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 necessitate policies that are robust, efficient, and generalizable. A widely adopted approach for policy learning from offline data is Behavior Cloning (BC), in which expert trajectories are directly mimicked without the use of reinforcement signals. Nevertheless, standard BC methods often struggle with generalization due to compounding errors and limited expressivity in capturing complex action distributions.

In this work, a simplified yet effective version of the *Behavior Transformer* is implemented for the PushT-v0 benchmark task within the `LeRobot` framework. PushT is a planar robotic pushing environment where a 2D agent must learn to execute control actions that manipulate objects within the scene. The objective is to train a Transformer-based policy that imitates expert behavior using only the agent's 2D positional observations, without incorporating vision or object state information.

The proposed method reformulates behavior cloning as a two-stage prediction task: a coarse classification over discretized action bins, followed by a regression stage to learn fine-grained residuals. This hybrid modeling strategy balances expressivity and computational efficiency, enabling effective imitation with minimal input requirements. The implementation is modular, fully reproducible, and adheres to the challenge constraint of using only state-based inputs, excluding full 5D observations.

The remainder of this report presents:

- The architectural design and motivation behind the Transformer-based policy.
- The procedure for dataset generation from expert trajectories and the applied discretization strategies.
- Training objectives, evaluation methodology, and performance metrics.
- Key technical challenges encountered and insights gained throughout development.

Subsequent sections elaborate on the methodological decisions, practical implementation details, and the insights derived during experimentation with the PushT environment.

2 Data Collection and Preprocessing

2.1 Expert Trajectory Generation

The PushT-v0 environment simulates a planar pushing task in which an agent is required to move a puck to a designated goal location using continuous 2D actions. To generate supervision data for behavior cloning, expert actions were obtained using the environment's built-in `get_expert_action()` method at each time step.

A total of 500 episodes were collected using fixed seeds to ensure reproducibility. Each episode was executed until either successful task completion or a time limit of 300 steps was reached. At each time step, the following data were recorded:

- The 2D agent position $\mathbf{s}_t \in \mathbb{R}^2$
- The corresponding expert action $\mathbf{a}_t \in \mathbb{R}^2$

The resulting data were stored as NumPy arrays with shape $(N, 2)$, where N denotes the total number of collected transitions across all episodes.

2.2 Action Discretization via KMeans

To enable coarse classification of actions, the continuous action space was discretized using KMeans clustering with $K = 32$ clusters. Each expert action $\mathbf{a}_t$ was decomposed as:

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

where $\mathbf{c}_{k_t}$ represents the nearest centroid from the learned codebook and $\mathbf{r}_t$ is the residual vector.

This decomposition yielded three aligned datasets:

- `obs` $\rightarrow \mathbf{s}_t$ (agent position)
- `bin_idx` $\rightarrow k_t$ (coarse bin index)
- `residual` $\rightarrow \mathbf{r}_t$ (residual vector)

The centroid vectors $\{\mathbf{c}_k\}_{k=1}^{32}$ were stored as a `torch.Tensor` and reused during both the training and inference stages. This hybrid coarse-fine representation improved data efficiency and aligned well with the architecture of transformer-based decoders.

2.3 Dataset Construction

The final dataset contained approximately 60,000 transitions and was saved in `.npz` format. During training, each sample was retrieved as a tuple $(\mathbf{s}_t, k_t, \mathbf{r}_t)$ and organized into mini-batches for stochastic optimization. No data augmentation or normalization was applied in order to preserve fidelity to the expert distribution.

3 Results and Analysis

The trained policy was evaluated within the PushT-v0 environment, with a maximum episode length of 300 steps. The observation mode was set to `pixels_agent_pos`; however, only the 2D agent position was utilized during both training and inference.

3.1 Training Metrics

Training proceeded for 10 epochs, during which both classification and regression losses showed rapid convergence. Specifically, the cross-entropy classification loss decreased from approximately 0.96 to 0.18, while the mean squared error (MSE) regression loss dropped from 0.033 to 0.0055. The evolution of these metrics is summarized below:

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

These results indicate that the model successfully learned both the discrete action structure and the residual refinement necessary for accurate control in the continuous domain.

3.2 Qualitative Evaluation

A qualitative assessment was conducted by rendering a rollout of the final policy. The episode achieved a total reward of **0.0**, indicating that the task was not successfully completed.

