Evaluating Generalization in Robotic Pick-and-Place from Limited Demonstrations

Franklin Wu, Teresa Nguyen
Final Paper for CPSC 381: Introduction to Machine Learning

***Abstract*—In this project, we explored how imitation learning can be used to train a robotic arm to perform a pick-and-place task using a limited number of demonstrations. We collected our own dataset through a teleoperation setup, where a leader robot controlled a follower robot to generate 50 demonstration trials. Using this data, we trained and evaluated two different models: the Action Chunking with Transformers (ACT) model and our own behavior cloning (BC) model. The ACT model achieved a 70% pick success rate and a 60% full task success rate, demonstrating that it was able to learn the task and generalize reasonably well to new conditions. On the other hand, while the BC model showed decreasing training loss and was able to follow general trends in the data, it failed to successfully complete the task in real-world execution, achieving a 0% success rate. We learned that models like ACT, which predict sequences of actions, are more robust to small errors and better suited for real-world deployment, while models employing standard behavior cloning struggles due to error accumulation over time. Our project shows both the potential and challenges of utilizing imitation learning on robotic systems using limited data.**

I. Introduction

Robotic manipulation is a major challenge in robotics and machine learning because it requires systems to perceive their environment, reason about objects, and execute precise physical actions [7]. Tasks that appear simple to humans, such as picking up an object and placing it into a container, require a robot to interpret visual information, estimate object position, and control a multi-joint arm in real time. Recent advances in machine learning have made it possible to train robotic policies directly from data rather than relying on hand-engineered pipelines. In particular, imitation learning shown in Figure 1 has shown promise in enabling machines to learn manipulation behaviors by observing demonstrations or learning from a relatively small amount of data [2].

In this project, our goal was to explore how imitation learning [3], shown in Figure 1 can be used to train a robotic arm to perform pick-and-place tasks using human demonstrations and to evaluate whether the learned model can generalize to new conditions compared to existing policies. We collected demonstrations through a teleoperation setup and used this data to train models that predict robot actions from visual observations and robot state information. This problem is a supervised learning task, where the model learns a mapping from inputs to outputs using training data and optimization techniques such as gradient descent. We built our own behavior cloning (BC) model, which is a supervised imitation learning method where a model learns to directly imitate the actions from demonstrations [5] and compared it to an action chunking with transformers (ACT) model as a baseline.

There were several challenges throughout the project, particularly in hardware setup, data collection, and training. We had to manually adjust some of the 3D-printed parts to ensure the robot joints moved correctly. Data collection was also challenging: we initially used one camera, but during early trials, the robot struggled to perform the pick-and-place task. To address this, we reconfigured the setup by adding a point-of-view camera to provide better visual coverage and improve generalization. Training was another challenge, as running experiments on a MacBook was too slow and would have taken up to a week. To overcome this, we used a more powerful personal computer, which reduced training time to around 10 hours per run. Overall, our goal was to build our own policy using concepts learned in class and compare it to existing methods, while navigating the practical challenges of working with real-world robotic systems.

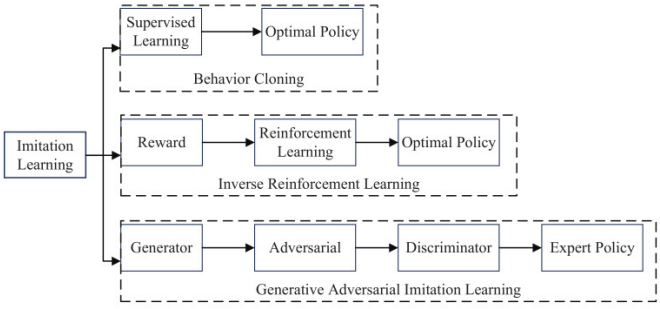

Fig. 1. Classification of Imitation Learning from Hua et. al [2]

II. Data Collection

For this project, we collected our own dataset using a real-world teleoperation setup rather than relying on a pre-existing public dataset. Initially, our data collection procedure involved a pen and a pen holder, where the goal was for the robot to pick up the pen and place it inside the holder. We collected 50 trials using this setup. For this initial setup, we used a single overhead camera, and the leader robot was positioned directly next to the follower robot.

