Diffusion Policy Evaluation

Ram Goel
ramkgoel@mit.edu

Franklin Wang
fxwang@mit.edu

Abstract

Diffusion Policy [1] is a recent method for sensorimotor learning in the context of offline reinforcement learning, where the model learns a policy based on demonstration data only (also known as behavior cloning). Diffusion Policies leverage DDPMs (Denoising Diffusion Probabilistic Models)—with key technical differences—to more effectively model the multimodal distribution of the many possible trajectories that the policy can take. We implement the Diffusion Policy algorithm and train it on the Push-T task, obtaining improved performance over the Implicit Behavioral Cloning baseline. We also analyze the effect of different hyperparameters on the model performance.

1 Introduction

Behavior Cloning (BC) in RL is much more challenging than supervised learning since there is not necessarily a single correct ground truth solution. Instead, different trajectories might give equally valid solutions, requiring these models to have multimodal modeling capabilities (Fig. 1). This is especially true when the action space is continuous, and we will be focusing on such environments in this work.

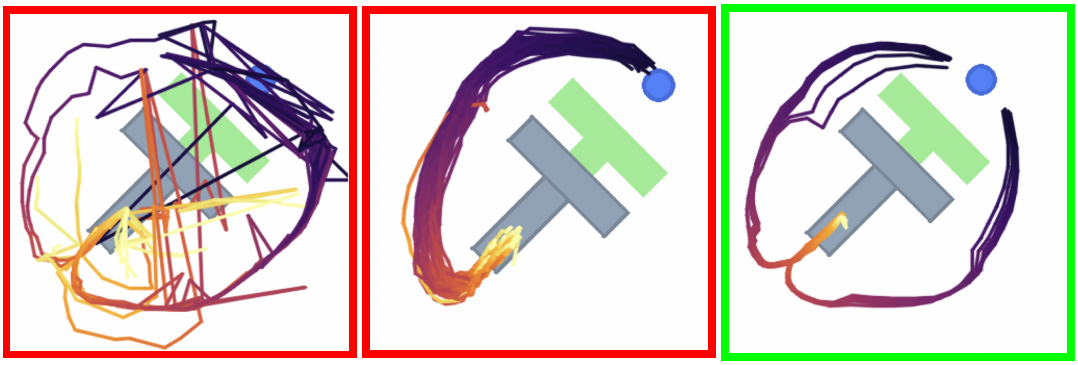

Figure 1: Examples of an instance where there are multiple possible correct trajectories, leading to a multimodal distribution. Left: the policy fails to model the multimodal space and can't commit to a single mode. Middle: The policy only selects a single mode. Right: The policy correctly models both modes of the possible trajectories (this is the kind of behavior we would like our model to have). Images adapted from [1].

1.1 Relevant Works

Early methods for BC relied on explicit policies, such as directly predicting a scalar value in a regressive manner or allowing for the modeling of multimodal distributions through binning or Gaussian mixture models. More recent work such as Behavior Transformers (BET) [7] have continued to improve upon this to allow for better multimodality modeling.

6.484 Computational Sensorimotor Learning Final Project (Spring 2022).

Implicit Behavior Cloning (IBC) [2] is another method which has the capability to model multimodal distributions in an implicit (rather than explicit) manner. IBC evaluates the a certain energy function $E_\theta(o, a)$ for a given observation and action pair. It then finds the action which minimizes this energy function for a given observation, potentially allowing for more complex multimodal distributions to be modeled. We will use IBC as a baseline that we compare the Diffusion Policy to.

1.2 Diffusion Policy Approach and Advantages

Diffusion Policy is another new approach that strives to tackle this problem. The general method of Diffusion Policy is that of a Denoising Diffusion Probabilistic Model (DDPM) [4], adapted to the context of offline policy learning for visuomotor control [1]. Diffusion Policy uses a diffusion-based method to train a conditional denoising network which sequentially predicts actions conditioned on previous observations.

The main advantage of the Diffusion Policy approach is that it has better multimodal modeling capabilities. Diffusion models have been shown to be extremely effective at modeling very complex distributions (e.g. Stable Diffusion [5]), making it much less constrained that other BC approaches. They are also able to be easily conditioned by some feature vector, allowing us to condition our action policy distribution based on the observation. As we stated in the beginning, these capabilities are extremely important since for a given observation, there may be many different valid trajectories that reach an optimal solution, and thus the model needs to be able to ideally capture all of these to truly understand the environment dynamics.