Visual inspection revealed that the policy produced coherent local movements, yet failed to guide the puck toward the target location. Although low-level motion was replicated effectively, the absence of object state and goal information likely hindered the agent's capacity to perform task-level planning and spatial reasoning.

3.3 Experiments and Observations

Several experimental variations and design evaluations were conducted during development:

- **Discretization Sensitivity:** Different values for K in KMeans clustering (ranging from 16 to 64) were tested. A value of $K = 32$ provided a favorable tradeoff between classification performance and residual accuracy.
- **Model Architecture:** A lightweight Transformer architecture (1 encoder layer, 2 heads) was found to be sufficient for the low-dimensional state space. Deeper architectures exhibited overfitting without notable improvements in performance.
- **Synthetic Data Limitations:** Early experiments using synthetic demonstrations yielded noisy and unrealistic transitions. Superior results were obtained after switching to expert trajectories from the `LeRobot` framework.
- **Residual Stability:** Initial training runs exhibited unstable residual outputs, leading to erratic final actions. While clamping and soft-thresholding strategies were considered, stability was ultimately improved by relying more heavily on centroid-based discretization.

3.4 Failure Analysis

The most significant limitation of the current approach was the absence of goal and object positional information in the input. Since the policy was trained solely on the agent's own position, it lacked the contextual awareness required to evaluate progress toward the task objective. This restriction impaired high-level reasoning and long-horizon behavior.

Additionally, although the hybrid classification-regression structure captured fine-grained movement patterns effectively, the policy appeared to overfit to short-term imitation. The lack of sequence-level objectives or trajectory consistency may have contributed to the policy's inability to generalize across longer episodes or more complex spatial relations.

4 Challenges and Lessons Learned

The implementation process presented a range of non-trivial technical challenges, from data handling issues to architectural tradeoffs. This section summarizes the most impactful obstacles encountered and the corresponding lessons learned throughout the development cycle.

4.1 Incompatible Observation Modes

The `LeRobot` framework supports multiple observation configurations, including visual inputs and full 5D state information. To comply with the challenge constraints and maintain architectural simplicity, the observation space was restricted to 2D agent positions. This decision initially led to API mismatches and runtime errors, particularly in environments defaulting to richer observation modes. The issue was resolved by explicitly configuring the environment using `obs_type="pixels_agent_pos"` and selecting only the first two dimensions from the observation vector during policy inference.

4.2 Corrupted or Incomplete Expert Data

During dataset generation, several corrupted or partially serialized expert trajectory files were encountered. These issues, often arising from improper unpacking of `.npz` or `.npy` archives, typically manifested as shape mismatches or runtime exceptions during training. To mitigate this, data integrity was verified through explicit dimensionality checks, array content validation, and episode-wise consistency testing prior to training.

4.3 Expert Policy Limitations

Initial attempts to generate training data using handcrafted or rule-based policies resulted in noisy and inconsistent demonstrations. These synthetic agents often failed to reflect meaningful task dynamics, degrading downstream performance. A significant improvement in policy stability and learning speed was observed after switching to the built-in expert policy provided by `LeRobot`, which yielded smoother and more task-relevant trajectories.

4.4 Action Discretization Tradeoffs

Discretizing the action space using KMeans clustering introduced a tradeoff between expressivity and classification reliability. Lower cluster counts resulted in coarse control and reduced task accuracy, whereas higher counts reduced classification accuracy and introduced instability during regression. Empirical testing indicated that $K = 32$ achieved the best balance for this task, particularly when paired with a residual prediction head to model fine-scale variations.

4.5 Debugging Policy Rollouts

Several challenges emerged during closed-loop policy evaluation, including incorrect tensor shapes, device mismatches, and improperly formatted outputs. These issues were not immediately visible through numerical logs, making visual inspection of rollouts critical. The use of video rendering during rollout evaluation proved essential for identifying subtle behavioral errors and verifying end-to-end consistency across the data and model pipeline.

4.6 Framework Integration and Dependency Management

Incorporating the `LeRobot` framework into the training and evaluation pipelines introduced versioning and compatibility issues, particularly when switching between different branches or dependency configurations. These challenges were mitigated by isolating the project in a dedicated Conda environment, pinning all package versions, and modularizing the scripts for trajectory collection, training, and evaluation. This setup significantly improved reproducibility and reduced debugging overhead.