After training our first ACT model and evaluating its performance, we observed that the robot was unable to successfully complete the task and would often fail or crash. Upon review of our data, we realized that the leader robot frequently covered the camera's view of the pen, making it difficult for

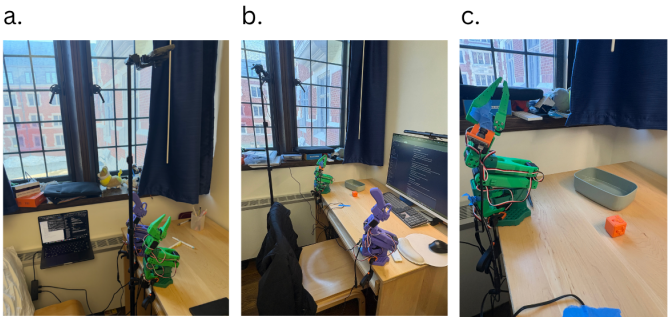

Fig. 2. (a) is the old setup (b) new setup (c) close up of the new setup with the pick and place goal

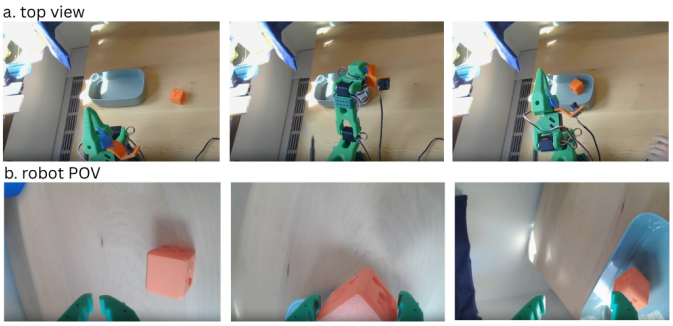

Fig. 3. (a) shows the top view, with the first image being recognizing the block, second being grabbing, and the third being placing the block into the bowl. (b) shows the same but through the robot's point of view.

the model to learn visual features. Additionally, the pen task itself required precise alignment and was too difficult given the limited dataset, which made generalization challenging. To address these issues, we redesigned the task and environment. We switched to a simpler setup by using a small 1-inch 3D printed cube and a bowl (in contrast to our pen and cup setup), where the new objective was to pick up the cube and place it into the bowl. Our old and new setup is shown in Figure 2.

For the final dataset, we used the LeRobot recording pipeline to collect 50 demonstration episodes. Each episode consisted of a full pick-and-place trajectory performed through teleoperation, where the leader robot controlled the follower arm. We introduced variation across episodes by changing the position of the cube and bowl, as well as adjusting lighting conditions to improve generalization. The data was stored in a Hugging Face repository using the LeRobotDataset format, which includes synchronized multimodal data such as robot actions, joint states, and camera observations. In this final setup, we used two camera views: an overhead (top) view capturing the entire workspace and a point-of-view (wrist/mouth) camera, installed directly on the robot, which provided a closer perspective of the gripper and object interaction, shown with an example of our experiment in both camera angles in Figure 3.

To ensure data quality, we manually reviewed 20 random trials during collection and discarded any demonstrations where the task was not completed, re-recording trials as needed. Additionally, we randomly sampled 10 collected episodes and replayed them on the robot to verify that the recorded trajectories were smooth and executed the task physically. These checks helped to ensure that the dataset was consistent and suitable for training.

For evaluation, we used a separate set of trials in which the trained policy was deployed on the robot without human control. To test generalization, we varied conditions such as object position and workspace configuration, and measured how many trials the robot successfully completed out of a fixed number of attempts. This allowed us to assess how well the learned policy could handle variations beyond the training distribution.

III. Methodology

A. Dataset Preparation

For our experiments, we utilized the SO101 robots [9] since it had extensive documentation, easy access to parts, and the ability to build upon them. We assembled the robots using 3D printed parts and calibrated all of the servos for the joints to ensure proper movement. After calibration, we verified the camera setup by teleoperating the robot to make sure everything was functioning correctly.

B. Action Chunking Transformer

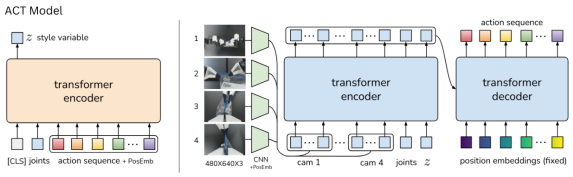

Fig. 4. ACT Model Architecture from Zhao et. Al [11]

We then ran the baseline ACT model [11], which was provided through the LeRobot framework. ACT stands for Action Chunking with Transformers and is a lightweight and efficient imitation learning model. It has around 80 million parameters, is relatively fast to train, and is data-efficient. In our setup, training ACT on 50 demonstration trials took approximately 10 hours [1].

