UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022 / Imitation Learning, Reinforcement Learning

[¬Re] Reproducibility Study of Behavior Transformers

Anonymous[1, ID]

[1]Anonymous Institution

February 3, 2023

Edited by
(Editor)

Reviewed by
(Reviewer 1)
(Reviewer 2)

Received
03 February 2023

Published
–

DOI
–

A replication of N. M. M. Shafiullah et al. Behavior Transformers: Cloning k modes with one stone. arXiv:2206.11251 [cs]. Oct. 2022.

Reproducibility Summary

Scope of Reproducibility — This work analyses the reproducibility of "Behavior Transformers: Cloning k modes with one stone" [1] by assessing five aspects: *i*) the performance and *ii*) the multi-modality capturing capability, *iii)* the relevance of each component in BeT architecture, *iv* the sensitivity of the model to critical hyperparameters, and *v* the justification of the design choices.

Methodology — The authors provide an implementation in PyTorch [2] for training and evaluating BeT. However, the implementation does not include all the environments mentioned in the paper, all the ablations performed by the authors, or a way to report all the evaluation metrics. Consequently, we extend this implementation by filling in the missing parts to have a full pipeline and support all the experiments performed in this report. We ran training on an NVIDIA GeForce GTX 780 GPU for 276 GPU hours.

Results — Although our work uses the code provided by the authors at its core the numbers we report diverge from the authors'. Therefore, we cannot conclude that the state of reproducibility of Shafiullah et al. [1]'s work is sufficient. Nevertheless, our empirical results do support their claims in a significant subset of the experiments.

What was easy — It was easy to identify the main claims of the paper and the experiments supporting them. It was easy to get the core parts, such as training and obtaining rollouts with the publicly-available implementation provided by the authors.

What was difficult — It was challenging to set up the environment because not all the dependencies were pinned. This created a source of divergence. Not having the evaluation metrics code available also hindered the reproducibility level of the paper. As the model weights were not public and parts of the code were missing for some experiments, it was difficult to assess the source of discrepancies in our numbers.

Communication with original authors — We communicated via email with the authors throughout the project. They provided multiple clarifications that helped us improve the reproducibility level of the paper, however, communication, naturally interrupted by deadlines, was not enough to reach a faithful reproduction.

Copyright © 2023 Anonymous, released under a Creative Commons Attribution 4.0 International license.
Correspondence should be addressed to (lorn.apogee07@icloud.com)
The authors have declared that no competing interests exists.
Code is available at https://github.com/mlrc2022bet/bet-reproduction.

1 Introduction

Reinforcement Learning (RL) [3] has been a successful method for training agents in sequential decision-making tasks [4]. When a reward signal is available to judge the actions performed by the learning agent in the environment, RL can be used to optimize the total rewards obtained by the agent throughout its experience. However, such a reward signal is often hard to design in real-world environments such as autonomous driving or robotic manipulation, where it could be subjective or hard to attribute to individual actions [5]. Behavior Cloning (BC) [6] can be used in cases where a dataset of expert demonstrations is available, instead of the reward function. It consists in training the agent to mimic, in a supervised learning fashion, the actions performed in the expert demonstrations at every state. More precisely, given a dataset of observation and expert action pairs $\mathcal{D} \equiv \{(o_t, a_t)\} \subset \mathcal{O} \times \mathcal{A}$, the aim is to learn a policy $\pi : \mathcal{O} \to \Delta(\mathcal{A})$ that given observations specifies a distribution over the actions to be performed by the agent. A typical approach is to model this policy by choosing a hypothesis class parameterized by $\theta \subset \Theta$ and optimize the likelihood of observed expert actions:

$$\theta^* = \underset{\theta \subset \Theta}{\operatorname{argmax}} \prod_t \pi(a_t | o_t; \theta) \tag{1}$$

Still, this formulation assumes that the demonstrations come from a single Markovian expert demonstrator, which is a strong assumption that may not hold in large real-world settings or may restrict the amount of data available to the learner.

