Behavior Transformer for Robotic Manipulation: A Two-Stage Behavior Cloning Approach on PushT

Berke Özmen

July 31, 2025

1 Introduction

Robotic manipulation tasks in real-world environments require policies that are both robust and generalizable. A common approach to offline policy learning is Behavior Cloning (BC), where expert demonstrations are directly imitated. However, standard BC methods often suffer from compounding errors and limited expressivity.

This work implements a streamlined version of the *Behavior Transformer* for the PushT-v0 task within the `LeRobot` framework. PushT is a planar pushing benchmark where a 2D agent must learn to move objects using only its position as input—no object state or vision is provided.

The approach frames BC as a two-stage process: classification over discretized action bins using KMeans, followed by regression to refine residuals. This hybrid strategy enables compact yet expressive modeling under strict input constraints. The resulting implementation is modular, reproducible, and respects the challenge requirement to use only state-based data.

This report covers:

- The architecture and rationale behind the Transformer design.
- Expert data generation and action binning methodology.
- Training objectives and evaluation outcomes.
- Core implementation challenges and lessons learned.

The following sections detail the methods, implementation choices, and outcomes observed in applying this framework to the PushT benchmark.

2 Data Collection and Preprocessing

2.1 Expert Trajectory Generation

PushT-v0 simulates a planar pushing task where an agent moves a puck to a goal using continuous 2D actions. Expert demonstrations were generated using `get_expert_action()` and recorded for 500 episodes (max 300 steps each) using fixed seeds. At each timestep, the 2D agent position $\mathbf{s}_t \in \mathbb{R}^2$ and expert action $\mathbf{a}_t \in \mathbb{R}^2$ were stored as NumPy arrays of shape $(N, 2)$.

2.2 Action Discretization via KMeans

To enable coarse classification, actions were clustered into $K = 32$ bins via KMeans. Each action was decomposed as:

$$\mathbf{a}_t = \mathbf{c}_{k_t} + \mathbf{r}_t \tag{1}$$

where $\mathbf{c}_{k_t}$ is the nearest centroid and $\mathbf{r}_t$ is the residual. This yielded aligned arrays for state, bin label, and correction term.

Centroids $\{\mathbf{c}_k\}_{k=1}^{32}$ were saved as a `torch.Tensor` for inference. This hybrid structure improved data efficiency and aligned with Transformer decoding.

2.3 Dataset Construction

Approximately 60,000 transitions were saved in `.npz` format. Training batches consisted of $(\mathbf{s}_t, k_t, \mathbf{r}_t)$ tuples. No data augmentation or normalization was used to maintain fidelity to expert trajectories.

3 Results and Analysis

The trained policy was evaluated in PushT-v0 with up to 300 steps per episode. Only the 2D agent position was used during training and inference.

3.1 Training Metrics

Loss curves converged within 10 epochs. Classification loss dropped from 0.96 to 0.18, and regression loss from 0.033 to 0.0055:

Epoch	Classification Loss	Regression Loss
1	0.960	0.033
5	0.222	0.014
10	0.184	0.0055

3.2 Qualitative Evaluation

A rendered rollout with the final policy achieved a reward of **0.0**, indicating task failure. Despite locally coherent motion, the agent failed to reach the goal—likely due to missing object or goal input, limiting spatial reasoning.

3.3 Experiments and Observations

Key experiments included:

- **Discretization Sensitivity:** 32 clusters balanced expressivity and classification accuracy.
- **Model Architecture:** A shallow Transformer performed well on the 2D space. Deeper variants showed overfitting.
- **Synthetic Data:** Early synthetic rollouts were unstable. Real expert trajectories improved quality.
- **Residual Clamping:** Unbounded residuals led to instability; using centroids maintained control stability.

3.4 Failure Analysis

The policy lacked object/goal state inputs, limiting task-level planning. Its performance suggests overfitting to short-term behavior. Future work could explore longer sequences or auxiliary inputs to improve global reasoning.

4 Challenges and Lessons Learned

Several implementation challenges were encountered throughout the development process, primarily stemming from interfacing with the LeRobot framework and managing data integrity.

Observation Mode Compatibility: The environment defaults to high-dimensional observations, whereas this implementation relies solely on 2D agent positions. Explicit configuration of `obs_type="pixels_agent_pos"` resolved initial mismatches.

Expert Data Integrity: Issues arose due to malformed or incomplete trajectory files. Manual inspection and shape verification ensured data consistency.

Synthetic vs. Real Expert Behavior: Early experiments with synthetic expert policies proved unstable. Switching to the built-in expert improved trajectory quality and learning stability.

Discretization Tradeoffs: Varying the number of KMeans clusters highlighted a balance between classification resolution and residual error. Thirty-two clusters provided an optimal tradeoff.

Rollout Debugging: Closed-loop evaluations uncovered subtle issues in action formatting and tensor transfers, reinforcing the importance of visualization and strict data shape control.

Framework Integration: Dependency conflicts were mitigated by isolating the environment in a pinned Conda setup. Modular scripts improved reproducibility and clarity.

These challenges deepened understanding of practical constraints in imitation learning pipelines and emphasized the importance of robust preprocessing and system isolation.

5 Future Work and Improvements

Several extensions are proposed to enhance performance and generalization:

Richer Input Signals: Incorporating full D observations: such as object and goal positions—would improve task-awareness and planning capabilities.

Normalization: Standardizing input features and regression targets could enhance convergence speed and training stability.

Model Capacity: While a shallow Transformer sufficed, deeper architectures with dropout and GELU activations could better model temporal dependencies if regularized appropriately.

Curriculum Learning: Gradually increasing rollout length may reduce compounding errors and improve long-horizon performance.

Vision-Based Extensions: Incorporating image observations via CNN or ConvTransformer modules would align with the full capabilities of the original Behavior Transformer design.

Comparative Evaluation: Future work should include baseline comparisons (e.g., MLPs, RNNs) and ablations on discretization and architecture. Visualizing attention and clustering policy rollouts could offer further behavioral insight.

These directions provide a roadmap for scaling this approach toward more complex, multimodal robotic manipulation tasks.

Bonus: Ablation Study

To evaluate the role of model architecture, a baseline using a multi-layer perceptron (MLP) was trained on the same behavior cloning task. The MLP shared the Transformer's hybrid classification-regression setup but used a simpler feedforward structure.

The MLP achieved:

- **Classification Loss:** 0.226
- **Regression Loss:** 0.0071
- **Reward (Rollout):** 0.0

While the MLP captured short-term movement patterns, it lacked global coherence. The Transformer outperformed it in terms of loss and qualitative behavior, suggesting its advantage in temporal representation. However, both models were limited by the 2D-only input, highlighting that input richness is a more critical factor than architecture alone.

Note: Complete training curves and evaluation visuals are provided in the GitHub repository.

Acknowledgments

I would like to thank the sereact.ai team and sereact GmbH for their interest and for organizing this challenge. It was a great opportunity to explore behavior cloning in a robotics context. I learned a lot throughout the process, and I appreciate the chance to work on such a well-designed and challenging task.