In this paper, we will be striving to replicate the Diffusion Policy paper and comparing our results to the IBC model as a baseline. We will just be focusing on the problem of offline RL for visuomotor control, where the input space consists of images and the output involves controlling a physical agent. Although the Diffusion Policy paper also applies their method to real world robot arms, we will just be testing our models in simulation due to access constraints.

2 Diffusion Policy Method

2.1 Overview of DDPM Method

A Diffusion Policy model which has been given expert data works as follows. Given some randomly sampled noise, we want to iteratively denoise it so that we end up with a possible action that the expert policy could have taken.

To generate our data, we take our action vectors and add random noise to it. To denoise this, we train a neural network whose goal is to predict the difference between a noisy vector and the correct expert action (in other words, the noise which was added to the original action vector). The loss then computes the difference between the ground truth added noise and the noise predicted by the neural network. We can now train the neural network to minimize this loss across many different expert actions and added noise.

Now, this noise prediction network is able to distill out the noise from a piece of data. We can iteratively denoise a randomly sampled noise vector by using the noise prediction network to repeatedly predict and subtract noise from the vector until we finally converge to the correct action vector.

2.1.1 Hyperparameters

The main hyperparmeters used in the Diffusion Policy are as follows:

Hyperparameter	Description
D	Number of diffusion timesteps
T_a	Action horizon, the number of actions predicted at once
T_o	Observation horizon, the number of observations taken in to predict next action
Visual Encoder	ResNet encoder used for visual encoding of images to latent space
Diffusion Scheduler	Denoising hyperparameters varying with iteration step

The diffusion schedulers control the additional denoising hyperparameters α, γ, σ which will be functions of the iteration step k, and we will use the Hugging Face Diffusers library's existing implementations. In particular, we will use the Linear and Squared Cosine Noise Schedulers. These hyperparameters will be used to control the diffusion process as shown in Section 2.4.

2.2 Visual Encoder

In this context of offline policy learning, we are given an expert dataset of valid trajectories. For our context of visumotor control, we will be given a time-indexed image dataset of the task being performed under multiple different starting states. We use a visual encoder to encode the dataset of observations into a latent embedding. The raw image data will be passed into a standard non-pretrained ResNet, with two modifications: (1) Replace global average pooling with a spatial softmax pooling (to maintain spatial information), and (2) Replace BatchNorm with GroupNorm, as has been done for DDPMs [4]. This ResNet model is trained in conjunction with the rest of the model.

2.3 Training

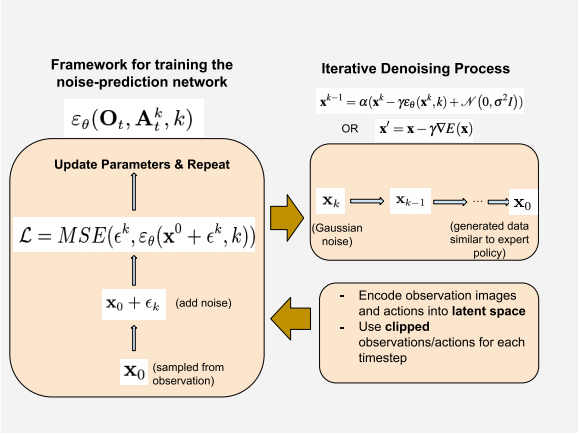

Figure 2: Summary of the Diffusion Training Process: (1) Observation dataset encoded into latent space, (2) Framework for training noise-prediction network ε_θ, (3) Iterative denoising process from randomly sampled Gaussian noise to generated action data.

First, we will describe the process of training a general noise-prediction network ε_θ. This takes in data and the number of noise iterations k, and its goal is to predict the noise from a piece of data. To train this network, we iteratively (1) sample $\mathbf{x}^0$ from the expert dataset, (2) add some Gaussian noise ε^k to get noisy data $\mathbf{x}^0 + \varepsilon^k$, and then (3) compare the prediction of noise from the noisy data to the actual noise:

$$\mathcal{L} = MSE(\varepsilon^k, \varepsilon_\theta(\mathbf{x}^0 + \varepsilon^k, k)). \tag{1}$$

Now, we iteratively apply gradient descent steps to update the parameters θ of the noise prediction network.