Shafiullah et al. [1] propose Behavior Transformer (BeT) to overcome two key assumptions in the above BC formulation. First, instead of assuming a Markovian policy, they model $\pi(a_t | o_t, o_{t-1}, \ldots, o_{t-h+1}; \theta)$ for the window size h i.e. a policy that conditions on a history of size h. Second, instead of assuming a unimodal action distribution, they learn a mixture of k Gaussians. They leverage the Transformer architecture [7] for such context-based multi-modal prediction tasks, adapting the discrete classes predicted by transformers to continuous actions. With this architecture, the authors claim that:

1. BeT achieves higher performance during online rollouts than the other BC baselines in multi-modal datasets.
2. BeT can correctly capture the multi-modality of the dataset rather than collapsing onto a single mode.

2 Scope of reproducibility

Shafiullah et al. [1] train BeT on expert trajectories exhibiting multiple modes in four different environments of varying complexity. For each environment they compute two types of metrics on the rollouts generated by BeT:

1. *Performance* metrics to measure the performance of a model during the rollouts (e.g. number of completed tasks). This directly supports their 1st claim.
2. *Diversity* metrics to measure the diversity of the rollouts (e.g. entropy of completed tasks). This directly supports their 2nd claim.

To assess the claims above we reproduce the experiments performed in three of the four environments using BeT with hyperparameters specified by the authors, but not reproducing the experiments for the baselines. The authors additionally make claims that are not consistently supported by experiments, or critical design choices not supported by evidence. We provide experiments to assess those, addressing the following questions:

3. Are all the proposed components of the BeT architecture relevant to *both* the performance and diversity of its rollouts?
4. How sensitive is BeT to critical hyperparameters such as k and h?
5. Are the design choices made in the evaluation metrics empirically justified?

3 Methodology

3.1 Datasets

Shafiullah et al. [1] run experiments in four environments of varying complexity. For each, they provide a dataset of trajectories with different modes and an environment to deploy a trained model on and generate rollouts. They are summarized below.
Point mass. Used as a toy dataset to showcase the limitations of different baselines and the representational power of BeT in faithfully capturing the different modes.
CARLA [8]. It features high-dimensional observations; otherwise, it has similar multi-modality complexity to Point mass and lower than the datasets below. We do not use the CARLA self-driving environment [8] due to limitations in computational resources.
Blockpush [9]. It is a multi-modal dataset, whose stochasticity adds extra complexity.
Kitchen [10]. It is the most complex environment here considered due to its large action space and longer trajectories coming from real human demonstrations.
In all of the experiments, we follow the original paper's indication of considering 95% of the dataset as the training set and the remaining 5% as a test set.

3.2 Model description

Behavior Transformer — To model a context-based policy that outputs multi-modal continuous actions with a transformer architecture, Shafiullah et al. [1] proceed as follows. They decompose an action, a, into two components: a *center*, $A_{\lfloor a \rfloor}$, and an *offset*, $\langle a \rangle$, such that $a := A_{\lfloor a \rfloor} + \langle a \rangle$. The action centers are found by running a k-means clustering algorithm on all the actions in the dataset, with k defined *a priori*, giving a set of k action centers $\{A_1, A_2, \ldots, A_k\} \subset A$ fixed for the rest of the process. The offset is consequently uniquely defined by assigning the center of a to the closest action center $\lfloor a \rfloor := \arg\min_i \|a - A_i\|_2$. We use the term "encode" to mean the decomposition of an action into its center and offset and "decode" to its reconstruction (sum) from its components (see Figure 1 (A)).
Considering this setup, the Transformer takes as input a sequence of continuous observations $(o_i, o_{i+1}, \ldots, o_{i+h-1})$ and learns a sequence-to-sequence mapping of each observation to a categorical distribution over the k action centers (i.e., a $1 \times k$ action center probability vector) and k proposed action offsets, one for each action center (shape: $k \times \dim(A)$). The action center probability distribution head is trained with the Focal Loss [11] to the ground truth action centers. The Focal loss is a simple modification to the standard cross entropy loss that enforces more precise modeling of the low-probability classes, helping the model against class imbalance in the dataset. The offset head loss consists of the squared error to the ground truth offset, but only for the one corresponding to the true action center. These two losses are then summed, with the offset rescaled to make both terms of a similar order of magnitude at initialization (see Figure 1 (B)).
After training, BeT interacts with the environment by taking actions according to the following procedure. Given its observation history so far, it first samples an action center, where the probability of picking each action center is given by the action center head output. Then, by adding the sampled action center with the corresponding offset (in the offset head), it recovers (decodes) the action to be performed (see Figure 1 (C)).

