Technical Report

Transformer-Based Behaviour Cloning Using PPO Expert Demonstrations

Submitted By

Prathamesh Palatshaha [25201310]

MTech - AI

Submitted To

Dr. Ranjeet Kumar Rout

IT Department

November 2025

Contents

1. Abstract

This project shows a full imitation learning pipeline in which a Transformer-based policy learns to balance the cartpole by copying a high-quality PPO expert. The PPO agent learns to act almost perfectly through reinforcement learning. After that, its paths are collected and used to train a Behaviour Cloning (BC) model in a supervised way. The BC model uses a Transformer encoder to find time-based connections between stacked observations. The last model is tested in the environment and can accurately mimic the expert's behaviour, showing that Transformers work well for imitation-based control tasks.

2. Introduction to the Algorithm

Imitation Learning (IL) emphasises acquiring decision-making policies through the observation of an expert, rather than through direct interaction with the environment. We use Proximal Policy Optimisation (PPO), a reliable and stable reinforcement learning algorithm, to create expert actions for this project. These expert demonstrations are used to teach a Transformer-based Behaviour Cloning model how to turn a stack of observations into expert actions.
The pipeline is made up of:

1. PPO training to figure out the best way to control a Cart-Pole.
2. A collection of expert demonstrations (state–action pairs).
3. Teaching a Transformer model to copy the expert policy (BC).
4. Testing how well the BC model works and making a video of it in action.

This method is like modern robotics pipelines like ACT, RT-1, and RT-2, where big models learn from expert demonstrations instead of expensive RL interactions.

3. Mathematical Interpretation

3.1 PPO Expert Training

Cart-Pole is formulated as an MDP:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

PPO optimizes the clipped surrogate loss:

$$L^{CLIP}(\theta) = E[\min(r_t(\theta)A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

This ensures controlled, stable policy updates.

3.2 Behaviour Cloning (Supervised Imitation)

Once PPO has converged, we collect dataset:

$$\mathcal{D} = \{(x_t, a_t)\}_{t=1}^{N}$$

where

$$x_t \textit{ is a } \boldsymbol{stacked\ observation} \textit{ and } a_t \textit{ is the expert action.}$$

BC minimizes cross-entropy:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{N} \log \pi_\theta (a_t|x_t)$$

3.3 Transformer Policy Model

Each stacked observation x_t is passed through:

Embedding

$$h_0 = W_{\text{embed}} x_t$$

Self-Attention

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Action Prediction

$$\pi_\theta(a|x_t) = softmax(W_{\text{out}} \cdot h_L)$$

The model learns to mimic the PPO expert without interacting with the environment.

4. Pseudocode for the Complete Algorithm

Algorithm: Transformer-Based Behaviour Cloning from PPO Expert

Step 1: PPO Expert Training

```
Initialize PPO policy π_E
for t in 1 … T:
  s_t ← environment state
  a_t ← sample from π_E(a | s_t)
  Execute a_t and store transition
  Update π_E using PPO clipped objective
Save π_E
```

Step 2: Dataset Collection

```
Initialize dataset D = {}
for each episode:
  Reset env → s
  Initialize a stack of last K observations
  while episode not done:
    a = π_E(s)
    x = stacked observation
    Add (x, a) to D
    s ← next state
Save D
```

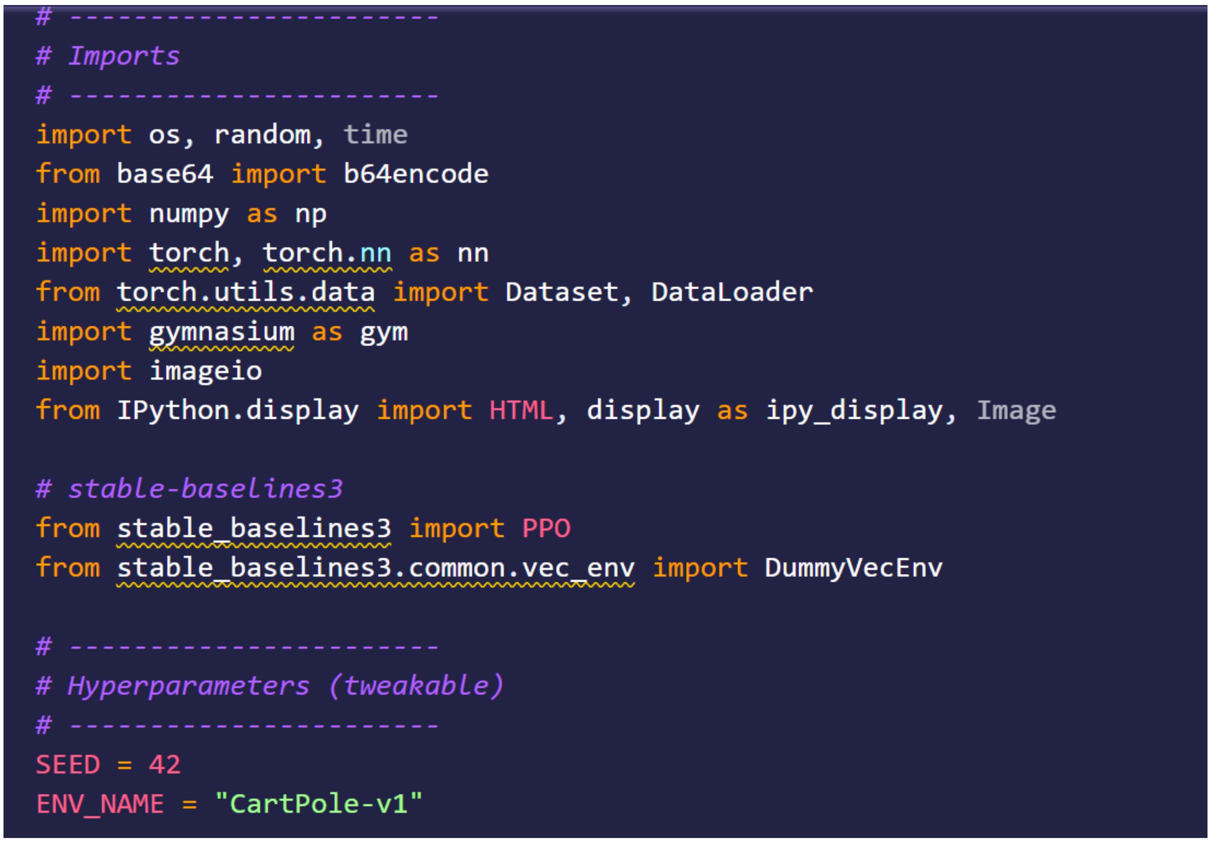
Step 3: Train Transformer for Behavior Cloning

```
Load dataset D & Normalize observations
Initialize Transformer policy π_θ
for epoch = 1 to E:
  for each mini-batch (x_b, a_b):
    logits = π_θ(x_b)
    loss = CrossEntropy(logits, a_b)
    Backpropagate loss and update θ
Save π_θ
```