ACT uses a ResNet-18 backbone, which is a lightweight convolutional neural network (CNN), to process images from multiple camera viewpoints. It also uses a transformer encoder to combine information from the camera features, joint positions, and a learned latent variable. A transformer decoder is then used to generate sequences of actions using cross-attention, allowing the model to predict temporally consistent control commands [11]. Its architecture is shown in Figure 4.

For the objective function, the model is trained using a combination of imitation loss and regularization. The action prediction loss is defined as the L1 loss, which minimizes the difference between the predicted actions and the expert actions:

$$L_{\text{action}} = \|a_{\text{pred}} - a_{\text{expert}}\|_1$$

where a_{pred} represents the predicted action and a_{expert} represents the ground truth action from the demonstrations. In our context, this corresponds to the mean absolute error (MAE) over the action space.

ACT also includes a KL-divergence loss, where a latent variable z is introduced and regularized toward a prior distribution:

$$L_{\text{KL}} = D_{\text{KL}}(q(z \mid x) \,\|\, p(z))$$

The total loss is then defined as:

$$L = L_{\text{action}} + \beta L_{\text{KL}}$$

where β controls the strength of the regularization.

For the optimization algorithm, we use stochastic gradient-based optimization with the Adam optimizer. Training is done using mini-batch gradient descent, and gradients are computed through backpropagation across the entire network, including the CNN, transformer encoder, and decoder (to update the model parameters).

The ACT model's weights and biases were tracked using the Weights and Biases platform [10], which allowed us to monitor and visualize our machine learning experiments and tune hyperparameters.

C. Behavior Cloning Model

After we trained the ACT model, we built our own behavior cloning (BC) model policy to explore whether a simpler and more lightweight model could achieve comparable performance. A diagram to our setup is in Figure 5. Similar to the ACT setup, we used the same 50 teleoperated demonstration episodes collected using the leader–follower system. At each timestep, we recorded a top-view RGB image, a wrist-view RGB image, the robot state $s_t \in \mathbb{R}^6$ (joint positions), and the expert action $a_t \in \mathbb{R}^6$ (target joint positions), resulting in approximately 16,857 frames.

To prevent data leakage, the dataset was split by episode rather than by frame. We used 40 episodes for training, 5 episodes for validation, and 5 episodes for testing. The robot state and action values were normalized using z-score normalization, where the mean and standard deviation were computed from the training set only.

Our behavior cloning model directly maps observations to actions without using any recurrence or action chunking. We used two separate ResNet-18 encoders to process the top and wrist camera images. Each image was resized to 224×224 and normalized using ImageNet statistics. The output of each encoder was a 512-dimensional feature vector. These feature vectors were concatenated with the normalized robot state to form a single feature vector:

$$z_t = [f_{\text{top}}; f_{\text{wrist}}; s_t]$$

This combined feature vector was passed into a multi-layer perceptron (MLP) with two hidden layers of sizes 512 and 256, using ReLU activations and a dropout rate of 0.1. The final output of the network was the predicted action $\hat{a}_t \in \mathbb{R}^6$. The total model contained approximately 23 million parameters, making it significantly smaller than the ACT model.

For the objective function, we used mean squared error (MSE) loss between the predicted and expert actions:

$$\mathcal{L} = \frac{1}{N} \sum_{t=1}^{N} \|\hat{a}_t - a_t\|_2^2$$

We also tracked mean absolute error (MAE) as an additional metric to better interpret prediction error in terms of joint angles.

For optimization, we used the Adam optimizer with a learning rate of 1×10^{-5} and a batch size of 32. Gradient clipping with a maximum norm of 10.0 was applied to improve training stability. The model was trained for up to 30,000 steps, with validation evaluated every 1,000 steps. Early stopping was used based on validation MAE, with a patience of 5 validation checks to prevent overfitting on the relatively small dataset.

During inference, the model takes in the current camera observations and robot state and outputs predicted joint positions. To reduce jitter in the predicted actions, we applied exponential moving average (EMA) smoothing:

$$\hat{a}_t^{\text{smooth}} = \alpha \hat{a}_t + (1 - \alpha)\hat{a}_{t-1}^{\text{smooth}}$$

where $\alpha = 0.5$. The model runs in real time at approximately 15 Hz during deployment.