Algorithm 1 General DDPM training process

for $i = 1, ..., D$ **do**:
  $\mathbf{x}^0 \sim \mathcal{D}$ (expert dataset)
  $k \leftarrow$ randomly selected denoising iteration
  $\varepsilon^k \sim \mathcal{N}(0, \sigma^2 I)$
  Loss $\mathcal{L} \leftarrow MSE(\varepsilon^k, \varepsilon_\theta(\mathbf{x}^0 + \varepsilon^k, k))$
  $\theta \leftarrow \theta - \gamma d\mathcal{L}/d\theta$
end for

We use the above general framework for DDPM training for Diffusion Policy. The key difference is that we condition on previous observations using a conditioning vector we denote as $\mathbf{O}_t$. To construct the conditioning vector $\mathbf{O}_t$, we concatenate the latent vectors from the visual encoder for the past T_o

observations. Instead of some general x^0, we specifically feed in $\mathbf{A}_t^k$, which is a noisy *sequence* of actions, which allows the model to generate a plan of actions. Thus, the noise-prediction network has the form $\varepsilon_\theta(\mathbf{O}_t, \mathbf{A}_t^k, k)$ for the tth timestep, where k is the number of denoising iterations. This network will have a CNN-based architecture that is similar to the UNet model [6] but modified to allow for the conditioning.

So, for our Diffusion Policy algorithm, we sample an action $\mathbf{A}_0^t$ from the expert dataset and noise ε^k as before, and modify Equation 1 to

$$\mathcal{L} = MSE(\varepsilon^k, \varepsilon_\theta(\mathbf{O}_t, \mathbf{A}_t^0 + \varepsilon^k, k)). \tag{2}$$

2.4 Inference

We will first describe the process of inference in the general DDPM case. Start with $\mathbf{x}^k$ sampled from Gaussian noise, and iteratively denoise $\mathbf{x}^k, \mathbf{x}^{k-1}, \ldots, \mathbf{x}^0$ using the denoising network. This ends with generated data $\mathbf{x}^0$. The iterative denoising process is given by:

$$\mathbf{x}^{k-1} = \alpha(\mathbf{x}^k - \gamma\epsilon_\theta(\mathbf{x}^k, k) + \mathcal{N}(0, \sigma^2 I)) \tag{3}$$

This process essentially repeatedly predicts the noise in the previous iteration and subtracts it. Equation 3 is equivalent to $\mathbf{x}^{k-1} = \mathbf{x} - \gamma\nabla E(\mathbf{x})$, so in essence, the noise-prediction network predicts the gradient field $\nabla E(\mathbf{x})$.

Algorithm 2 General DDPM inference process for k denoising iterations

$\mathbf{x} \sim \mathcal{N}(0, \sigma^2 I)$
for $i = 1, \ldots, k$ **do**
  $\mathbf{x} \leftarrow \alpha(\mathbf{x} - \gamma\varepsilon_\theta(\mathbf{x}, k) + \mathcal{N}(0, \sigma^2 I))$
end for
return $\mathbf{x}$

For Diffusion Policy, the key addition is that the denoising is conditioned on the vector $\mathbf{O}_t$, so we modify the denoising equation from Equation 3 to

$$\mathbf{A}_t^{k-1} = \alpha(\mathbf{A}_t^k - \gamma\varepsilon_\theta(\mathbf{O}_t, \mathbf{A}_t^k, k) + \mathcal{N}(0, \sigma^2 I)). \tag{4}$$

Note that we do not output observation features $\mathbf{O}_t$, only the predicted action conditioned on previous observations and actions.

2.5 Conditional Policy Distribution

Diffusion Policy learns the noise prediction network—which is the gradient of the score function—and iteratively denoises randomly sampled action sequences by conditioning on previous observations and repeatedly applying Langevin dynamics steps. The gradient of the conditional action probability distribution $p(\mathbf{A}_t|\mathbf{O}_t)$ can be found through recursively applying the Langevin method on the score function [8], which in turn can be found using the equation

$$\nabla_{\mathbf{a}} \log p(\mathbf{A}|\mathbf{O}) \approx -\varepsilon_\theta(\mathbf{A}, \mathbf{O}). \tag{5}$$

The output of this is a sequence of actions that the policy believes matches the expert data. When we deploy the policy, we take the first T_a actions from the denoised action seqeuence $\mathbf{A}$ and execute them all at once. This action horizon allows the model to plan out its actions by taking T_a actions at once.

