sensor data enhances imitation learning, enabling robots to perform more sophisticated in-hand manipulation tasks. Multimodal approaches equip robots with richer sensory input, improving their ability to deal with complex and uncertain environments.

2.5 Online Imitation Learning and Scalability

The demand for real-time learning in dynamic environments has led to growing interest in **online imitation learning**. This approach allows robots to continuously improve their strategies based on real-time feedback and demonstrations. Kormushev et al. (2011) proposed gesture-based learning, which extended imitation learning techniques to real-time tasks, enabling robots to adapt on the fly. More recently, Rajeswaran et al. (2018) introduced **Learning to Reinforcement Learn**, a framework that combines imitation learning with RL-based online updates, facilitating continuous performance improvement in complex environments.

2.6 Combining Deep Learning and Imitation Learning

Deep learning has significantly advanced imitation learning, particularly in handling high-dimensional sensory inputs such as images, videos, and sensor data. Bohg et al. (2014) utilized convolutional neural networks (CNNs) to process image data for robotic grasping tasks, employing imitation learning to derive effective manipulation strategies. Furthermore, Rusu et al. (2016) introduced **Policy Distillation**, which combines deep learning with imitation learning by transferring knowledge from complex models to more lightweight models, thereby enhancing computational efficiency and scalability.

2.7 ACT and Diffusion Policy Approaches

Recent advances in policy learning have introduced novel frameworks like **Action Chunking with Transformers (ACT)** and **Diffusion Policies**, which aim to improve policy learning in complex and high-dimensional environments. Zhao et al. (2023) introduced the ACT, a technique that addresses compounding errors in imitation learning by breaking action predictions into smaller chunks and integrating temporal information to stabilize long-term predictions. In particular, **Diffusion Policies**—which are grounded in continuous action space planning—have shown promise in scenarios where traditional RL methods struggle to converge or generalize. By modeling action distributions over continuous spaces, diffusion-based models provide more robust solutions to high-dimensional control problems. These advancements open up new opportunities for embodied intelligence and imitation learning, enabling robots to achieve more nuanced and generalizable task execution.

2.8 Challenges and Future Directions in Embodied Intelligence

Despite significant progress, several challenges remain in practical applications of imitation learning. One key issue is the ability to learn effectively in the absence of explicit reward signals. Additionally, improving robots' generalization abilities, particularly in unknown environments or on novel tasks, remains a crucial research challenge. Future directions in embodied intelligence may include exploring self-supervised learning techniques, as well as leveraging large-scale, unlabeled datasets to further enhance the capabilities of imitation learning models.

3 Methods

3.1 Action Chunking with Transformers (ACT)

3.1.1 Original Implementation of ACT

The CVAE encoder and decoder with transformers are implemented, as transformers are designed for both synthesizing information across a sequence and generating new sequences.

CVAE encoder: The CVAE encoder is implemented with a BERT-like transformer encoder Devlin et al. (2019). The inputs to the encoder are the current joint positions and the target action sequence of length k from the demonstration dataset, prepended by a learned "[CLS]" token similar to BERT. This forms a $k + 2$ length input (Figure 1 left).

After passing through the transformer, the feature corresponding to "[CLS]" is used to predict the mean and variance of the "style variable" z, which is then used as input to the decoder.

CVAE decoder: The CVAE decoder (i.e. the policy) takes the current observations and z as the input, and predicts the next k actions (Figure 1 right). We use ResNet image encoders, a transformer encoder, and a transformer decoder to implement the CVAE decoder. The observation includes 4 RGB images, each at 480×640 resolution, and joint positions for two robot arms (7+7=14 DoF in total). The action space is the absolute joint positions for two robots, a 14-dimensional vector.

With action chunking, the policy outputs a $k \times 14$ tensor given the current observation.