IV. Implementation Details

A. Data Processing

We ensured that each trial had a smooth pick-and-place objective, so no major data augmentation was applied. However, the images from both cameras were resized and normalized before being passed into the network. The data was used in the LeRobot dataset format, which organizes observations and actions as time-series data. As noted earlier, we utilized two camera inputs (top and wrist views), each at a resolution of 640×480 and 10 FPS. We first used 30 FPS, but once we realized our computer could not keep up, we brought it down to 10 FPS, and the results came out reasonable. The images were processed using the ResNet-18 vision encoder within ACT, and the robot's state (joint positions) was concatenated with the visual features before being passed into the transformer.

B. Action Chunking Transformer Parameters

For training, we used the default ACT hyperparameters provided by LeRobot, and the model was trained for approximately 100,000 gradient steps using mini-batch stochastic gradient descent. We used an Adam optimizer with a learning rate on the order of 10^{-4}. The batch size was set to 8, and no explicit momentum term was used.

Additionally, as noted earlier, the loss function consisted of an L1 action prediction loss combined with a KL-divergence regularization term. During training, we monitored metrics such as training loss, L1 loss, and KL-divergence loss using

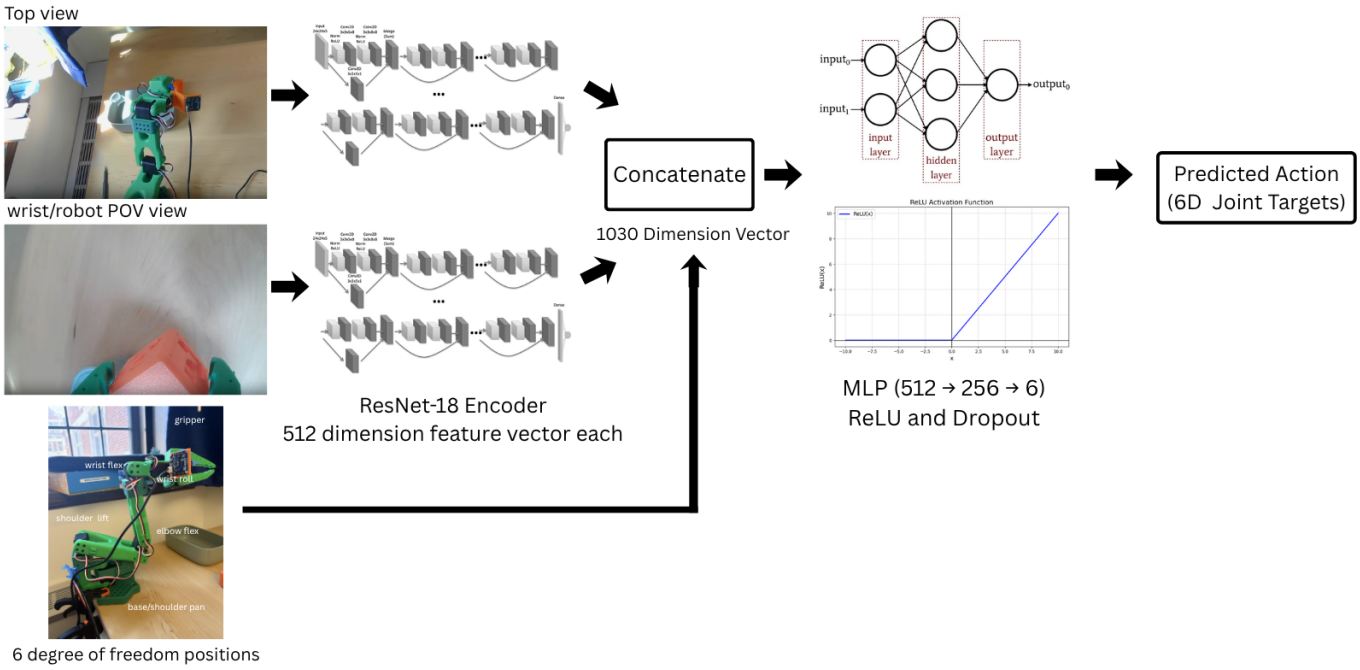

Fig. 5. Our behavior cloning model architecture. We take image inputs from both the top view and the wrist (robot POV) camera, which are passed into separate ResNet-18 encoders [4] to extract visual features. Each encoder outputs a 512-dimensional feature vector. We also include the robot state, consisting of 6 degrees of freedom (gripper, wrist flex, wrist roll, shoulder lift, elbow flex, and base shoulder pan). The two image feature vectors and the robot state are concatenated into a 1030-dimensional vector. This vector is then passed through a multi-layer perceptron (MLP) [6] with a 512 → 256 → 6 architecture, using ReLU activations [8] and dropout. The MLP outputs the predicted 6D joint actions.