3 Problem Formulation

For the experiments, we will be using the Push-T simulation environment for which the Diffusion Policy has been shown to work well [1]. This environment involves controlling a robot hand (modeled as a circular object that moves around in a 2D grid) to push a T-shaped object to a specific goal position and orientation. We will use the implementation of the Push-T Gym simulation environment found in the [1] paper GitHub repository.

Figure 3: Image of the Push-T environment. The green T is the goal position and orientation, the grey T is the T-shaped object that needs to be moved, the blue circle is the robot hand, and the red cross is where the agent would like to go next.

The observation space consists of 96 by 96 pixel RGB images of the scene from a 2D top-down view. An example of this is shown in Fig. 3.

The action space consists of (x, y) coordinates that are within the bounding box of the scene, and each such action represents the desired (x, y) coordinate for the hand to move to (more specifically, the acceleration of the robot hand is shifted to be closer to the direction of the velocity of the robot hand).

The expert dataset consists of 200 time-indexed observation and action sequences from proficient human demonstrators. The initial configuration of the T starts at randomly sampled positions and rotations.

4 Results

We trained our model over 100 epochs of the data of expert examples using a batch size of 256. We also used the AdamW optimizer with a learning rate of 10^{-4} and a cosine learning rate decay function that stretches over the entire 100 epochs. This resulted in the training curve shown in Fig. 4.

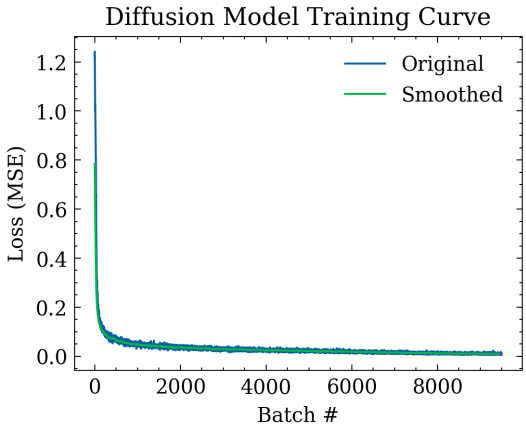

Figure 4: Training curve for our Diffusion Policy model.

4.1 Comparison to Baseline

We evaluate our model and the baseline on the following metrics for the Push-T task by averaging over trajectories limited at 200 timesteps for both models.

1. Average Best IoU: For each trajectory, get the best intersection over union and average across trajectories. Note that sometimes the policy will push the T out of the goal position after already attaining a high IoU, so taking the best reward accounts for this.
2. Success Rate: What percentage of the trajectories attain best IoU > 0.9 (which we consider to be a success).

	Our Best Diffusion Model	IBC Baseline
Average Best IoU	75%	64%
Success Rate (IoU >0.9)	47%	42%

Table 1: Comparison of our model to IBC using our metrics for the Push T task. Our model performs better in all metrics compared to IBC while using less epochs on the same training dataset.

Overall, we were able to beat the IBC baseline on all metrics, suggesting that the Diffusion Policy algorithm outperforms IBC. This is especially true since the pretrained IBC model we used was trained for over 3000 epochs while our model was only trained for 100 epochs using the same dataset due to computational constraints.

4.2 Multimodality Evaluation

One key property of our policy that we would like to demonstrate is multimodal modeling capabilities. We tested this using configurations of the environment that clearly had multiple possible solution trajectories (going left vs going right) and the results are shown in Fig. 5. We also leveraged models of different observation horizons (T_o) to see if this would have an impact. Unfortunately, as shown by the figure, we were unable to show significant multimodality. While our policy is able to generate multiple valid paths, they are always on the same side of the T. We theorize that our inability to capture the multimodal nature of Push-T was because we were not able to train our model for enough epochs since the original Diffusion Paper trained for 3000 epochs [1].

Our results from these experiments also suggest that models with higher T_o perform worse since we can see that they lead to trajectories which collide with the T too early on. This suggests that the larger horizon confuses the model into performing worse (since there is too much unnecessary data).