3.3 Experimental setup and code

Code — The authors provide a codebase [1] for training and generating rollouts for BeT. However, they do not provide the required scripts to explicitly run the experiments presented in the paper, nor does the code include a way to compute and report the evaluation metrics shown in their tables or run the ablations they considered. Moreover,

the development environment (a `conda` environment) released by the authors was broken due to unpinned dependencies, and platform-specific. Therefore, it was hard to get started with the reproducibility task and obtain the relevant metrics to compare to the paper's numbers and assess its contribution. In what follows we detail our efforts in extending the implementation provided by the authors to provide a complete pipeline supporting all the experiments performed in this report, from a reproducible development environment to reproducible reporting of evaluation metrics.

First, we address the development environment setup. We use Cresset [12], a simple MLOps template designed for research purposes. It supplies a flexible and reproducible development environment that can be used in heterogeneous computing environments. It provides configurable options for OS, CUDA, and PyTorch versions, which are necessary as different computing environments and hardware require different combinations of libraries. It also includes various options, such as providing an easy way to compile PyTorch from source. This feature is necessary on GPUs whose available PyTorch binaries were not compiled on, such as the GPU we use for this reproduction. We adapt Cresset for our use case, pin all versions of our direct and indirect dependencies, and make the generated Docker image available to reproduce our work.

Second, we aggregate the bits of code found in the authors' repository to include the evaluation metrics of every environment experimented with in the reproducibility pipeline. We noticed some discrepancies between the metrics reported in the paper and how they were actually computed. We also investigated critical design choices made in the evaluation metrics (see appendix C). These address point 5 in our reproducibility scope.

Finally, the numbers reported in the paper are all relative to single runs, which could be subject to cherry-picking and high variance. To have a more informed evaluation of BeT we include cross-validation runs in our pipeline. For each cross-validation run, we consider the model which attained the lowest test loss, computed at the end of every epoch (contrarily to the code provided by the authors, which for the single run recorded, considered the model at the end of training, regardless of test loss). In addition, we set up the option to perform a hyperparameter search in the pipeline. This includes either a grid search/sweep or using the *Optuna* [13] library. These changes were supported by a refactoring of the *Hydra* [14] configurations design.

A notable side contribution we make is by vectorizing the rollout generation code, i.e., having several copies of the environment running asynchronously in parallel, all of them consulting the same model, to leverage batching. With this, we achieve a speedup of N× (20 in our case) by running N environments in parallel.

Experiments — We perform all the experiments done with the BeT model in the original paper on the subset of datasets described in section 3.1. These correspond to tables 1 and 2 in the original BeT paper and directly assess the two main claims of the paper (claims 1 and 2 in our reproducibility scope). They include training BeT on all the datasets, rollout out the policies on their respective environments, and computing the relevant metrics. Each environment features a set of performance metrics and diversity metrics. We describe the metrics in detail in appendix C. The results are described in section 4.

Although the authors only reported the relative performance (in terms of reward) of the BeT ablations they considered, we go beyond by carrying out a more granular analysis. We report the full set of *performance* and *diversity* metrics of each environment for all the ablations adressing point 3 in our reproducibility scope. This is straightforward with the pipeline we built. The relevant results are reported in section 4.

Finally, we experiment with LSTM models and find that they produce competitive results with the minGPT-based decoder model on the Point mass datasets. This opens the path for more investigation on whether RNN architectures can achieve competitive performance against small-sized transformer models.

Unlike the single-run numbers reported by the authors, we report averages (with standard deviation) across cross-validation runs. Every command line needed to run any of

the experiments we ran in this work is readily available in our GitHub repository. They are listed in the *reproducibility_scripts* directory and described in the `readme`. In addition, we share a link in our `readme` to download the model weights, rollouts, and logs of all runs we performed. These are also tracked on *Weights & Biases*[1]. This allows the inspection and reproduction of every single result in this report.