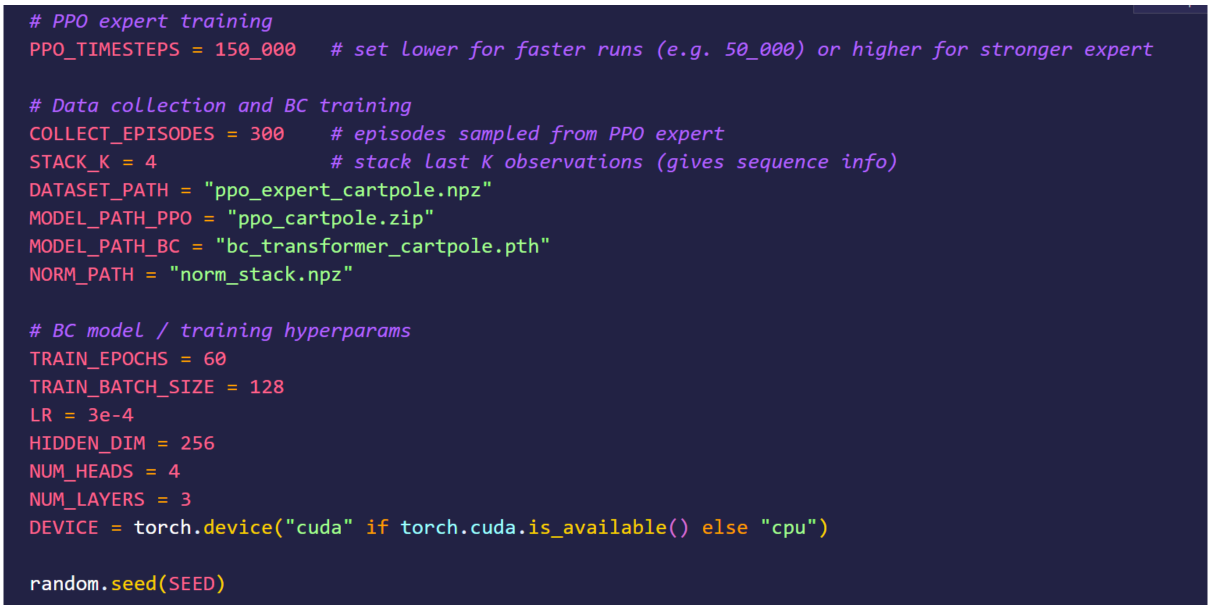
Step 4: Evaluate and Record

```
Load model $\pi_\theta$
for each evaluation episode:
    Reset env
    while not done:
        x ← stacked observation
        a_pred = argmax $\pi_\theta(a | x)$
        Step environment and record frame
Save frames as MP4/GIF
```