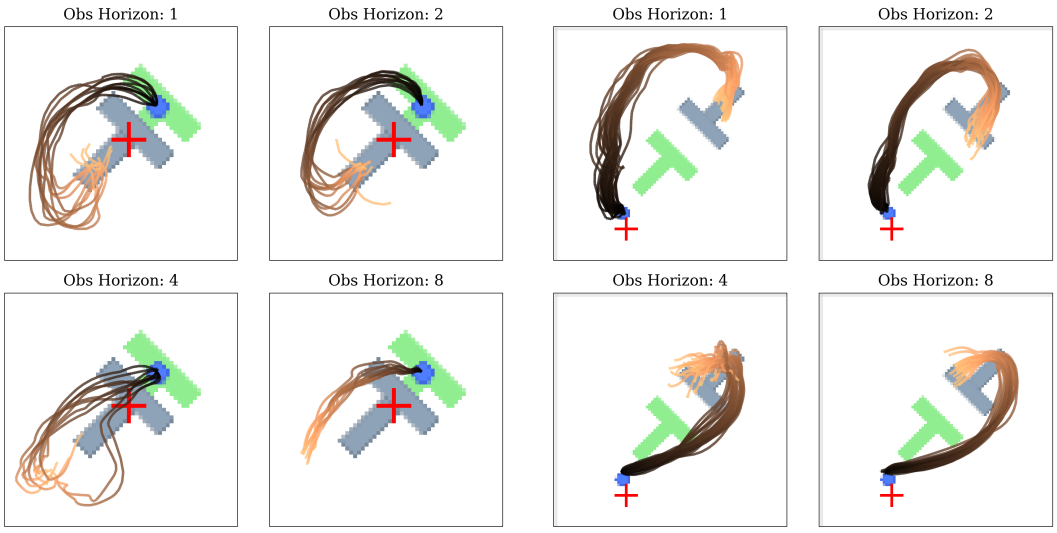

Figure 5: For each of our Diffusion Policy models trained with different observation horizons, we sample trajectories for different starting configurations. Ideally, we would see that the policy learns to sample both possible trajectory modes (going left and going right) but our policy is only able to generate one of these modes, so we were unable to find examples of multimodality in our model. Also, note that when the observation horizon increases, problems with the trajectory such as the robot colliding with the T too early arise, showing that larger observation horizon hurts performance.

4.3 Hyperparameter Importance

We ran tests where we fixed all hyperparameters except for one to see how it influences the results when a model is trained with that set of hyperparameters. This experiment was performed on the following hyperparameters: T_o (observation horizon), T_a (action horizon), D (diffusion timesteps),

ResNet vision encoder, and the diffusion scheduler function. See Section 2.1.1 for an explanation of the hyperparameters. For the ResNet vision encoder, we use 3 variations of the ResNet CNN model (18, 34, and 50 layers). For the diffusion timesteps, we vary the number of timesteps over which the model denoises the random action vector. For the diffusion scheduler function, we compare a squared cosine function and a simple linear function to schedule the amount of noise used over time.

Using 200 sampled trajectories, we calculate the average best IoU and success rate as described in Section 4.1. The results for each of these experiments are shown in Fig. 6.

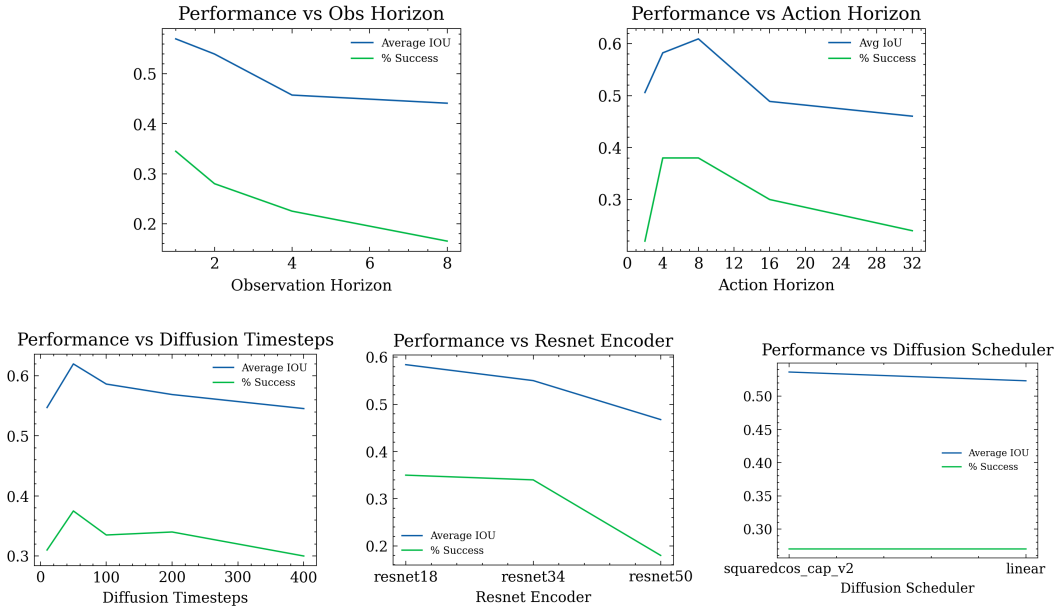

Figure 6: Results from hyperparameter exploration, using the metrics from Section 4.1. From these results we find that the observation horizon, action horizon, and ResNet encoder model are the most critical variables.