the Weights and Biases website. Training was performed on a single GPU and took around 10 hours to complete for the ACT model.

C. Behavior Cloning Model Parameters

For our own model, we implemented a behavior cloning (BC) policy as a simpler baseline compared to ACT. We used the same dataset consisting of 50 teleoperated demonstration episodes, which resulted in approximately 16,857 frames. Each timestep included a top-view image, a wrist-view image, the robot state (joint positions), and the corresponding expert action.

To avoid data leakage, the dataset was split by episode rather than by frame. We used 40 episodes for training, 5 for validation, and 5 for testing. The robot state and action values were normalized using z-score normalization, where the mean and standard deviation were computed only from the training set. This ensured that no information from the validation or test sets influenced the normalization process.

Similar to ACT, we used two ResNet-18 encoders to process the top and wrist camera images. Each image was resized to 224×224 and normalized using ImageNet statistics. The outputs from both encoders were 512-dimensional feature vectors, which were concatenated with the normalized robot state to form a combined feature representation. This feature vector was then passed into a multi-layer perceptron (MLP) with hidden layer sizes of 512 and 256, using ReLU activations and a dropout rate of 0.1. The final output of the model was a 6-dimensional action vector corresponding to the robot joints.

For the objective function, we used mean squared error (MSE) loss between the predicted and expert actions. We also tracked mean absolute error (MAE) as an additional metric to better interpret prediction error in terms of joint angles.

For optimization, we used the Adam optimizer with a learning rate of 1×10^{-5} and a batch size of 32. Gradient clipping with a maximum norm of 10.0 was applied to improve training stability. The model was trained for up to 30,000 steps, with validation evaluated every 1,000 steps. Early stopping was applied based on validation MAE, with a patience of 5 validation checks to prevent overfitting on the relatively small dataset.

Training was performed on a single NVIDIA RTX 3070 GPU using PyTorch. Compared to ACT, the BC model required fewer training steps and converged faster due to its simpler smaller architecture.

V. Results

A. Action Chunking Transformer

We first plotted the training metrics. During training, we monitored the total training loss, L1 loss, and KL-divergence loss, as shown in Figure 6. These metrics measure how well the model is able to imitate the ground truth demonstrations. The primary measure of error is the L1 loss, which represents the difference between the predicted robot actions and the ground truth actions from the demonstrations. In our context, this corresponds to the mean absolute error (MAE) over the

TABLE I
EVALUATION RESULTS FOR ACT MODEL

Episode	Pick	Place	Full
0	Y	Y	Y
1	Y	Y	Y
2	Y	Y	Y
3	N	N	N
4	N	N	N
5	Y	Y	Y
6	N	N	N
7	Y	N	N
8	Y	Y	Y
9	Y	Y	Y

action space and shows how accurately the model predicts joint positions, as noted in our proposal.

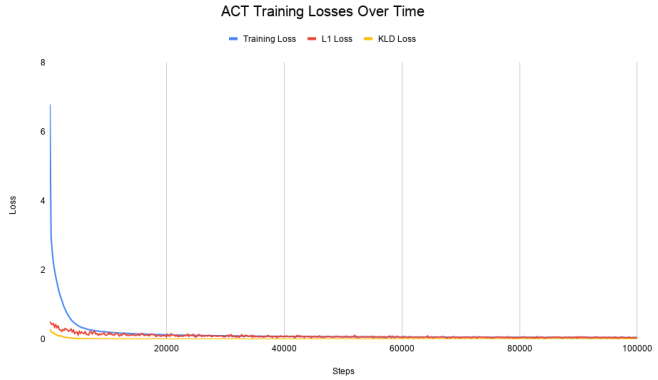

Fig. 6. ACT training losses over time. The losses decrease rapidly during early training and gradually stabilize, demonstrating convergence.