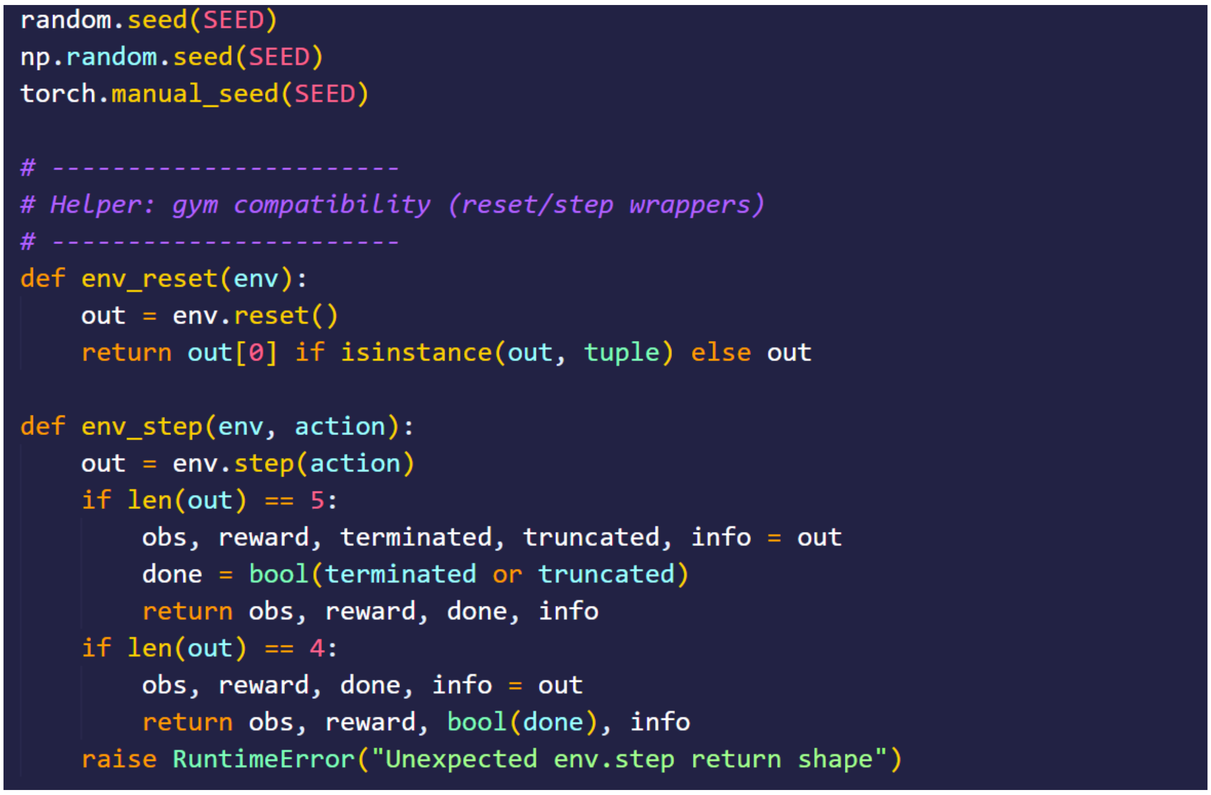
4. Implementation in Python



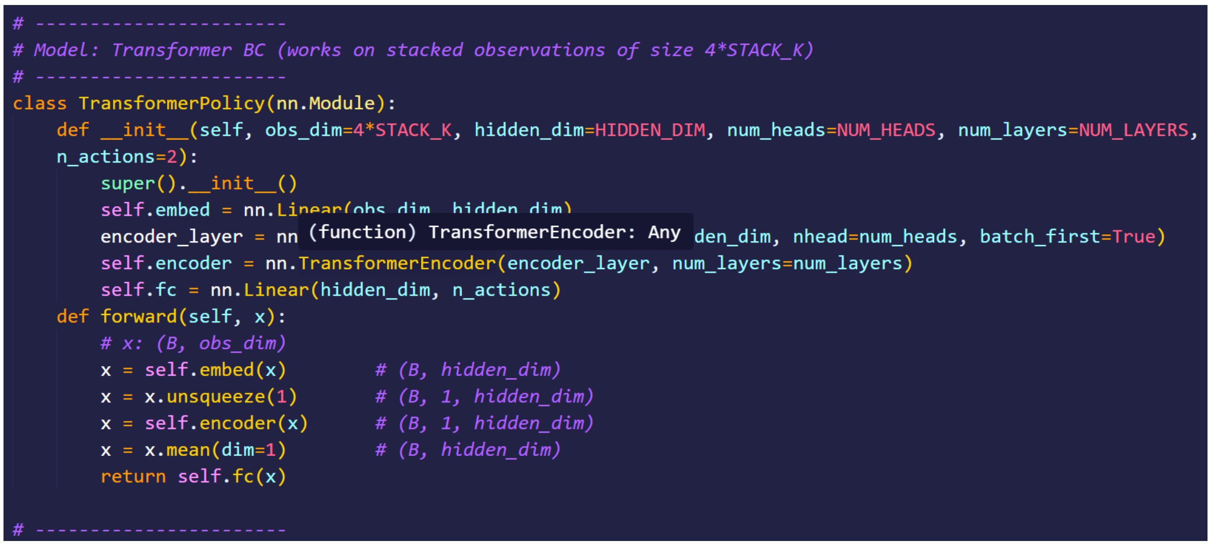
```python
# -----------------------
# Imports
# -----------------------
import os, random, time
from base64 import b64encode
import numpy as np
import torch, torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import gymnasium as gym
import imageio
from IPython.display import HTML, display as ipy_display, Image

# stable-baselines3
from stable_baselines3 import PPO
from stable_baselines3.common.vec_env import DummyVecEnv

# -----------------------
# Hyperparameters (tweakable)
# -----------------------
SEED = 42
ENV_NAME = "CartPole-v1"
```



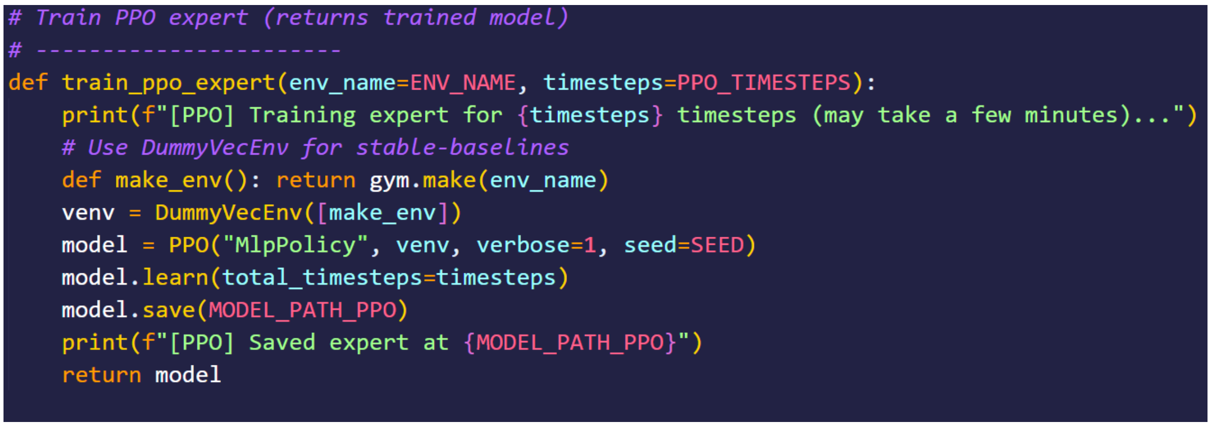
```python
# PPO expert training
PPO_TIMESTEPS = 150_000   # set lower for faster runs (e.g. 50_000) or higher for stronger expert

# Data collection and BC training
COLLECT_EPISODES = 300    # episodes sampled from PPO expert
STACK_K = 4               # stack last K observations (gives sequence info)
DATASET_PATH = "ppo_expert_cartpole.npz"
MODEL_PATH_PPO = "ppo_cartpole.zip"
MODEL_PATH_BC = "bc_transformer_cartpole.pth"
NORM_PATH = "norm_stack.npz"

# BC model / training hyperparams
TRAIN_EPOCHS = 60
TRAIN_BATCH_SIZE = 128
LR = 3e-4
HIDDEN_DIM = 256
NUM_HEADS = 4
NUM_LAYERS = 3
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

random.seed(SEED)
```



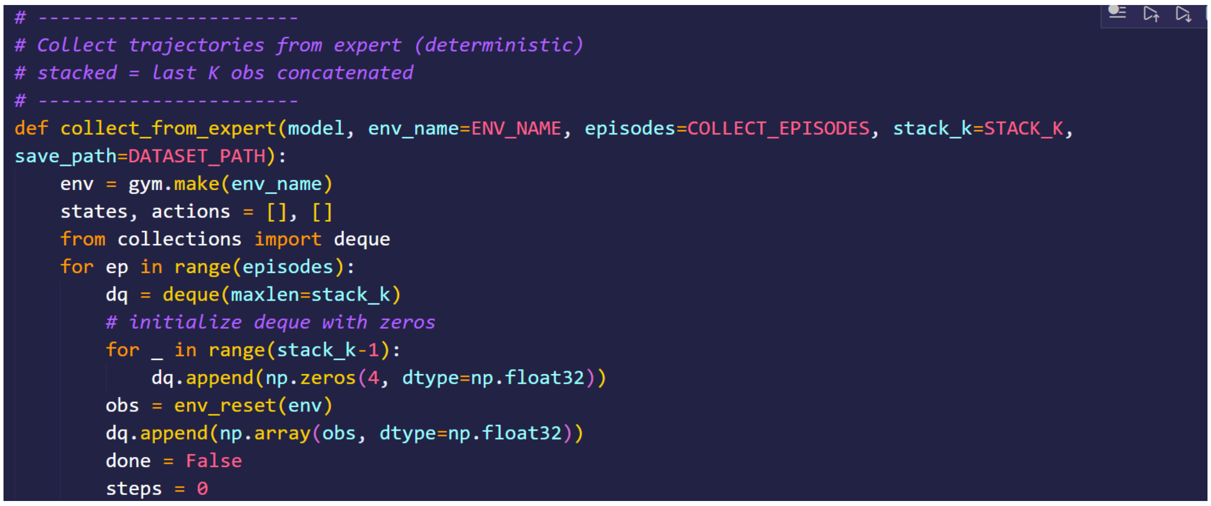
```python
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)

# -----------------------
# Helper: gym compatibility (reset/step wrappers)
# -----------------------
def env_reset(env):
    out = env.reset()
    return out[0] if isinstance(out, tuple) else out

def env_step(env, action):
    out = env.step(action)
    if len(out) == 5:
        obs, reward, terminated, truncated, info = out
        done = bool(terminated or truncated)
        return obs, reward, done, info
    if len(out) == 4:
        obs, reward, done, info = out
        return obs, reward, bool(done), info
    raise RuntimeError("Unexpected env.step return shape")
```