The policy first process the images with ResNet18 backbones He et al. (2015), which convert $480 \times 640 \times 3$ RGB images into $15 \times 20 \times 512$ feature maps. We then flatten along the spatial dimension to obtain a sequence of 300×512. To preserve the spatial information, we add a 2D sinusoidal

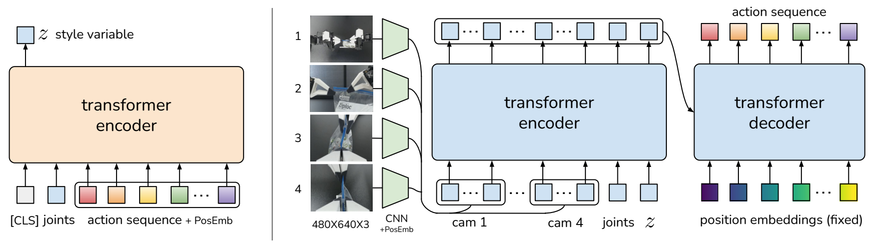

Figure 1: Architecture of Action Chunking with Transformers (ACT).

position embedding to the feature sequence Carion et al. (2020). Repeating this for all 4 images gives a feature sequence of 1200×512 in dimension. We then append two more features: the current joint positions and the "style variable" z. They are projected from their original dimensions to 512 through linear layers respectively. Thus, the input to the transformer encoder is 1202×512. The transformer decoder conditions on the encoder output through cross-attention, where the input sequence is a fixed position embedding, with dimensions $k \times 512$, and the keys and values are coming from the encoder. This gives the transformer decoder an output dimension of $k \times 512$, which is then down-projected with an MLP into $k \times 14$, corresponding to the predicted target joint positions for the next k steps.

3.1.2 Our Adjustment of ACT

We mainly changed the observation and the action space.

The changed observation includes 1 RGB image at 240×426 resolution, and Cartesian position concatenated with gripper state of the robot arm (6 + 1 = 7 Dim in total). The action space is the Cartesian position concatenated with gripper state of the robot arm, a 7-dimensional vector. Additionally, we skipped the encoder setup and directly set z to 0 for test.

And with action chunking, the policy outputs a $k \times 7$ tensor given the current observation.

3.1.3 Training and Reference Algorithms

We summarize the training and inference of ACT in Algorithms 1 and 2. The model has around 80M parameters. The training takes around 20 Minutes on a single 6G RTX 3060 Laptop GPU.

3.1.4 Principle of ACT: Action Chunking and Temporary Ensemble

A major shortcoming of BC is compounding errors, where errors from previous timesteps accumulate and cause the robot to drift off of its training distribution, leading to hard-to-recover states Ross et al. (2010); Tu et al. (2021).

To combat the compounding errors of imitation learning in a way that is compatible with pixel-to-action policies, ACT predicts a chunk of actions.

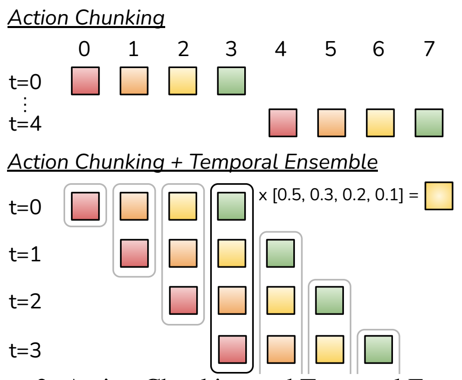

Figure 2: Action Chunking and Temporal Ensembling

In the implementation of ACT, the chunk size is fixed to be k: every k steps, the agent receives an observation, generates the next k actions, and executes the actions in sequence (Figure 2). This implies a k-fold reduction in the effective horizon of the task. Concretely, the policy models $\pi_\theta(a_{t:t+k}|s_t)$ instead of $\pi_\theta(a_t|s_t)$.