3.4 Ablations

In addressing point 3 of our reproducibility scope, we focused on testing BeT architectural design choices (*binning*, *offsets*, *history*, and *Focal Loss*) and representational power (replacing its Transformer trunk with a MLP). These ablations are detailed below[2].
No offsets. The model only predicts action centers and uses them without offset correction at rollout time. This ablation tests the need for learning an offset correction.
No binning. The model has a single action center ($k = 1$). With this, all the predictive power of BeT is then put in the offset head. This ablation tests the fact that the discrete action centers are responsible for the multi-modal modeling capability of BeT.
No history. The model has a window size $h = 1$. It tests how the lack of historical context harms BeT performance, recovering the standard Markovian model in BC.
[new] No Focal Loss. In the original paper, Focal Loss is presented as a relevant component for appropriate learning of the action center Transformer head, however, no empirical evidence was provided by the authors. We test this claim by setting the Focal Loss parameter γ to zero, recovering the standard binary cross entropy loss. This supports point 3 in our reproducibility scope.
Trunk MLP. The model has a MLP (instead of a Trasformer) trunk. This tests the relevance of the architecture in the final performance. We follow the original paper: the MLP has the same number of hidden layers and layer width as the corresponding MinGPT.
[new] Trunk MLP (Opt). We noticed that the proposed MLP trunk ablation is under parameterized when compared to the corresponding MinGPT. In particular, MinGPT had $\approx 2.6\text{e}5$ and $\approx 1.1\text{e}6$ parameters for Blockpush and Kitchen, respectively. The corresponding values for MLP were $\approx 4.8\text{e}4$ and $\approx 9.2\text{e}5$. Therefore, we performed a hyperparameter search using the Optuna [13] framework in a domain where the largest possible model has a similar number of parameters to those of MinGPT. The total number of configurations tried by Optuna was 25, for both environments. The domain explored is exposed in Table 6. From that exploration, we pick the best-performing model (smaller total loss in the validation set).
Furthermore, we also consider an ablation consisting of a LSTM trunk, but we only assess it on the Point mass environment.

3.5 Hyperparameters

In all the experiments, we adopted the hyperparameters proposed in the original paper. This includes BeT and its ablations except for the MLP ablation, for which we performed a hyperparameter optimization described in the definition of the ablation. In addition, we performed two (independent) linear searches on two critical hyperparameters of the proposed approach: the number of action centers, k, and the window size, h. These experiments tested the sensitivity of the BeT architecture to those hyperparameters, assessing point 4 in our reproducibility scope, and are described in detail in section 4.

3.6 Computational requirements

In this work, we used a single NVIDIA GeForce GTX 780 GPU for all the experiments. The runtime required to obtain a full trained model in different environments are 1 minute

[1]We provide a link to a *W&B* project in the readme of our GitHub repository.
[2]Novel ablations not considered by the authors are prefixed **[new]**.

for point-mass, 1 hour for Blockpush, and 1 hour for Kitchen. Considering all experiments, this took 276 GPU hours. Nevertheles, as training runs in the Blockpush and Kitchen could be run in parallel in the used GPU, it amounted to 128h in real time.
We performed the online rollouts on 2 Intel Xeon E5-2680 v3. A full online rollout takes 5 seconds for point-mass, 10 minutes for Blockpush (speedup: 20x), and in 10 minutes for Kitchen (speedup: 20x). Hence, it took 55 CPU hours in total. Again, we divided the total 40 cores evenly, yielding a total of 27.5h of real run time.

4 Results

4.1 Results reproducing original paper

We report the results of our runs with BeT and its ablations. We reuse the logical formatting of the tables used by the authors and for each experiment show BeT and its ablations in the same table. We highlight the numbers matching the authors' and those that do not. We defer any discussion about mismatches to section 5.

Performance — Assessing the authors' 1st claim.

Table 1. Performance of BeT and its ablations on the Blockpush environment. Performance is measured by computing the probabilities of reaching one (R1) and two blocks (R2), and push one (P1) and two blocks (P2) to the respective target squares.

