# Assignment 1: Imitation Learning

**Due February 11, 11:59 pm**

## 1 Introduction

In this assignment, you will train action chunking policies for the Push-T environment. You will first train a simple MSE (mean-squared error) policy that predicts action chunks in a single forward pass. Then, you will train a more expressive flow matching policy, similar to diffusion policy (Chi et al., 2025).

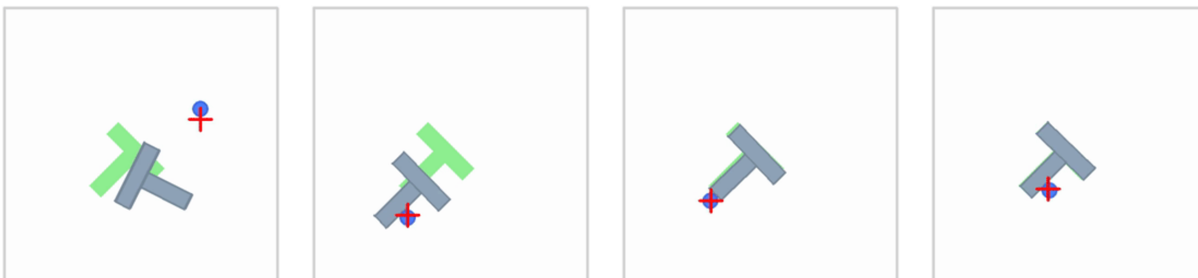

Figure 1: The Push-T environment. The observation is 5-dimensional state describing the position of the T and the agent. The action is a 2-dimensional vector representing the target position of the agent. The goal is to push the T into the goal zone.

## 2 Action Chunking with MSE Loss

Action chunking reduces decision frequency by predicting a short horizon of actions at once. At time $t$, the policy $\pi_\theta(\mathbf{A}_t|\mathbf{o}_t)$ maps the current observation $\mathbf{o}_t$ to an action chunk $\mathbf{A}_t = (\mathbf{a}_t, \mathbf{a}_{t+1}, \ldots, \mathbf{a}_{t+K-1})$ for some fixed chunk length $K$. The chunk is executed open-loop: the environment receives $\mathbf{a}_t$ at time $t$, then $\mathbf{a}_{t+1}$ at time $t+1$, and so on until $\mathbf{a}_{t+K-1}$. After the chunk finishes, the policy is queried again on the latest observation $\mathbf{o}_{t+K}$ to produce the next chunk.

The simplest way to train an action chunking policy is to use a mean-squared error (MSE) loss. That is, given a dataset of paired observations and expert chunks $(\mathbf{o}_t^{(j)}, \mathbf{A}_t^{(j)})$, we fit $\pi_\theta$ by minimizing

$$\mathcal{L}_{\text{MSE}}(\theta) = \frac{1}{B}\sum_{j=1}^{B}\left\|\mathbf{A}_t^{(j)} - \pi_\theta(\mathbf{o}_t^{(j)})\right\|_2^2 \tag{1}$$

for each batch, where $\pi_\theta(\mathbf{o}_t^{(j)})$ denotes the output of the policy network, and $B$ is the batch size.

### 2.1 Implementation

The first part of this assignment is to implement the MSE policy and the main training loop.

The starter code for this assignment can be found at

`https://github.com/berkeleydeeprlcourse/homework_spring2026/tree/main/hw1`

We recommend starting by reading the following files thoroughly:

- `README.md` describes basic project setup.
- `src/hw1_imitation/data.py` defines the dataset class, and handles downloading, extracting, and loading the Push-T dataset for you.
- `src/hw1_imitation/model.py` defines the `BasePolicy` class.

- `src/hw1_imitation/evaluation.py` defines the `evaluate_policy` function. You do **not** ever need to modify this file, but you do need to understand how to call the `evaluate_policy` function periodically in your training loop.

**What you'll need to do**:

- Implement the `MSEPolicy` class by filling in the `TODO` in `src/hw1_imitation/model.py`. We recommend using a simple MLP architecture with ReLU activations.
- Implement the main training loop by filling in the `TODO` in `src/hw1_imitation/train.py`.
- Call the `evaluate_policy` function periodically in your training loop, which will log evaluation metrics and videos to WandB.

**Deliverables**:

- A working training loop and MSE policy.
- WandB-generated logs (including videos) of a successful training run. An MSE policy should be able to achieve a reward of at least 0.5.
- Note that, for grading purposes, you **must** call the `evaluate_policy` function periodically in your training loop, and call `logger.dump_for_grading()` at the end of training.
- A brief report including:
  - Training curves (training steps vs. loss and reward) of your best MSE policy. Please generate these plots yourself rather than taking screenshots of WandB.
  - A brief description of your MLP architecture (number of layers, hidden size, activation functions, etc.).