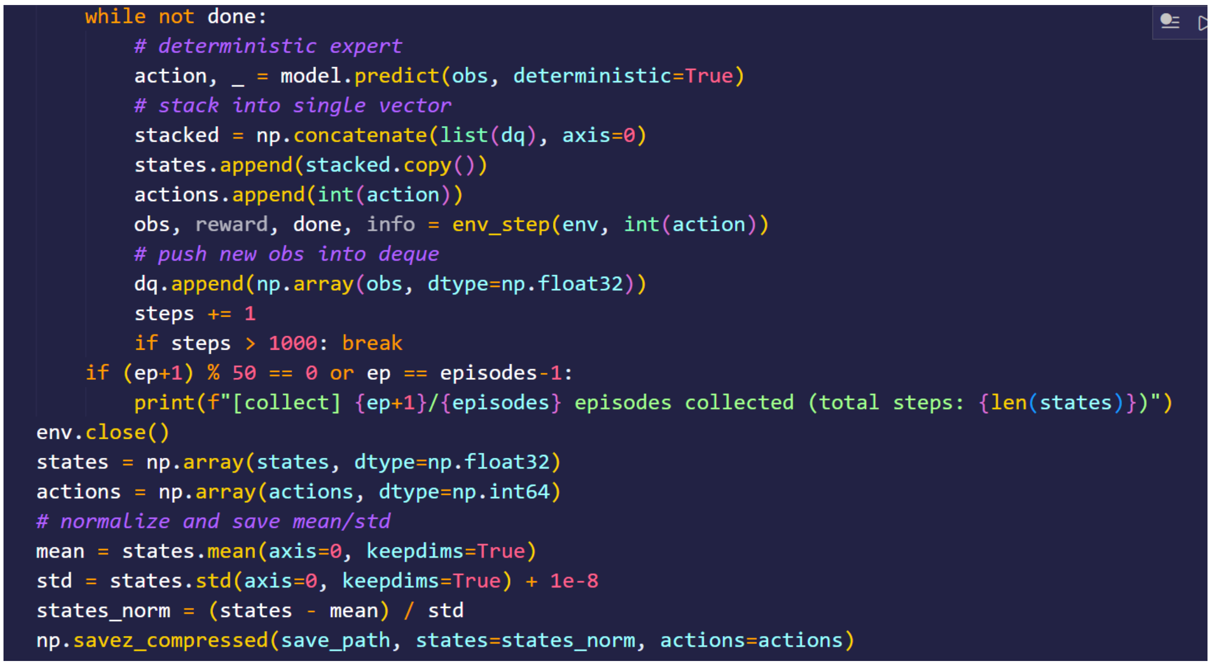
```python
# -----------------------
# Model: Transformer BC (works on stacked observations of size 4*STACK_K)
# -----------------------
class TransformerPolicy(nn.Module):
    def __init__(self, obs_dim=4*STACK_K, hidden_dim=HIDDEN_DIM, num_heads=NUM_HEADS, num_layers=NUM_LAYERS,
    n_actions=2):
        super().__init__()
        self.embed = nn.Linear(obs_dim, hidden_dim)
        encoder_layer = nn (function) TransformerEncoder: Any den_dim, nhead=num_heads, batch_first=True)
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.fc = nn.Linear(hidden_dim, n_actions)
    def forward(self, x):
        # x: (B, obs_dim)
        x = self.embed(x)        # (B, hidden_dim)
        x = x.unsqueeze(1)       # (B, 1, hidden_dim)
        x = self.encoder(x)      # (B, 1, hidden_dim)
        x = x.mean(dim=1)        # (B, hidden_dim)
        return self.fc(x)

# -----------------------
```



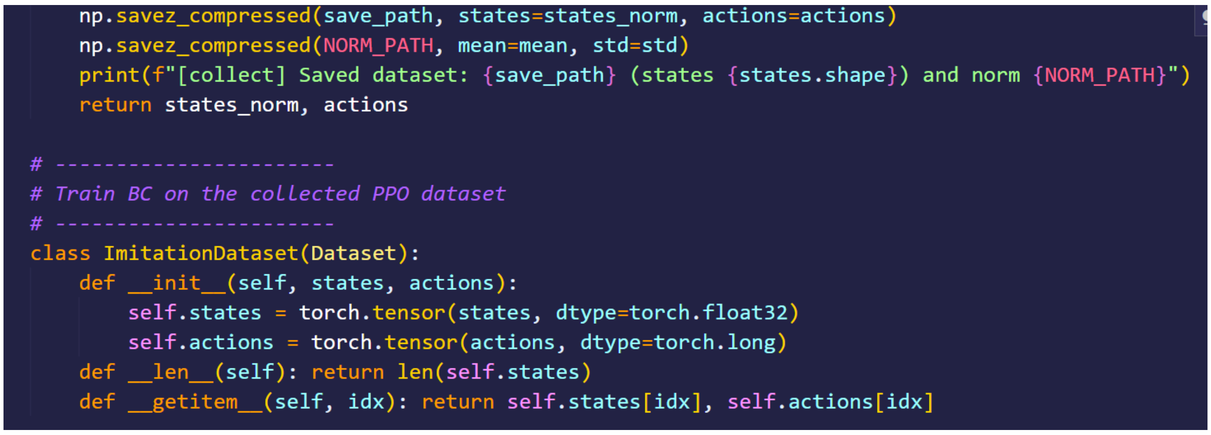
```python
# Train PPO expert (returns trained model)
# -----------------------
def train_ppo_expert(env_name=ENV_NAME, timesteps=PPO_TIMESTEPS):
    print(f"[PPO] Training expert for {timesteps} timesteps (may take a few minutes)...")
    # Use DummyVecEnv for stable-baselines
    def make_env(): return gym.make(env_name)
    venv = DummyVecEnv([make_env])
    model = PPO("MlpPolicy", venv, verbose=1, seed=SEED)
    model.learn(total_timesteps=timesteps)
    model.save(MODEL_PATH_PPO)
    print(f"[PPO] Saved expert at {MODEL_PATH_PPO}")
    return model
```