The total training loss includes both the L1 loss and the KL-divergence term, while the KL loss regularizes the latent variable used in ACT. From Figure 6, we observe that all losses decrease rapidly during the early stages of training and gradually stabilize as training progresses. This shows that the model is successfully learning to imitate the demonstrations and that training remains stable without divergence. The KL-divergence loss remains relatively small compared to the total loss, suggesting that the latent variable is properly regularized.

In addition to the training metrics, we evaluate the model based on its real-world task performance of picking up the orange cube and placing it into the bowl. As noted in our proposal, our measure of success is defined by whether the robot can successfully complete each stage of the task. We ran 10 evaluation episodes to test whether the robot could pick, place, and complete the full procedure. Our results are shown in Table I. The ACT model achieved a 70% pick success rate, 60% place success rate, and 60% full task success rate over the 10 evaluation episodes.

This shows that even with our limited demonstrations of 50, the ACT model was mostly able to accurately pick up the block, place it into the bowl, and return to its original position.

B. Behavior Cloning Model

To evaluate our behavior cloning (BC) model, we analyzed both training metrics and real-world task performance. During training, we monitored mean squared error (MSE) and mean absolute error (MAE) for both the training and validation sets. As shown in Figure 7, the training loss steadily decreased over time, indicating that the model was able to fit the training data. However, the validation loss remained relatively constant and did not decrease significantly, suggesting that the model was not generalizing well to unseen data.

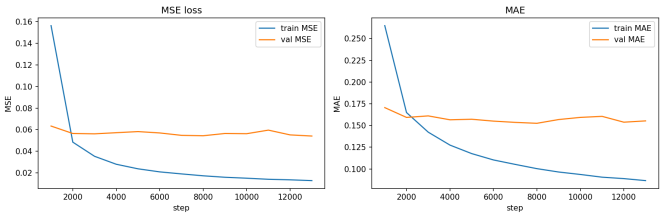

Fig. 7. Training and validation MSE and MAE over time for the BC model. While training loss decreases, validation loss remains relatively flat, indicating poor generalization.

We further analyzed the model by comparing the predicted joint actions to the expert actions. As shown in Figure 8, the predicted trajectories generally follow the overall trend of the expert actions but exhibit noticeable noise and deviation, especially in joints such as wrist roll and gripper position.

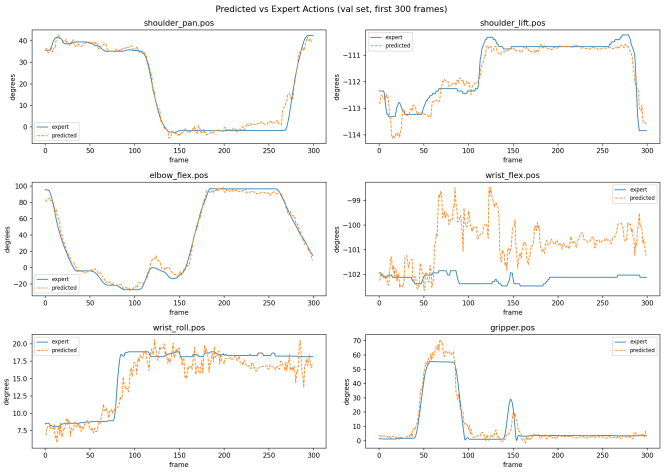

Fig. 8. Comparison of predicted and expert joint trajectories for the validation set. The BC model follows general trends but exhibits noise and deviation, particularly in more complex joints.

To better quantify this error, we plotted the absolute prediction error over time and its distribution for each joint, as shown in Figures 9 and 10. The results show that while some joints such as shoulder lift have relatively low error, others such as elbow flex, wrist roll, and the gripper have significantly higher error and large spikes. These errors indicate that the model struggles to accurately predict fine-grained joint movements required for the task.