**Tips**:

- A small MLP should be able to train fairly quickly (within a few minutes) on a laptop CPU.
- Use the Adam optimizer (Kingma, 2014) that is built into PyTorch.
- Use `torch.compile` on the train step to speed up training quite significantly!
- Besides the eval metrics, which are required, you can use WandB to log other useful metrics, such as the loss and training speed.

# 3 Action Chunking with Flow Matching

MSE policies predict chunks in one shot, which can struggle to model complex, multimodal chunk distributions. Flow matching addresses this by learning a conditional vector field that transports noise into realistic action chunks. This is very similar to diffusion, but it is easier to implement and generally exhibits better performance.

Let $\mathbf{A}_t^{(j)}$ be an action chunk and $\mathbf{A}_{t,0}^{(j)} \sim \mathcal{N}(0, I)$ be noise of the same shape. We first sample a "flow matching timestep" $\tau^{(j)} \sim \mathcal{U}(0,1)$ and define the interpolation $\mathbf{A}_{t,\tau}^{(j)} = \tau^{(j)}\mathbf{A}_t^{(j)} + (1-\tau^{(j)})\mathbf{A}_{t,0}^{(j)}$. We then train a network $v_\theta$ to predict the velocity that moves $\mathbf{A}_{t,\tau}^{(j)}$ toward $\mathbf{A}_t^{(j)}$, using the flow-matching loss

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \frac{1}{B}\sum_{j=1}^{B}\left\| v_\theta(\mathbf{o}_t^{(j)}, \mathbf{A}_{t,\tau}^{(j)}, \tau^{(j)}) - (\mathbf{A}_t^{(j)} - \mathbf{A}_{t,0}^{(j)}) \right\|_2^2. \tag{2}$$

At inference time, we sample initial noise $\mathbf{A}_{t,0} \sim \mathcal{N}(0, I)$ and integrate the ODE $\frac{d\mathbf{A}_{t,\tau}}{d\tau} = v_\theta(\mathbf{o}_t, \mathbf{A}_{t,\tau}, \tau)$ from $\tau = 0$ to $\tau = 1$. The simplest integration method is Euler integration, which is given by the following update rule:

$$\mathbf{A}_{t,\tau+\frac{1}{n}} = \mathbf{A}_{t,\tau} + \frac{1}{n} \cdot v_\theta(\mathbf{o}_t, \mathbf{A}_{t,\tau}, \tau), \tag{3}$$

which is repeated $n$ times from $\tau = 0$ to $\tau = 1$ to obtain $\mathbf{A}_{t,1}$, where $n$ is the number of integration steps (also called "denoising steps"). $\mathbf{A}_{t,1} = \mathbf{A}_t$ is the final action chunk that is executed open-loop, as before.

### 3.1 Implementation

For this part, you only need to implement the `FlowMatchingPolicy` class by filling in the `TODO` in `model.py`. We recommend using the same MLP architecture as the MSE policy.

**Deliverables**:

- A working training loop and flow matching policy.
- WandB-generated logs (including videos) of a successful training run. A flow matching policy should be able to achieve a reward of at least 0.7.
- A brief report including:
  - Training curves (training steps vs. loss and reward) of your best flow matching policy. Please generate these plots yourself rather than taking screenshots of WandB.
  - A brief qualitative description of how the flow matching policy behaves compared to the MSE policy (based on the videos).

**Tips**:

- Don't forget that your neural network needs the flow matching timestep, $\tau$, as part of its input.

## 4 Submitting the Code and Experiment Runs

In order to turn in your code and experiment logs, create a directory that contains the following:

- A directory named `exp` with your best experiment runs for this assignment. The experiment runs will initially be saved with names like `seed_42_20260119_161512_my_expriment_name`; you should **rename** the directories accordingly as specified below, and include only your best run for each part.
- The `src` folder with all the `.py` files, with the same names and directory structure as the original homework repository.

The unzipped version of your submission should have the following file structure. **Make sure that the `exp` directory has the exact structure as shown below. Make sure that you copy the entire run directory, including all of the `.wandb`, `.json`, `.mp4`, and `.pkl` files.**

```
submit.zip
├─ exp
│  ├─ mse
│  │  ├─ log.csv
│  │  └─ wandb
│  │     └─ ...
│  └─ flow
│     ├─ log.csv
│     └─ wandb
│        └─ ...
├─ src
│  └─ hw1_imitation
│     ├─ model.py
│     ├─ train.py
│     ├─ data.py
│     └─ evaluation.py
├─ pyproject.toml
├─ uv.lock
└─ README.md
```

Turn in your assignment on Gradescope. Upload the zip file with your code and log files to **HW1 Code**, and your report to **HW1 Report**.

# References

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.