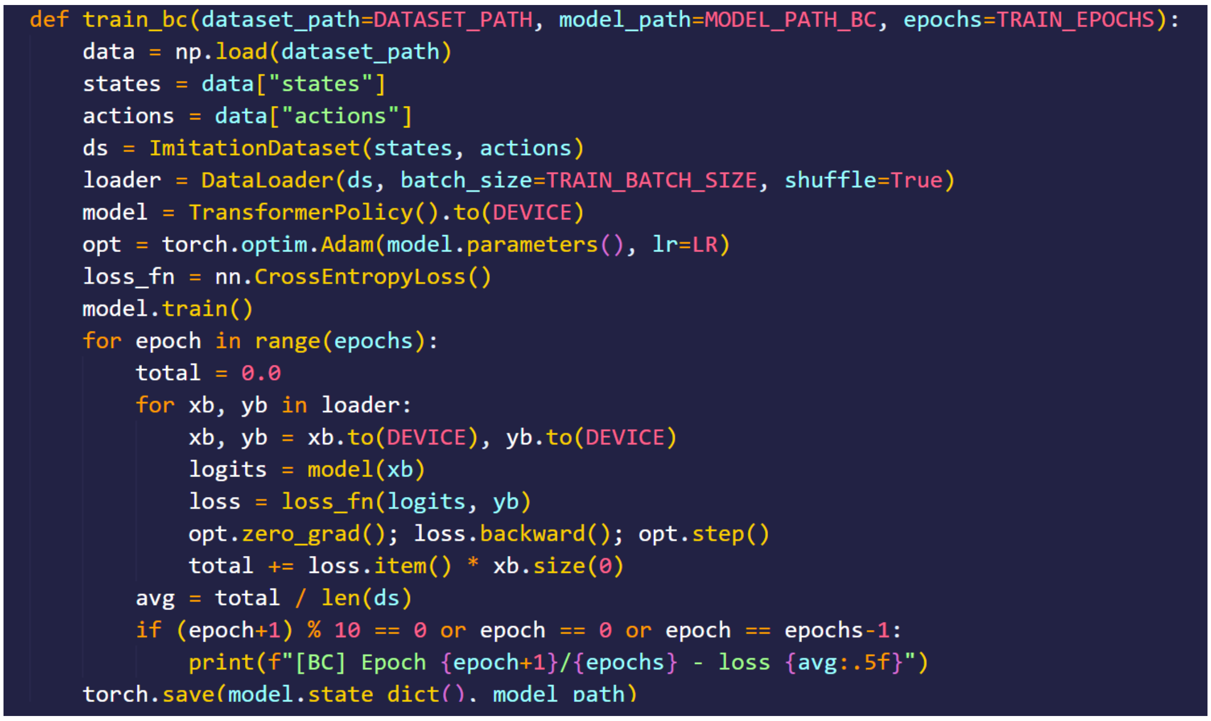
```python
# -----------------------
# Collect trajectories from expert (deterministic)
# stacked = last K obs concatenated
# -----------------------
def collect_from_expert(model, env_name=ENV_NAME, episodes=COLLECT_EPISODES, stack_k=STACK_K,
save_path=DATASET_PATH):
    env = gym.make(env_name)
    states, actions = [], []
    from collections import deque
    for ep in range(episodes):
        dq = deque(maxlen=stack_k)
        # initialize deque with zeros
        for _ in range(stack_k-1):
            dq.append(np.zeros(4, dtype=np.float32))
        obs = env_reset(env)
        dq.append(np.array(obs, dtype=np.float32))
        done = False
        steps = 0
```



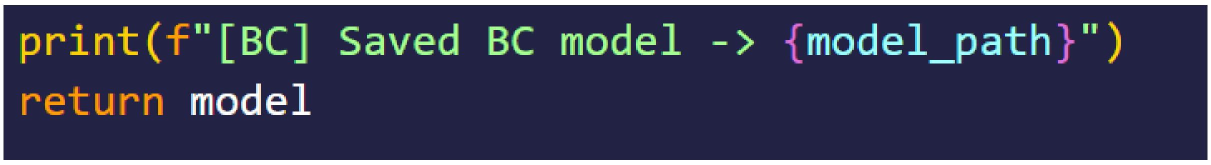
```python
        while not done:
            # deterministic expert
            action, _ = model.predict(obs, deterministic=True)
            # stack into single vector
            stacked = np.concatenate(list(dq), axis=0)
            states.append(stacked.copy())
            actions.append(int(action))
            obs, reward, done, info = env_step(env, int(action))
            # push new obs into deque
            dq.append(np.array(obs, dtype=np.float32))
            steps += 1
            if steps > 1000: break
        if (ep+1) % 50 == 0 or ep == episodes-1:
            print(f"[collect] {ep+1}/{episodes} episodes collected (total steps: {len(states)})")
    env.close()
    states = np.array(states, dtype=np.float32)
    actions = np.array(actions, dtype=np.int64)
    # normalize and save mean/std
    mean = states.mean(axis=0, keepdims=True)
    std = states.std(axis=0, keepdims=True) + 1e-8
    states_norm = (states - mean) / std
    np.savez_compressed(save_path, states=states_norm, actions=actions)
```



```python
    np.savez_compressed(save_path, states=states_norm, actions=actions)
    np.savez_compressed(NORM_PATH, mean=mean, std=std)
    print(f"[collect] Saved dataset: {save_path} (states {states.shape}) and norm {NORM_PATH}")
    return states_norm, actions

# -----------------------
# Train BC on the collected PPO dataset
# -----------------------
class ImitationDataset(Dataset):
    def __init__(self, states, actions):
        self.states = torch.tensor(states, dtype=torch.float32)
        self.actions = torch.tensor(actions, dtype=torch.long)
    def __len__(self): return len(self.states)
    def __getitem__(self, idx): return self.states[idx], self.actions[idx]
```