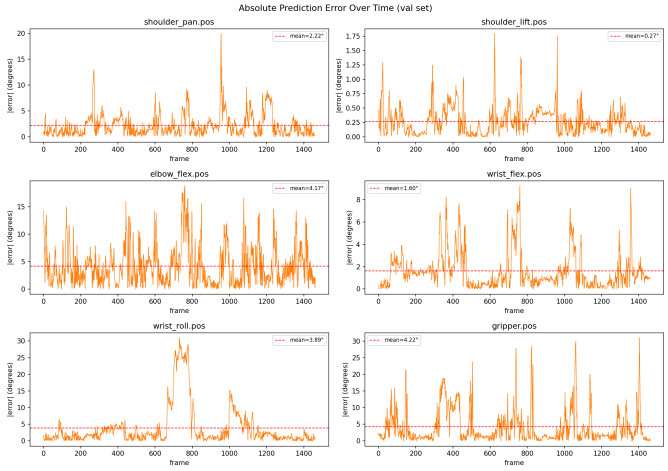

Fig. 9. Absolute prediction error over time for each joint on the validation set. Large spikes indicate instability in predicted actions.

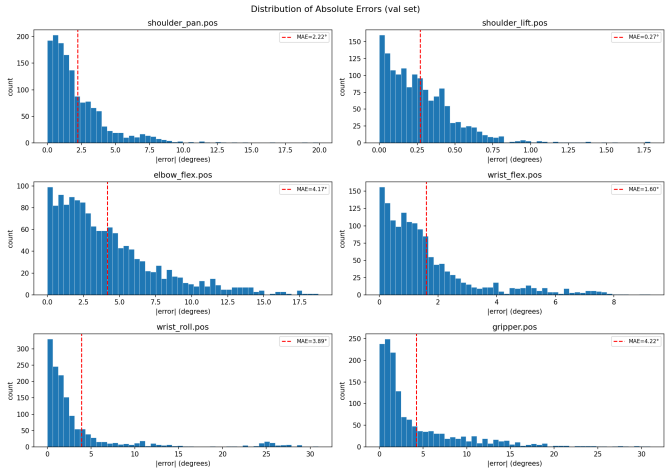

Fig. 10. Distribution of absolute prediction errors for each joint. Some joints exhibit long-tailed error distributions, indicating occasional large prediction errors.

In terms of real-world performance, we evaluated the BC model over 10 trials of the pick-and-place task. The model was unable to successfully complete any stage of the task, resulting in a 0% success rate for pick, place, and full task completion. We should note that our BC model was trained over 14,000 steps while our ACT model was trained over 100,000. However, this suggests that, although the model was able to learn patterns from the training data, the prediction errors accumulated during execution, leading to failure in performing the task on the physical robot.

C. Comparison Between ACT and Behavior Cloning

We compared the performance of the ACT model and our behavior cloning (BC) model to evaluate how different imitation learning approaches perform on the pick-and-place task. The ACT model achieved a 70% pick success rate, 60% place success rate, and 60% full task success rate, while the BC model achieved 0% across all metrics.

One of the main differences between the two models is how they handle temporal information. ACT uses a transformer-based architecture and predicts chunks of future actions, which allows it to generate smoother and more consistent trajectories. In contrast, the BC model predicts actions independently at each timestep, making it more sensitive to small prediction errors. These errors accumulate over time, which leads to unstable behavior during real-world execution.

This difference is reflected in the prediction plots. The BC model is able to follow the general trend of the expert trajectories, but shows noticeable noise and deviations, especially in joints such as the wrist and gripper. The error plots further confirm this, where the BC model exhibits large spikes and higher variance in prediction error. Although these errors may appear small during offline evaluation, they compound over time and prevent the robot from successfully completing the task.

Additionally, the training curves show that while the BC model is able to reduce training loss, the validation loss does not improve significantly, indicating poor generalization. In contrast, the ACT model demonstrates stable training behavior and is able to learn a policy that transfers more effectively to real-world execution.

Overall, these results show that ACT is significantly more effective than standard behavior cloning for this task. The ability to model temporal dependencies and predict sequences of actions allows ACT to better handle the complexity of the manipulation task, while the BC model struggles with error accumulation and instability during execution.

VI. Conclusion

In this project, we explored how imitation learning can be used to train a robotic arm to perform a pick-and-place task using a limited number of demonstrations. We collected our own dataset using a teleoperation setup and evaluated two different approaches: the ACT model provided by the LeRobot framework and our own behavior cloning (BC) model.

From our results, we found that the ACT model was able to successfully learn the task and generalize to new conditions, achieving a 70% pick success rate and 60% full task success rate. The training curves also showed stable learning behavior, and the robot was able to execute smooth and consistent trajectories in the real world. This showed that ACT is a good approach for learning manipulation tasks from a relatively small dataset.