Models	#Runs	R1	R2	P1	P2
BeT	5	**1.00 ± 0.00**	**0.98 ± 0.01**	**0.94 ± 0.02**	0.18 ± 0.01
No offsets	3	**1.00 ± 0.00**	0.96 ± 0.02	0.89 ± 0.09	0.11 ± 0.05
No binning	3	0.99 ± 0.01	0.68 ± 0.07	0.52 ± 0.11	0.04 ± 0.01
No history	3	**1.00 ± 0.00**	**0.98 ± 0.01**	**0.94 ± 0.02**	0.17 ± 0.04
Trunk MLP	3	0.93 ± 0.05	0.49 ± 0.06	0.01 ± 0.00	0.00 ± 0.00
[new] Trunk MLP (Opt)	3	0.95 ± 0.04	0.41 ± 0.15	0.03 ± 0.01	0.00 ± 0.00
[new] No Focal Loss	3	**1.00 ± 0.00**	0.96 ± 0.00	0.91 ± 0.05	**0.20 ± 0.07**
BeT (paper)	1	1.00	0.99	0.96	0.71
Demonstration		1.00	1.00	1.00	1.00

Blockpush (table 1) Regarding BeT, we recover the same numbers on R1, R2, and P1, however, our number for P2 significantly diverges from the author's number. Yet, those numbers still beat the baselines reported by the authors and hence support their 1st claim. Moreover, we show that the variance across different training folds is low. Regarding the ablations, for the no-offsets, no-binning, and no-history ablations we recover numbers coherent with what the authors report. For the MLP ablation, the authors report a relative performance of flat 0 (performance is computed from the reward which only accounts for P1 and P2). With our detailed reporting of the metrics, we can see that nonetheless, this ablated model reaches to the blocks, and that it puts 1 block in a few rollouts. Finally, the focal loss ablation performs on par with BeT, but with a much higher variance. This sheds more light on point 3 of our reproducibility scope.
Kitchen (table 2) Regarding BeT, our numbers significantly diverge from the authors' when it comes to more than one task achieved. Although the numbers are still high, they make some baselines stronger than BeT, thus refuting the 1st claim of the authors. Our numbers for the no-history and MLP trunk ablations match the relative performance reported by the authors. However, the numbers for the no-offset and no-binning ablations do not match, with the no-binning ablation outperforming BeT.
Point mass We show qualitatively in figure 4 that BeT does not collapse.

Diversity — Assessing the author's 2nd claim.
When the performance numbers do not match, the diversity numbers will also not

Table 2. Performance of BeT and its ablations on the Kitchen environment. Performance is measured by computing the probabilities of n tasks being completed during an episode.

Models	# Runs	1	2	3	4	5
BeT	5	0.92 ± 0.06	0.69 ± 0.18	0.47 ± 0.19	0.20 ± 0.01	0.02 ± 0.01
No offsets	3	0.18 ± 0.04	0.03 ± 0.01	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
No binning	3	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.90 \pm 0.12}$	$\mathbf{0.74 \pm 0.19}$	$\mathbf{0.40 \pm 0.16}$	$\mathbf{0.06 \pm 0.03}$
No history	3	0.87 ± 0.02	0.53 ± 0.14	0.32 ± 0.09	0.07 ± 0.02	0.00 ± 0.00
Trunk MLP	3	0.20 ± 0.17	0.03 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
[new] Trunk MLP (Opt)	3	0.19 ± 0.07	0.01 ± 0.01	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
[new] No Focal Loss	3	0.96 ± 0.02	0.77 ± 0.12	0.52 ± 0.14	0.25 ± 0.09	0.02 ± 0.01
BeT (paper)	1	0.99	0.93	0.71	0.44	0.02
Demonstration		1.00	1.00	1.00	0.98	0.00

match, however, we can still compare those numbers relatively to the performance obtained and thus assess claim 2 independently of claim 1.

Blockpush (table 3) Regarding BeT, we recover numbers coherent with the authors': BeT has equal probabilities of reaching to and pushing blocks with different colors. This result supports the authors' 2nd claim. Regarding the ablations, the authors did not provide diversity metrics for those, but we can see from our numbers that the no-binning and MLP ablations do not generate balanced rollouts. Hence supporting the architectural choices of BeT and shedding light on point 3 of our reproducibility scope.