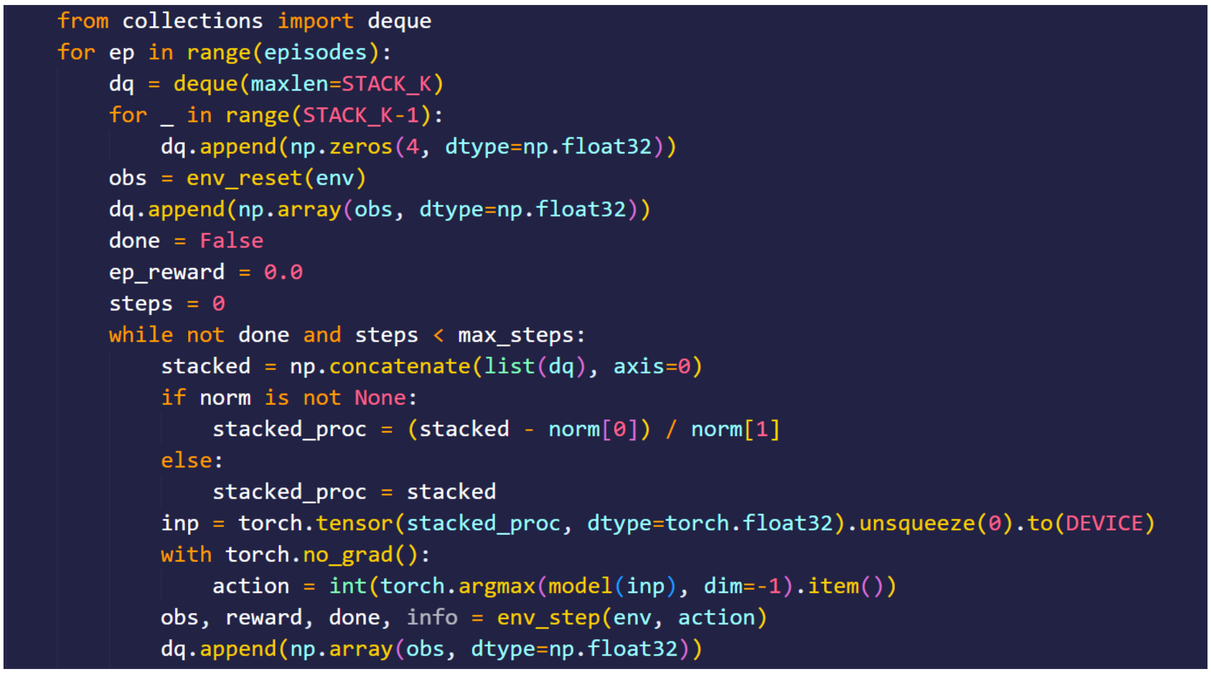


```python
def train_bc(dataset_path=DATASET_PATH, model_path=MODEL_PATH_BC, epochs=TRAIN_EPOCHS):
    data = np.load(dataset_path)
    states = data["states"]
    actions = data["actions"]
    ds = ImitationDataset(states, actions)
    loader = DataLoader(ds, batch_size=TRAIN_BATCH_SIZE, shuffle=True)
    model = TransformerPolicy().to(DEVICE)
    opt = torch.optim.Adam(model.parameters(), lr=LR)
    loss_fn = nn.CrossEntropyLoss()
    model.train()
    for epoch in range(epochs):
        total = 0.0
        for xb, yb in loader:
            xb, yb = xb.to(DEVICE), yb.to(DEVICE)
            logits = model(xb)
            loss = loss_fn(logits, yb)
            opt.zero_grad(); loss.backward(); opt.step()
            total += loss.item() * xb.size(0)
        avg = total / len(ds)
        if (epoch+1) % 10 == 0 or epoch == 0 or epoch == epochs-1:
            print(f"[BC] Epoch {epoch+1}/{epochs} - loss {avg:.5f}")
    torch.save(model.state_dict(), model_path)
```

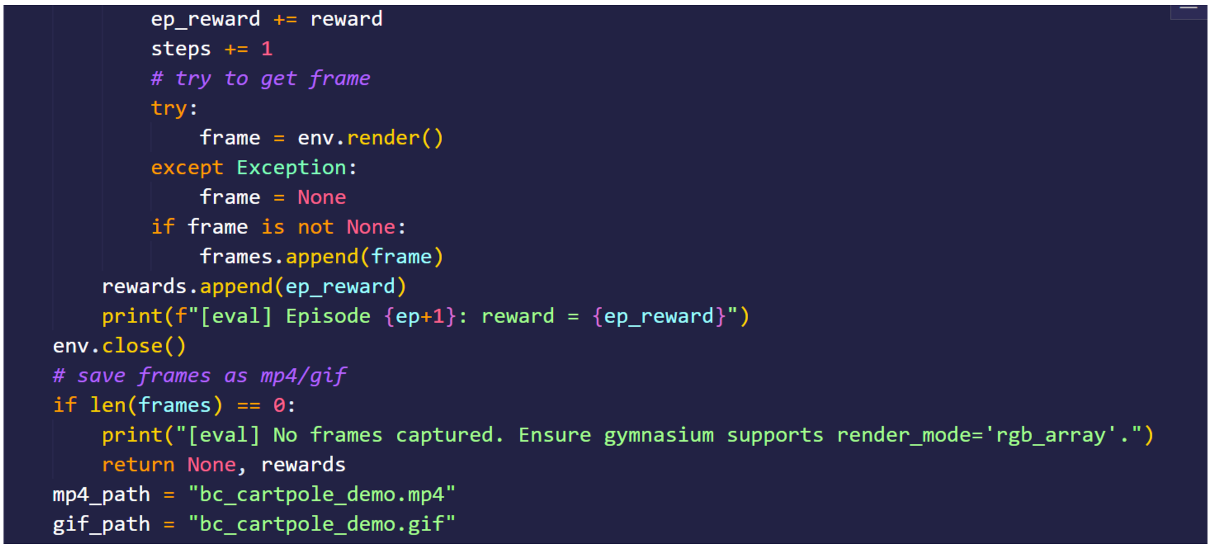


```python
print(f"[BC] Saved BC model -> {model_path}")
return model
```

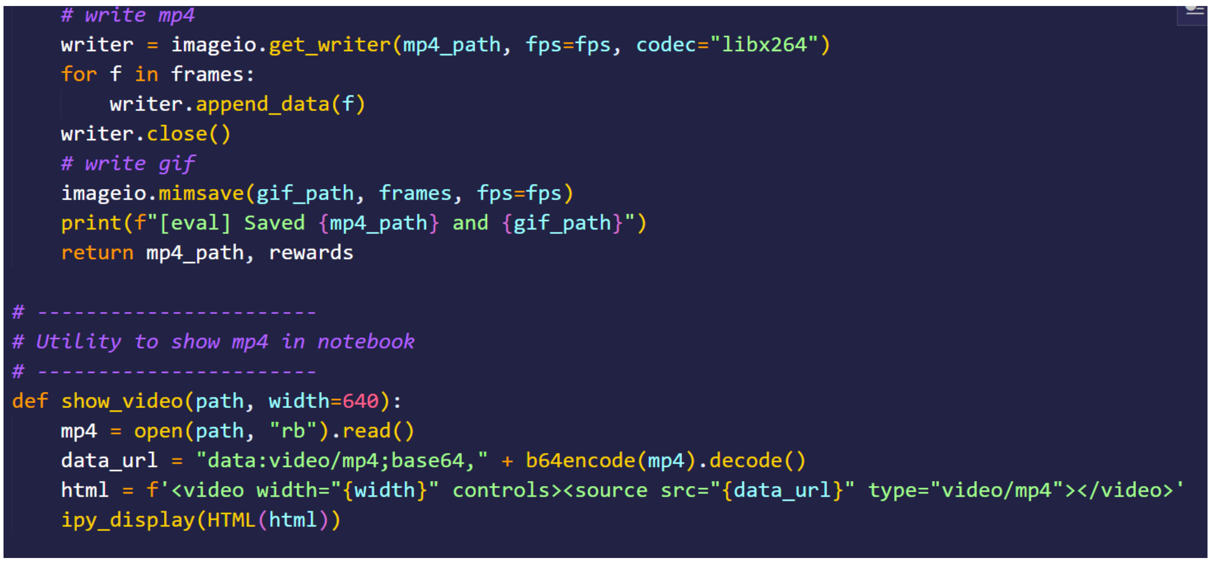
```python
# Evaluate BC and record frames (rgb_array)
# ------------------------
def evaluate_and_record_bc(model_path=MODEL_PATH_BC, env_name=ENV_NAME, episodes=2, max_steps=500, fps=30):
    # load normalization
    norm = None
    if os.path.exists(NORM_PATH):
        d = np.load(NORM_PATH)
        mean, std = d["mean"], d["std"]
        norm = (mean.squeeze(), std.squeeze())
    # create env with rgb_array
    try:
        env = gym.make(env_name, render_mode="rgb_array")
    except TypeError:
        env = gym.make(env_name)
    model = TransformerPolicy().to(DEVICE)
    model.load_state_dict(torch.load(model_path, map_location=DEVICE))
    model.eval()
    frames = []
    rewards = []
    from collections import deque
    for ep in range(episodes):
```