A naïve implementation of action chunking can be sub-optimal: a new environment observation is incorporated abruptly every k steps. To improve smoothness and avoid discrete switching between executing and observing, ACT query the policy at every timestep. This makes different action chunks overlap with each other, and at a given timestep there will be more than one predicted action. This process is illustrated in Figure 2 and ACT proposes a *temporal ensemble* to combine these predictions. The *temporal ensemble* performs a weighted average over these predictions with an exponential weighting scheme $w_i = \exp(-m * i)$, where w_0 is the weight for the oldest action.

The speed for incorporating new observation is governed by m, where a smaller m means faster incorporation.

Algorithm 1 ACT Training

```
1: Given: Demo dataset $\mathcal{D}$, chunk size $k$, weight $\beta$.
2: Let $a_t$, $o_t$ represent action and observation at timestep $t$, $\bar{o}_t$ represent $o_t$ without image observations.
3: Initialize encoder $q_\phi(z|a_{t:t+k}, \bar{o}_t)$
4: Initialize decoder $\pi_\theta(\hat{a}_{t:t+k}|o_t, z)$
5: for iteration $n = 1, 2, ...$ do
6:     Sample $o_t$, $a_{t:t+k}$ from $\mathcal{D}$
7:     Sample $z$ from $q_\phi(z|a_{t:t+k}, \bar{o}_t)$
8:     Predict $\hat{a}_{t:t+k}$ from $\pi_\theta(\hat{a}_{t:t+k}|o_t, z)$
9:     $\mathcal{L}_{reconst} = MSE(\hat{a}_{t:t+k}, a_{t:t+k})$
10:    $\mathcal{L}_{reg} = D_{KL}(q_\phi(z|a_{t:t+k}, \bar{o}_t) \,||\, \mathcal{N}(0, I))$
11:    Update $\theta$, $\phi$ with ADAM and $\mathcal{L} = \mathcal{L}_{reconst} + \beta\mathcal{L}_{reg}$
```

Algorithm 2 ACT Inference

```
1: Given: trained $\pi_\theta$, episode length $T$, weight $m$.
2: Initialize FIFO buffers $\mathcal{B}[0 : T]$, where $\mathcal{B}[t]$ stores actions predicted for timestep $t$.
3: for timestep $t = 1, 2, ...T$ do
4:     Predict $\hat{a}_{t:t+k}$ with $\pi_\theta(\hat{a}_{t:t+k}|o_t, z)$ where $z = 0$
5:     Add $\hat{a}_{t:t+k}$ to buffers $\mathcal{B}[t : t + k]$ respectively
6:     Obtain current step actions $A_t = \mathcal{B}[t]$
7:     Apply $a_t = \sum_i w_i A_t[i] / \sum_i w_i$, with $w_i = \exp(-m * i)$
```

3.2 Behavior Cloning

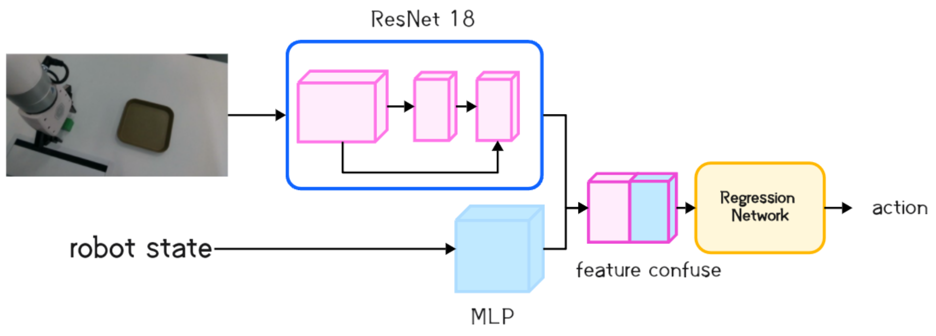

Figure 3: Structure of BC network

In this section, we outline our approach for applying **Behavior Cloning (BC)** to the **Pick-and-Place** task using a robotic arm. The goal of this task is for the robot to autonomously pick up an object from a specified location and place it at a target position based on visual input and robotic state information.