Kitchen (table 3) We describe this in appendix C fundamental sources of divergences between the metrics that we compute and the authors. Regardless of those discrepancies, we can assess whether our numbers support the authors 2nd claim. Regarding BeT, the relative entropy between BeT and the demonstrations is high and comparable to the author's for the task entropies and the sequence entropies, which is a new metric we compute. This supports the authors' claim that BeT capture multi-modality. Regarding the ablations, we observe that ablations with low performance also have low entropy as they perform fewer tasks. The no-history ablation exhibits a significant decrease in the diversity of its rollouts, supporting the author's architectural choices. The no-binning and no-focal loss ablations, however, achieve a higher entropy than BeT.

Point mass We show qualitatively in plots of rollouts in figure 4 that BeT exhibits different modes, though not in the satisfactory way as the authors achieved.

Table 3. Metrics to measure the capability of the models capturing the dataset multi-modality. For blockpush, we measure the relative frequencies of the first block to be reached, those of the different targets to where the red block is pushed, and the different targets to where the green block is pushed. For kitchen, we measure the task and sequence entropy of the set of tasks performed during an episode. These metrics are described in further detail in appendix C.

		Blockpush						Kitchen	
		1st Block Reached		Push: Red Block Target		Push: Green Block Target			
Models	# Runs	Red	Green	Red	Green	Red	Green	Sequence Entropy	Task Entropy
BeT	5	0.51 ± 0.02	0.49 ± 0.02	0.29 ± 0.02	0.28 ± 0.01	0.28 ± 0.02	0.28 ± 0.01	3.98 ± 0.65	2.63 ± 0.07
No offsets	3	0.50 ± 0.01	0.50 ± 0.01	0.26 ± 0.04	0.24 ± 0.04	0.26 ± 0.03	0.25 ± 0.03	1.16 ± 0.23	1.79 ± 0.1
No binning	3	0.54 ± 0.03	0.45 ± 0.04	0.15 ± 0.03	0.13 ± 0.03	0.14 ± 0.02	0.14 ± 0.04	$\mathbf{4.26 \pm 0.61}$	$\mathbf{2.71 \pm 0.04}$
No history	3	0.50 ± 0.02	0.50 ± 0.02	0.29 ± 0.03	0.29 ± 0.02	0.27 ± 0.04	0.27 ± 0.06	3.51 ± 0.43	2.39 ± 0.06
Trunk MLP	3	0.51 ± 0.06	0.42 ± 0.03	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	1.0 ± 0.85	1.59 ± 0.37
Trunk MLP (Opt)	3	0.50 ± 0.03	0.45 ± 0.01	0.01 ± 0.00	0.01 ± 0.01	0.01 ± 0.00	0.01 ± 0.00	0.98 ± 0.27	1.66 ± 0.09
No Focal Loss	3	0.50 ± 0.02	0.50 ± 0.02	0.27 ± 0.02	0.29 ± 0.02	0.27 ± 0.04	0.28 ± 0.06	4.16 ± 0.66	2.67 ± 0.05
BeT (theirs)	1	0.54	0.46	0.43	0.44	0.41	0.40	/	2.47*
Demonstrations		0.50	0.50	0.50	0.50	0.50	0.50	4.69	2.88

Sensitivity to critical hyperparameters — In figure 6 of [1] the authors show a sweep over the number of bins (equivalent to action centers) and claim that the range for near-optimum

performance is quite wide. However, as they use a logarithmic scale over k, the figure does not support this claim. We perform a more granular sweep over both the number of bins k and the window size h independently to first reproduce their sweep results over cross-validation runs and, more importantly, better assess the claim they make about the sensitivity of BeT near high-performing hyperparameters. Results are in figure 3. The results do not match the numbers obtained by the others. However, we can see that there is indeed a sweet spot around high-performing hyperparameters which supports the authors' claims.

4.2 Results beyond original paper

A limitation — A major design choice in the action binning of BeT is that action bins are determined from clusters computed on *all* the actions in the dataset. In particular, this means that all actions at all timesteps "compete" together to give the bin centers. However, the presence of multi-modalities is often restricted to specific timesteps in the rollouts, so there are fewer actions "competing" for action centers. As the idea is to approximate the different modes with these centers, BeT assumes that to clone k different modes, one can use k action bins. For example, in Point mass 1, we deem that $k = 2$ is enough to clone the two modes "up" and "down", appearing at the early timestep when the agents diverge in the up and down directions. When aggregating these up & down actions with the third action going forward to obtain the bin centers, the problem becomes approximating those 3 actions with only 2 centers. With this, it is possible that the K-Means clustering converges to the centers $[0, 0]$ and $[1, 0]$ collapsing the up/-down actions to a single bin and hence collapsing the performance of BeT in the environment. We empirically show this failure mode in figure 5. We believe that considering a timestep-dependent or context-dependent binning can be a valid solution to overcome this problem.