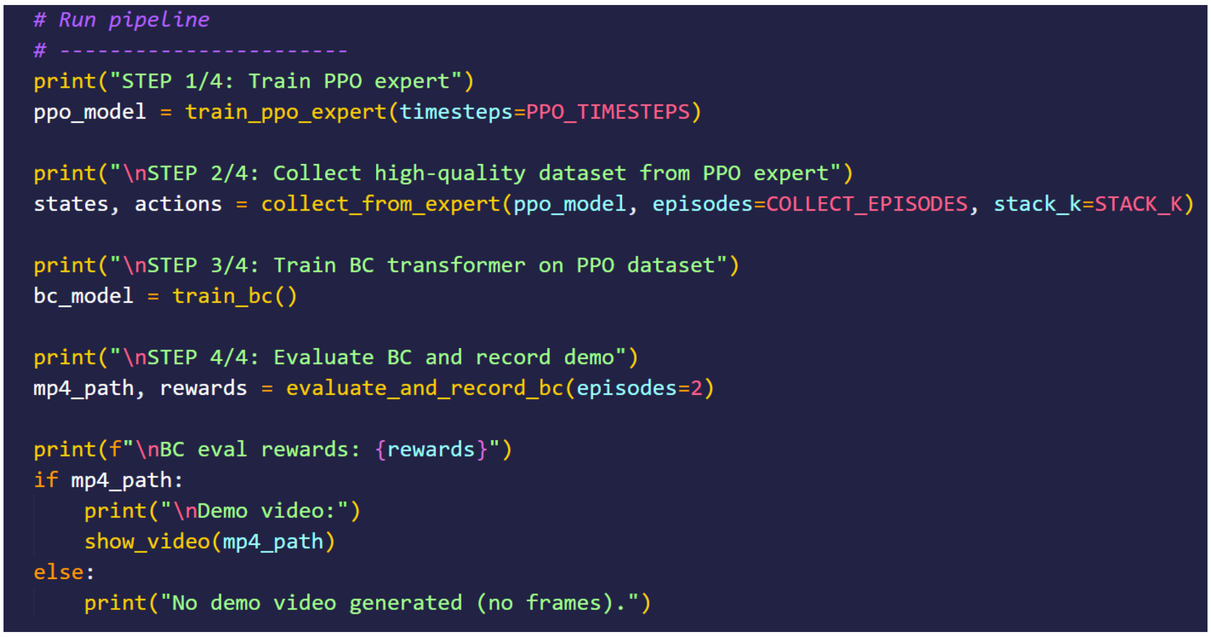
```
    from collections import deque
    for ep in range(episodes):
        dq = deque(maxlen=STACK_K)
        for _ in range(STACK_K-1):
            dq.append(np.zeros(4, dtype=np.float32))
        obs = env_reset(env)
        dq.append(np.array(obs, dtype=np.float32))
        done = False
        ep_reward = 0.0
        steps = 0
        while not done and steps < max_steps:
            stacked = np.concatenate(list(dq), axis=0)
            if norm is not None:
                stacked_proc = (stacked - norm[0]) / norm[1]
            else:
                stacked_proc = stacked
            inp = torch.tensor(stacked_proc, dtype=torch.float32).unsqueeze(0).to(DEVICE)
            with torch.no_grad():
                action = int(torch.argmax(model(inp), dim=-1).item())
            obs, reward, done, info = env_step(env, action)
            dq.append(np.array(obs, dtype=np.float32))
```



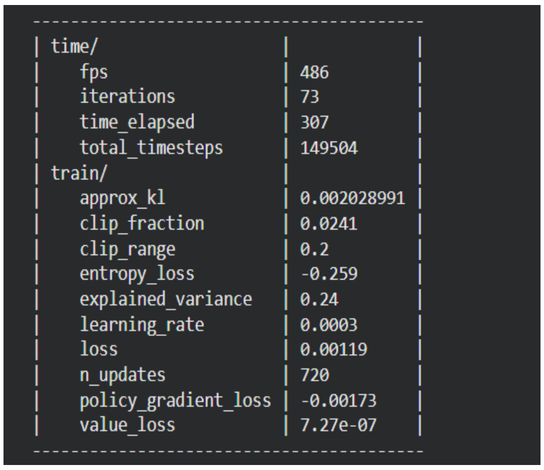
```
            ep_reward += reward
            steps += 1
            # try to get frame
            try:
                frame = env.render()
            except Exception:
                frame = None
            if frame is not None:
                frames.append(frame)
        rewards.append(ep_reward)
        print(f"[eval] Episode {ep+1}: reward = {ep_reward}")
    env.close()
    # save frames as mp4/gif
    if len(frames) == 0:
        print("[eval] No frames captured. Ensure gymnasium supports render_mode='rgb_array'.")
        return None, rewards
    mp4_path = "bc_cartpole_demo.mp4"
    gif_path = "bc_cartpole_demo.gif"
```



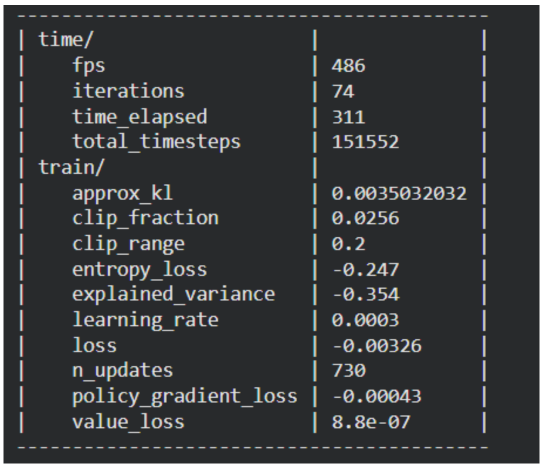
```
    # write mp4
    writer = imageio.get_writer(mp4_path, fps=fps, codec="libx264")
    for f in frames:
        writer.append_data(f)
    writer.close()
    # write gif
    imageio.mimsave(gif_path, frames, fps=fps)
    print(f"[eval] Saved {mp4_path} and {gif_path}")
    return mp4_path, rewards

# -----------------------
# Utility to show mp4 in notebook
# -----------------------
def show_video(path, width=640):
    mp4 = open(path, "rb").read()
    data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
    html = f'<video width="{width}" controls><source src="{data_url}" type="video/mp4"></video>'
    ipy_display(HTML(html))
```