3.2.1 Model Architecture

For the **Pick-and-Place** task, we use a deep neural network as the behavior cloning model. The model takes as input both **camera images** and the **robotic state** and outputs the action required for the robot to perform the task. The architecture is composed of two main components:

- **Image Encoder**: The camera image ($224 \times 224 \times 3$) is passed through a convolutional neural network (CNN) to extract spatial features. The CNN consists of several convolutional layers followed by pooling layers to capture the visual information about the scene. The encoder outputs a feature vector representing the visual state of the environment.

- **State Encoder**: The robot's state (7-dimensional vector) is passed through a fully connected network (FCN) that processes the robot's internal state, such as joint angles, velocities, and other relevant features. This encoding captures the robot's configuration and its understanding of the environment.
- **Fusion and Action Output**: The features from the image encoder and state encoder are concatenated and passed through a fully connected layer to produce the final action. The action could be in the form of a continuous vector representing the position and orientation of the gripper or joint velocities, depending on the task's requirements.

3.2.2 Training Process

The training of the behavior cloning model follows a supervised learning approach, where the network learns to replicate the actions taken by an expert during the task. We use expert demonstrations in the form of **state-action pairs** collected from a previous run of the robotic arm in the environment.

- **Data Collection**: Expert demonstrations are collected using a pre-trained robotic controller or through human demonstrations. Each sample consists of a pair $(\text{camera\_image}, \text{state})$ and the corresponding action taken by the expert. The action is usually represented as a vector of joint velocities or directly as the gripper's position and orientation.
- **Preprocessing**: The images are normalized and resized to fit the model's input dimensions. The robotic state is also normalized to a consistent range. For each demonstration, the sequence of states and corresponding actions are aligned to maintain the temporal continuity of the task.
- **Model Training**: The model is trained to minimize the discrepancy between the predicted actions and the expert's actions. During training, we utilize **stochastic gradient descent (SGD)** with **mini-batches** of data to update the model's parameters. The model is trained for a fixed number of epochs or until convergence is achieved.

3.2.3 Loss Function

The core objective of behavior cloning is to minimize the difference between the predicted actions and the ground-truth actions from the expert. The most commonly used loss function for this task is the **mean squared error (MSE)** loss:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|a_i - \hat{a}_i\|^2$$

where:

- N is the number of training samples.
- a_i is the expert's action at sample i.
- $\hat{a}_i$ is the predicted action by the model at sample i.

This loss function ensures that the predicted actions converge to those of the expert. In some cases, if the task involves discrete actions or classification-based actions (e.g., gripper state: open or closed), **cross-entropy loss** may be employed.

3.2.4 Regularization and Constraints

Given the nature of the task, it's essential to avoid overfitting, especially since the expert demonstration dataset might not be large. We apply the following regularization techniques to improve generalization:

- **Weight Decay**: L2 regularization is applied to the weights of the neural network to prevent large weights, which could lead to overfitting.