In contrast, while our BC model was able to reduce training loss and partially match the expert (ground) trajectories, it failed to perform the task in real-world execution, achieving a 0% success rate across all trials. The prediction and error plots showed that the model produced noisy outputs and larger deviations, especially for more complex joints. These small errors accumulated over time and led to unstable behavior, preventing the robot from completing the task.

In the future if we continue something like this, we want to explore collecting a larger and more diverse dataset, improving the BC model with temporal components, or experimenting

with other imitation learning methods. Additionally, improving the hardware setup and camera placement could further help the performance and generalization.

Additionally, this project showed the difficulties of working with real-world robotic systems, including data collection, hardware limitations, and training constraints. Despite these challenges, we were able to successfully implement and evaluate two different approaches and gain a better understanding of how imitation learning models perform.

Overall, our final project demonstrates the importance of temporal modeling in robotic manipulation. Models like ACT, which consider sequences of actions, are better suited for these types of tasks, compared to standard behavior cloning that predicts actions independently at each timestep.

VII. What We Learned

At the beginning of this project, we wanted to use robots that made advanced AI robotic learning more accessible, and we came across a video titled "LeRobot – Lowering the Entry Barrier to AI for Robotics" by Google. After watching it, we knew we wanted to try it ourselves, so we purchased the robot and used materials from the CEID (3D printing) and a lot of polycarbonate (PCA) to build it. The physical setup took around 3 hours, since we had to manually adjust some parts to make them fit, and another 3–4 hours to fully set up the robot, calibrate it, and figure out the wiring.

By far the most difficult part of this project was figuring out what we actually wanted to do and how to build our own model using the concepts we learned in machine learning (and computer vision, although only Teresa took that class). Even though our behavior cloning model did not work, we learned that predicting sequences of actions is much more effective than predicting actions one step at a time, which helped us better understand transformer architectures and how crucial temporal modeling is for robotic tasks.

We also learned that there are many trade-offs in machine learning, especially when it comes to balancing performance, training time, and computational resources. Additionally, we learned not to start projects at the last minute. This project took around 120 hours in total, including roughly 60 hours spent on training and tuning hyperparameters to get the best results.

Overall, we learned a lot from this project, including how to read and understand scientific papers, how to design experiments beyond the scope of the class, and even how to create figures and properly format a research paper. Despite the challenges, it was a really valuable experience, and we would definitely do something like this again!

References

[1] "ACT (Action Chunking with Transformers)," LeRobot. [Online]. Available: https://huggingface.co/docs/lerobot/act. [Accessed: Mar. 4, 2026].

[2] J. Hua, L. Zeng, G. Li, and Z. Ju, "Learning for a Robot: Deep Reinforcement Learning, Imitation Learning, Transfer Learning," *Sensors*, vol. 21, no. 4, p. 1278, Feb. 2021.

[3] "Imitation Learning on Real-World Robots," LeRobot. [Online]. Available: https://huggingface.co/docs/lerobot/il_robots. [Accessed: Mar. 4, 2026].

[4] M. Kuwabara *et al.*, "Effectiveness of tuning an artificial intelligence algorithm for cerebral aneurysm diagnosis: A study of 10,000 consecutive cases," *Scientific Reports*, vol. 13, no. 1, p. 16202, Sep. 2023, doi: 10.1038/s41598-023-43418-x.

[5] W. Liang, J. Xie, Z. Wang, J. Tan, and X. Ma, "Constrained behavior cloning for robotic learning," arXiv:2408.10568, Aug. 2024.

[6] "How to Train a Multilayer Perceptron Neural Network," All About Circuits. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/how-to-train-a-multilayer-perceptron-neural-network/.

[7] S. Nahavandi, R. Alizadehsani, D. Nahavandi, C. P. Lim, K. Kelly, and F. Bello, "Machine learning meets advanced robotic manipulation," *Information Fusion*, vol. 105, p. 102221, May 2024.

[8] "ReLU Activation Function in Deep Learning," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/relu-activation-function-in-deep-learning/.

[9] "SO-101," Hugging Face. [Online]. Available: https://huggingface.co/docs/lerobot/en/so101.

[10] "Weights and Biases," Weights & Biases. [Online]. Available: https://wandb.ai/site.

[11] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," arXiv:2304.13705, Apr. 2023, doi: 10.48550/arXiv.2304.13705.