From these experiments we conclude the following for each hyperparameter:

1. Observation horizon (T_o): Smaller observation horizons seem to lead to better performance, with our best performance being achieved at a horizon of 1. This suggests that larger horizons lead the model to perform worse since there is too much unnecessary information to process, matching our experiments in Section 4.2.
2. Action horizon (T_a): An action horizon of around 8 leads to the best performance. Smaller action horizons lead the model to be unable to effectively plan its trajectory over many timesteps while large action horizons force the model to make a longer plan than it can handle.
3. Diffusion timesteps (D): It is best to have 50 timesteps, so that there are adequate iterations for the model to find the correct path while not having too many unnecessary iterations.
4. ResNet encoder: The smallest ResNet encoder, resnet18, works the best. This is likely because bigger models do not provide any additional benefit to identifying the position and rotation of the T and they take longer to train.
5. Diffusion scheduler: There is no meaningful distinction between the diffusion schedulers, suggesting that the choice of scheduler is not that important.

5 Issues Stymying Progress

Over the course of this project, some of the issues that we encountered were the following:

1. Computational constraints made it difficult to train our model for a large number of epochs on the order of magnitude the original paper did. This significantly hurt the performance

of our models and the accuracy of our experiments. We will discuss more about how this impacts our findings in the next section.

2. In order to more efficiently run our experiments, we ended up having to spend a lot more time than expected figuring out how to parallelize the environment and inference code so that our experiments could run in a reasonable time.

6 Discussion

Overall, our main takeaway is that the Diffusion Policy is a promising approach that is able to perform better than the IBC baseline without using as many training epochs. Moreover, to ensure that the Diffusion Policy performs well, one must be careful in selecting the observation horizon, action horizon, diffusion timesteps, and vision encoder. These parameters are likely dependent on the task being trained on, so tuning these hyperparameters is key.

Our testing of the manipulation of the diffusion timesteps, vision encoder, and diffusion scheduler is novel and was not done by the original Diffusion Policy paper. For the other hyperparameters however, we had some differences in results from the original paper. In particular, they find that an observation horizon of 2 is better for the Push-T task while we found that a horizon of 1 is best. This may be because we trained for much less epochs than they did, since training for longer may lead to different behavior in the observation horizon since the model will be more capable of processing the information contained in multiple previous timesteps. Our shorter training time also likely resulted in our model being unable to exhibit multimodal behavior, as we discussed previously, and we were not able to match their performance on the metrics we tested on. In future testing of the diffusion policy algorithm it would be beneficial to have the compute resources to train the model for much longer periods for time.

6.1 Limitations and Future Work

One of the main limitations of this work is that it is constrained by the expert data provided. Unlike other behavior cloning work, it is not straightforward to perform online fine-tuning on a Diffusion Policy, since the model does not output the probability of an action being selected (only the gradient of the score function is predicted as shown in Eq. 5). There has been some very recent work which strives to address this, but it has not been tested on visuomotor control so more research is needed to ensure it works for more complex environments [3].

Another area of possible improvement is the architectures of the models used in the denoising and visual encoding process. For instance, instead of a CNN-based UNet model a transformer can also be used. However, the Diffusion Policy paper found that the transformer model is harder to hyperparameter tune, so there is more work to be done in this area to find better architectures [1].

6.2 Contributions

This project involved a lot of collaboration for the replication and analysis of the Diffusion Policy model. The work was roughly split as follows:

1. Ram: Worked on the inference and deployment of the model, processing the expert dataset, and creating the midterm, slides, and final report.
2. Franklin: Worked on training the model, running the hyperparameter and multimodality experiments, and creating the midterm, slides, and final report.

References

[1] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion, 2023.

[2] Pete Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning, 2021.

[3] Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies, 2023.

[4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[5] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[7] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone, 2022.

[8] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2020.