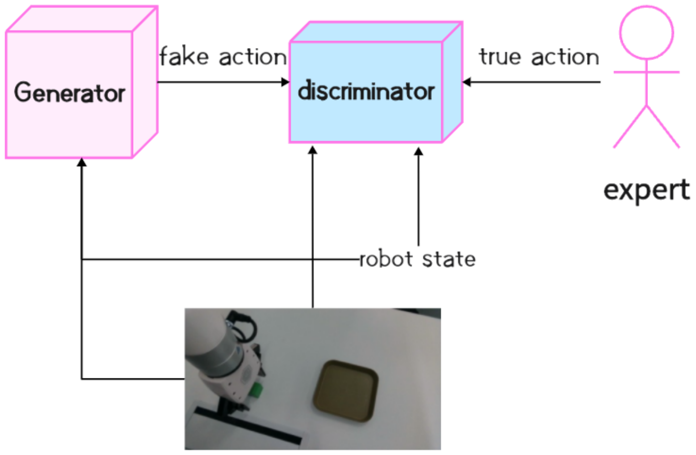

Figure 4: Structure of Gail network

- **Data Augmentation**: To enhance the robustness of the model, we apply several data augmentation techniques to the images, including random cropping, rotation, and scaling. For the robotic state, we apply noise to simulate sensor inaccuracies.
- **Action Smoothing**: To make the predicted actions smoother and more stable during inference, we apply a **smoothing filter** over the predicted actions. This helps mitigate jittering movements that could result from the model's predictions.

3.3 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) is a method that uses adversarial training to learn a policy by imitating expert behavior. It involves two main components: the *generator* (policy network) and the *discriminator*.Ho & Ermon (2016)

- **Generator (Policy Network)**: The generator is a policy network that generates actions (robot control commands) based on the current observations (image and state information). The goal of the generator is to mimic the expert's behavior as closely as possible.
- **Discriminator**: The discriminator is a binary classifier that evaluates whether a given trajectory (state-action pair) comes from the expert dataset or the generator. Its goal is to distinguish between expert trajectories and generated trajectories.

3.3.1 Model Architecture

Generator Model (Policy Network)

The generator is responsible for generating control actions for the robotic arm based on current state and image inputs. The inputs to the generator include both the image data (from the camera) and the state information (e.g., joint angles, velocities), and the output is the control action (e.g., joint angle commands).

To improve the expressive power of the generator, we use a convolutional neural network (CNN) to process image data, while fully connected layers are used to process the state information of the robotic arm.

- **Input**: At each timestep, the input consists of the state vector (including joint angles and velocities) and the corresponding image (224x224x3).
- **Output**: The output is the control action for the robotic arm (the change in joint angles or control commands).

The generator model architecture consists of:

- A CNN that extracts features from the input images.
- A fully connected layer that fuses the image features with the robotic arm's state information.
- The output layer produces the control action for the robotic arm.

Discriminator Model

The discriminator's task is to classify whether a given trajectory (state-action pair) comes from the expert data or from the generator. The input to the discriminator is a tuple consisting of the image, state information, and action, and the output is a binary classification probability indicating whether the trajectory is from the expert (label = 1) or generated (label = 0).

The discriminator model architecture is as follows:

- The image information is processed by convolutional layers.
- The state information and action are combined through fully connected layers, and the final output is a binary classification result.

The discriminator's goal is to maximize its classification accuracy, thereby encouraging the generator to produce trajectories that closely resemble the expert's.

3.3.2 Training Process

Alternating Training of Generator and Discriminator

During training, the generator and discriminator are trained alternately:

- **Training the Discriminator**: Given both expert and generated data, the discriminator is updated to distinguish between these two types of trajectories. The discriminator is trained to maximize the adversarial loss, outputting 1 for expert trajectories and 0 for generated ones.
- **Training the Generator**: With the discriminator's parameters fixed, the generator is updated to produce trajectories that are indistinguishable from expert trajectories. The generator's loss is minimized by making the discriminator classify generated trajectories as expert-like (label = 1).

Loss Functions

The GAIL loss function consists of two components: the *discriminator loss* and the *generator loss*.

- **Discriminator Loss**: The discriminator's goal is to distinguish between expert and generated trajectories. The loss function is typically the binary cross-entropy loss:

  $$L_D = -\mathbb{E}_{(s,a)\sim\mathcal{D}}[\log(D(s,a))] - \mathbb{E}_{(s,a)\sim\mathcal{G}}[\log(1 - D(s,a))]$$

  where $\mathcal{D}$ is the expert data, $\mathcal{G}$ is the generated data, and $D(s,a)$ is the discriminator's output probability.
- **Generator Loss**: The generator's goal is to make the discriminator unable to distinguish between generated and expert trajectories. Therefore, the generator's loss function is:

  $$L_G = -\mathbb{E}_{(s,a)\sim\mathcal{G}}[\log(D(s,a))]$$

  The generator is trained to maximize the probability assigned by the discriminator to the generated trajectories.