Overall, these challenges underscored the importance of robust data handling, minimal input design, and controlled experimentation when developing imitation learning systems in robotic contexts. Each obstacle contributed to a deeper understanding of both the strengths and limitations of Transformer-based policies for offline robotic control.

5 Future Work and Improvements

Although the current implementation satisfies the basic requirements of the challenge—training a Transformer-based behavior cloning policy using 2D state observations—multiple opportunities for improvement remain. These directions focus on enhancing performance, generalization capacity, and real-world applicability.

5.1 Higher-Dimensional Input Representation

The use of 2D agent position as the sole input was a deliberate design decision aimed at simplicity and interpretability. However, this limitation restricts the policy's capacity to reason about task context, object states, and relative spatial relationships. Incorporating higher-dimensional state inputs, such as full 5D observations that include object positions and velocities, is expected to provide richer representations and enable more effective planning behaviors. Transformer architectures are well suited to process such multimodal inputs.

5.2 Input Normalization and Scaling

The current training pipeline operates directly on raw coordinate values, which may exhibit varying scales across trajectories. This lack of normalization can increase sensitivity to magnitude fluctuations, slow down convergence, and lead to inconsistent residual predictions. Applying standardized preprocessing—such as normalization using dataset-wide statistics or running averages—could reduce gradient variance and improve training stability.

5.3 Architectural Enhancements

The Transformer encoder used in this implementation was intentionally lightweight, consisting of a single encoder layer with two attention heads. Preliminary experiments with deeper architectures showed signs of overfitting, but the potential of larger models remains underexplored. Future iterations may consider incorporating additional layers, dropout regularization, GELU activations, and positional encodings to better capture temporal dependencies and longer-horizon structure in observation sequences.

5.4 Longer Rollouts and Curriculum Training

Training was conducted using episodes of fixed length (300 steps), matching the environment's default setting. Curriculum learning strategies, in which agents are first trained on short-horizon tasks and gradually exposed to longer sequences, may reduce compounding error and improve robustness during extended rollouts. Additionally, temporal weighting techniques that emphasize critical decision points (e.g., early collisions or late-stage adjustments) could be introduced to improve task success rates.

5.5 Vision-Based Learning Extensions

While the current policy operates entirely on state-based observations, the infrastructure is compatible with pixel-based inputs. Extending the architecture to handle image observations, for example by integrating convolutional encoders or pre-trained vision transformers, could broaden the method's applicability to real-world robotic systems. Such a vision-based extension would also align more closely with the original Behavior Transformer framework, which targets sensor-rich environments.

5.6 Comparison to Baselines and Ablation Studies

Due to time constraints, baseline comparisons and ablation studies were not conducted. Future work should evaluate the current architecture against alternative approaches, such as MLPs or RNN-based policies, under identical training and evaluation protocols. Furthermore, ablation studies focusing on the discretization strategy, residual regression, and input resolution could provide deeper insights into model behavior and inform architectural decisions. Visualization techniques—such as attention heatmaps or trajectory clustering—may also reveal interpretable patterns in learned behavior.

In summary, while the current implementation provides a solid foundation for transformer-based behavior cloning in simplified robotic environments, substantial headroom exists for further enhancement. Future work should emphasize richer input modalities, more expressive model architectures, and comprehensive evaluation against standardized baselines.

Bonus: Ablation Study

To evaluate the importance of model architecture, a baseline was implemented using a multi-layer perceptron (MLP) trained on the same two-stage behavior cloning objective. The MLP consisted of two hidden layers and used the same discretized action space and residual formulation.

The MLP achieved the following performance metrics:

- **Classification Loss (final epoch):** 0.226
- **Regression Loss (final epoch):** 0.0071
- **Total Reward (rollout):** 0.0

Qualitative rollouts showed that the MLP policy captured short-term local movement patterns but lacked global coordination or task awareness, similar to the Transformer model under 2D-only input. However, the Transformer consistently achieved lower training loss and more coherent behavior, likely due to its enhanced capacity for temporal representation.

This study suggests that while architectural improvements are beneficial, the primary performance bottleneck lies in the limited input information. Enhancing the input space (e.g., including object or goal states) is likely more impactful than switching architectures in isolation.

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.