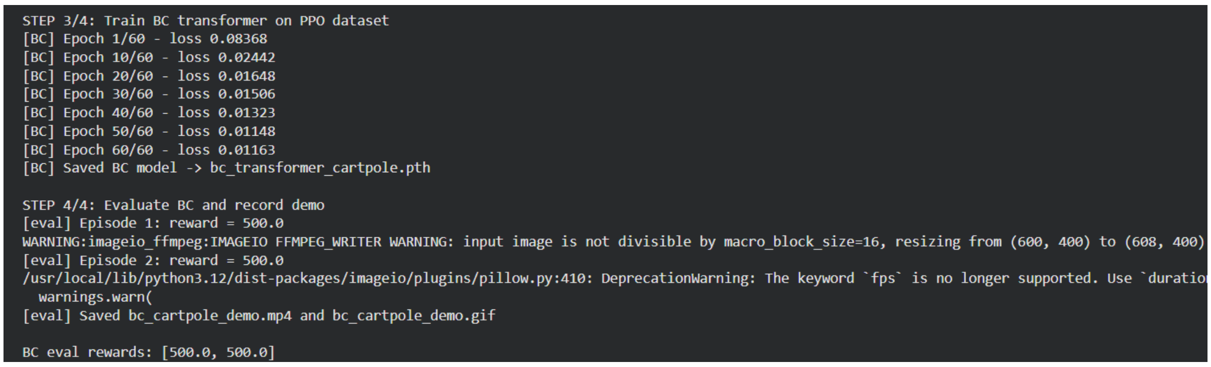
```python
# Run pipeline
# -----------------------
print("STEP 1/4: Train PPO expert")
ppo_model = train_ppo_expert(timesteps=PPO_TIMESTEPS)

print("\nSTEP 2/4: Collect high-quality dataset from PPO expert")
states, actions = collect_from_expert(ppo_model, episodes=COLLECT_EPISODES, stack_k=STACK_K)

print("\nSTEP 3/4: Train BC transformer on PPO dataset")
bc_model = train_bc()

print("\nSTEP 4/4: Evaluate BC and record demo")
mp4_path, rewards = evaluate_and_record_bc(episodes=2)

print(f"\nBC eval rewards: {rewards}")
if mp4_path:
    print("\nDemo video:")
    show_video(mp4_path)
else:
    print("No demo video generated (no frames).")
```



```
 -----------------------------------------
| time/                   |              |
|    fps                  | 486          |
|    iterations           | 73           |
|    time_elapsed         | 307          |
|    total_timesteps      | 149504       |
| train/                  |              |
|    approx_kl            | 0.002028991  |
|    clip_fraction        | 0.0241       |
|    clip_range           | 0.2          |
|    entropy_loss         | -0.259       |
|    explained_variance   | 0.24         |
|    learning_rate        | 0.0003       |
|    loss                 | 0.00119      |
|    n_updates            | 720          |
|    policy_gradient_loss | -0.00173     |
|    value_loss           | 7.27e-07     |
 -----------------------------------------
```



```
 -----------------------------------------
| time/                   |              |
|    fps                  | 486          |
|    iterations           | 74           |
|    time_elapsed         | 311          |
|    total_timesteps      | 151552       |
| train/                  |              |
|    approx_kl            | 0.0035032032 |
|    clip_fraction        | 0.0256       |
|    clip_range           | 0.2          |
|    entropy_loss         | -0.247       |
|    explained_variance   | -0.354       |
|    learning_rate        | 0.0003       |
|    loss                 | -0.00326     |
|    n_updates            | 730          |
|    policy_gradient_loss | -0.00043     |
|    value_loss           | 8.8e-07      |
 -----------------------------------------
```



```
STEP 3/4: Train BC transformer on PPO dataset
[BC] Epoch 1/60 - loss 0.08368
[BC] Epoch 10/60 - loss 0.02442
[BC] Epoch 20/60 - loss 0.01648
[BC] Epoch 30/60 - loss 0.01506
[BC] Epoch 40/60 - loss 0.01323
[BC] Epoch 50/60 - loss 0.01148
[BC] Epoch 60/60 - loss 0.01163
[BC] Saved BC model -> bc_transformer_cartpole.pth

STEP 4/4: Evaluate BC and record demo
[eval] Episode 1: reward = 500.0
WARNING:imageio_ffmpeg:IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by macro_block_size=16, resizing from (600, 400) to (608, 400)
[eval] Episode 2: reward = 500.0
/usr/local/lib/python3.12/dist-packages/imageio/plugins/pillow.py:410: DeprecationWarning: The keyword `fps` is no longer supported. Use `duratio
  warnings.warn(
[eval] Saved bc_cartpole_demo.mp4 and bc_cartpole_demo.gif

BC eval rewards: [500.0, 500.0]
```

5. Output in Terms of Iterations

This section interprets the detailed training logs obtained during execution, particularly the **PPO expert's iteration-by-iteration progress**.

The file shows around **75 iterations**, each corresponding to **2048 environment steps**, producing approximately **150,000 PPO timesteps** overall.

Key Observations from Iteration Logs

Iterations increase steadily from 1 to 75, showing consistent PPO training progression. At each iteration, PPO prints metrics such as:

- approx_kl → how much the new policy diverges from old policy
- clip_fraction → how often the objective is clipped
- entropy_loss → how random the policy is
- value_loss → critic network error
- explained_variance → how well value function predicts returns

Over time:

- **approx_kl decreases** → policy becomes stable
- **entropy_loss decreases** → policy becomes deterministic (typical for CartPole)
- **explained_variance increases** → critic becomes accurate
- **value_loss drops** → critic predictions improve

These results show **convergence** of PPO before collecting demonstrations.

BC Training Output (Iterations as Epochs)

BC training shows:

Epoch	Loss
1	0.08368
10	0.02442
20	0.01648
30	0.01506
40	0.01323
50	0.01148
60	0.01163

This shows:

- Sharp improvement in the first 10 epochs
- Loss plateauing after epoch 40
- Model has **successfully learned** the expert policy

Evaluation Output (Iterations as Episodes)

During evaluation: [500.0, 500.0]

Code Base: Behaviour-Cloning-With-Transformer