Training Procedure

The training procedure consists of the following steps:

1. **Initialization**: Initialize the parameters of both the generator and discriminator networks.

2. **Discriminator Training**: Sample from the expert and generated data, and update the discriminator's parameters.
3. **Generator Training**: With the discriminator fixed, update the generator's parameters to produce more expert-like trajectories.
4. **Alternating Optimization**: Alternate between training the discriminator and generator until convergence.

3.3.3 Loss Function Selection and Optimization

The choice of loss function is critical for the success of GAIL. We use the standard binary cross-entropy loss for training the discriminator, ensuring it can effectively distinguish between expert and generated trajectories. For the generator, the loss function encourages the generator to produce trajectories that are indistinguishable from expert data.

To ensure stable training, the learning rates for the generator and discriminator need to be carefully tuned to avoid issues such as vanishing or exploding gradients. Techniques such as experience replay or target networks may be used to further stabilize training and reduce variance.

4 Experiment & Analysis

4.1 ACT

4.1.1 Experiment Process

We selected the "grab cube" dataset, where the robotic arm moves from the starting position, grabs the cube, and places it on the plate, for this training.

Initially, we used the angles of the robotic arm's seven joints as the training target, running 4000 steps with the chunk size of 100. However, the arm performed poorly and failed to move correctly.

To improve, we switched to using the Cartesian position of the robotic arm concatenated with the gripper state as the training target, maintaining 4000 steps and a chunk size of 100. This adjustment allowed the arm to approach the cube, but it still struggled to grasp it, as the gripper could only move near the cube without successfully picking it up. We then extended the training steps to 8000 steps, but the robotic arm's performance remained unsatisfactory.

After reviewing the training data used in the ACT source code, we discovered that each training episode contained over 1000 frames, whereas our data had at most 400 frames. Upon closely studying the paper, we concluded that the issue might be related to the chunk size.Since our data had an average of about 200 frames per episode, predicting the next 100 steps at once could lead to instability in the robotic arm's movements.

To address this, we reduced the chunk size to 20 and maintained 4000 steps, which made the arm's movements more stable. However, it would still get stuck while moving toward the cube.

Later, while reviewing the repository's README file, I found the following tip: "ACT tuning tips: if your ACT policy is jerky or pauses in the middle of an episode, just train for longer! Success rate and smoothness can improve way after loss plateaus."

Based on this, we increased the training steps to 8000 and trained models with chunk sizes of 10, 20, and 35. We found that the models with chunk sizes of 10 and 20 would still get stuck while moving towards the cube. Only the model with a chunk size of 35 could successfully pick up the cube, but it would also get stuck when moving the cube onto the plate.

We then trained the model with chunk sizes 30 and 40 for 4000 steps, neither of them has a better performance.

4.1.2 Results

The best performance of the robotic arm is to move from the starting position to the cube and grab it, trained with 8000 steps and a chunk size of 35. The following is the visualization of the loss data during the training of this model.

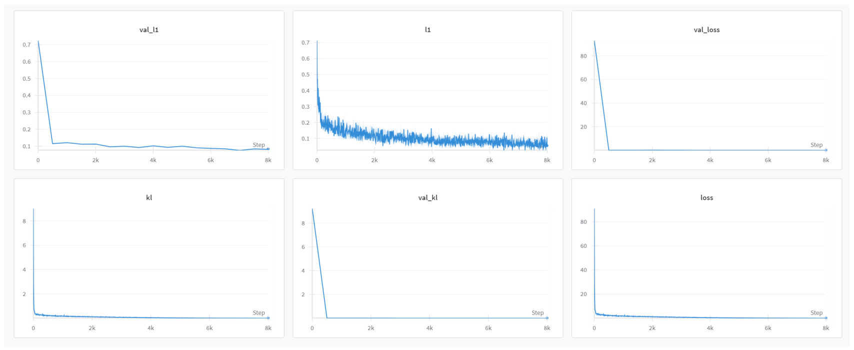

Figure 5: ACT Training Loss: 8000 steps, chunk size 35

4.2 GAIL

4.2.1 Experimental Setup

In our experiments, we applied GAIL to the Pick and Place task using a robotic arm. We trained the model in both a simulated environment (robosuite) and a real-world environment (Ufactory X-arm 7). The dataset for GAIL consisted of expert demonstrations, including both camera images and the robot's state information at each time step. The GAIL algorithm was trained using the following setup:

- **Generator (Policy Network):** The policy network generates actions based on the current state and image inputs, aiming to mimic the expert's behavior.
- **Discriminator:** The discriminator learns to distinguish between the expert's behavior and the behavior generated by the policy network.
- **Reward Function:** The discriminator's output serves as a reward signal for updating the policy network, encouraging it to produce more expert-like behavior.
- **Training:** The agent was trained using the Adam optimizer, with learning rates tuned for both the generator and discriminator networks. The training process was carried out for 500,000 timesteps, with performance evaluations occurring every 50,000 timesteps.

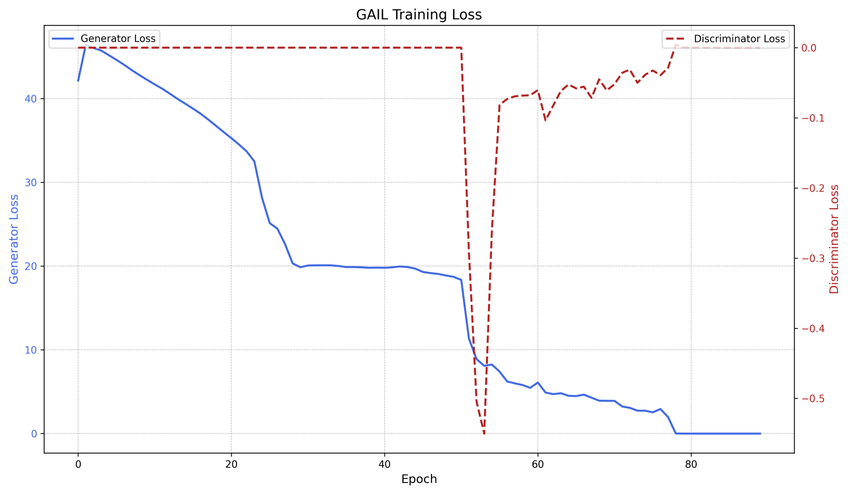

Figure 6: GAIL Training Loss

4.2.2 Results

The performance of GAIL was evaluated based on task completion and success rates in both simulation and real-world environments. The following observations were made:

- **Simulation Environment:** In the simulation environment (robosuite), GAIL achieved a success rate of around 50%. After training for 500,000 timesteps, the agent was able to grasp the object approximately half of the time. The policy learned to adjust its gripper and approach the object, but failures still occurred due to minor misalignments or incorrect grasping strategies.
- **Real-World Environment:** In the real-world setting (Ufactory X-arm 7), GAIL was unable to successfully complete the task. The added complexity of real-world noise, sensor inaccuracies, and environmental variability hindered the agent's ability to grasp the object reliably. However, it still showed some improvement over the baseline BC model, as the agent was able to occasionally attempt grasping.
- **Comparison with BC and ACT:** GAIL outperformed BC, which failed to complete the task, and was more robust than ACT in some aspects. However, GAIL's performance was less stable compared to ACT, which demonstrated a more consistent ability to complete the task in the simulation, even though ACT still faced challenges in placing the object in the designated container.

5 Conclusion & Reflection

In this project, we conducted an in-depth study of imitation learning algorithms for robotic arms and carried out multiple experiments in both a simulation environment (robosuite) and a real-world environment (Ufactory X-arm 7). Throughout the process, we not only learned how to set up and operate the robosuite simulation environment, but also gained hands-on experience interacting with the real robot using the Ufactory X-arm 7 SDK, which provided us with valuable insights into working with actual hardware.

We learned and applied several imitation learning algorithms, including IL (Imitation Learning), GAIL (Generative Adversarial Imitation Learning), and ACT (Action Chunking Transformer). Our experimental results showed that BC (Behavioral Cloning) performed the worst, failing to effectively complete the task; GAIL showed improvements but was still unstable in the real-world environment, occasionally able to grasp the object but with a low task completion rate. ACT, while able to consistently grasp the object, was unable to successfully place the object into the target container, exposing its limitations in handling the multi-step decision-making process required for such tasks.

Through these experiments, we recognized the limitations of imitation learning, particularly with single-step prediction. Single-step prediction fails to capture the multi-step relationships in tasks, causing the robotic arm to miss the correct execution flow from grasping to placing. This realization led us to understand that effective strategies require multi-step prediction, where the model takes into account the long-term effects of each action, rather than relying solely on immediate feedback. This insight prompted us to introduce time-dependent learning strategies in later experiments, combining reinforcement learning with imitation learning to improve performance on complex tasks.

Additionally, we found that deterministic models did not perform as expected, especially in real-world environments where stability was impacted by environmental noise and sensor inaccuracies. We discovered that using models with probabilistic output, which better simulate human behavior and deal with uncertainties, significantly improved the robustness of the model. This approach helped the robotic arm perform more consistently in tasks by making it more adaptable to dynamic and uncertain environments.

In tackling these issues, we optimized the training process and improved the diversity and quality of expert data to enhance the model's generalization capabilities. However, even with improvements such as multi-step prediction and probabilistic output, ACT still struggled to complete the full task from grasping to placing the object. This highlighted the importance of multi-step decision-making for complex tasks. Future work will focus on refining the multi-step prediction mechanism to enhance the model's decision-making abilities, especially in scenarios where each step of the task requires precise execution.

Overall, this project has significantly enhanced our understanding and technical ability in robotic arm control and imitation learning, while also revealing the limitations of current approaches. In the future, we will continue to explore hybrid strategies and multi-step learning algorithms to improve performance in complex and dynamic environments, particularly in tasks that require a complete execution sequence.

REFERENCES

Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, April 2014. ISSN 1941-0468. doi: 10.1109/tro.2013.2289018. URL `http://dx.doi.org/10.1109/TRO.2013.2289018`.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *ArXiv*, abs/2005.12872, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning, 2017. URL `https://arxiv.org/abs/1703.07326`.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization, 2016. URL `https://arxiv.org/abs/1603.00448`.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations, 2017. URL `https://arxiv.org/abs/1704.03732`.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning, 2016. URL `https://arxiv.org/abs/1606.03476`.

Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *Advanced Robotics*, 25(5):581–603, 2011. URL `https://kormushev.com/papers/Kormushev_AdvancedRobotics_2011.pdf`.

Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky (ed.), *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL `https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf`.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations, 2018. URL `https://arxiv.org/abs/1709.10087`.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation, 2016. URL `https://arxiv.org/abs/1511.06295`.

Stephen Tu, Alexander Robey, Tingnan Zhang, and N. Matni. On the sample complexity of stability constrained imitation learning. In *Conference on Learning for Dynamics & Control*, 2021.

Wenhao Yu, Greg Turk, and C. Karen Liu. Learning symmetric and low-energy locomotion. *ACM Transactions on Graphics*, 37(4):1–12, July 2018. ISSN 1557-7368. doi: 10.1145/3197517.3201397. URL `http://dx.doi.org/10.1145/3197517.3201397`.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. URL `https://arxiv.org/abs/2304.13705`.