An LSTM baseline — Shafiullah et al. [1] show very poor performance with the LSTM baseline. We run an LSTM on Pointmass 2 and show in figure 4 that it provides performant rollouts capturing the multimodality of the dataset.

5 Discussion

Overall, although our work uses the code provided by the authors at its core, the numbers we report diverge from the authors' in multiple experiments. Therefore, we cannot conclude that the state of reproducibility of Shafiullah et al. [1]'s work is sufficient, despite their efforts in open-sourcing their code. Nevertheless, our numbers, although different from the author's, do support their claims in a subset of the experiments. From the Blockpush experiments, the proposed method proves to exhibit the representational power desired for multi-model behavior cloning. Regarding the ablations, we have observed that the model without offsets presented almost as good performance as the base configuration when the action dimension space is low, such as in Blockpush, but proved to be essential in Kitchen, where the action space is higher dimensional as the action centers become much less expressive. The runs where the model horizon was set to 1 performed well, as they stick to one performant modality, but show a decrease in their diversity metric. For the two MLP ablations, we see that having a MinGPT trunk is extremely relevant, as the MLPs, despite fine-tuned, lack representative power to correctly model the sequence of observations received. Nevertheless, from the Focal Loss, we see that there is no improvement from using it instead of the standard binary cross-entropy. This may require further empirical evidence from the authors.

5.1 Limitations

Our work is limited by sources of discrepancies that we could not mitigate such that a slightly different development environment due to the unreproducible one provided by the authors, relatively different ways of computing evaluation metrics as not provided by the authors, and in one case incorrectly described.

5.2 What was easy

It was very easy to understand the main claims of the paper and which experiments supported them. Once we had the development environment set up, it was easy to train models on the Blockpush and Kitchen datasets, thanks to the authors sharing a link to their dataset and providing the code for the data loaders, model definition, and training. It was also easy to generate rollouts for the trained models with the code they provided defining the environments.

5.3 What was difficult

Although the authors provided a development environment specification via a conda environment configuration file, it did not pin all the dependencies, which were broken at the time we started the reproduction. This made us make our own environment, which already created a source of divergence between our results, and a difficulty in analyzing this divergence. Moreover, it has been very hard to get the first evaluation metrics and be confident that those were the ones computed by the authors as they did not provide the code to do so, nor did they describe the design choices of the metrics in the paper. We had to get those from bits in the codebase and clarifications from the authors. Finally, the paper did not provide model weights or training curves and reported single-run numbers without confidence intervals, hence making it very hard to judge the extent of the discrepancy in our results and investigate its sources.

5.4 Communication with original authors

We communicated with the original authors throughout the reproducibility project. They helped us identify relevant parts in the code that we leveraged to complete our reproducible pipeline. When we got drastically different results for some metrics, they shared python notebooks used to compute their numbers which allowed us to identify discrepancies in how they were computing and reporting them. However, with this back-and-forth communication, we could not take the time to deeply dive into the details that may have caused the discrepancies between our results. Although we greatly appreciate their effort in helping us reproduce their paper, we feel that many of the clarifications they gave should have been documented in the code or the paper. This would have significantly increased the reproducibility level of the paper.

References

1. N. M. M. Shafiullah, Z. J. Cui, A. Altanzaya, and L. Pinto. **Behavior Transformers: Cloning k modes with one stone**. arXiv:2206.11251 [cs]. Oct. 2022.
2. A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: **Advances in Neural Information Processing Systems**. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.
3. R. Sutton and A. Barto. **Reinforcement Learning, second edition: An Introduction**. Adaptive Computation and Machine Learning series. MIT Press, 2018.
4. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning." In: **nature** 518.7540 (2015). Publisher: Nature Publishing Group, pp. 529–533.
5. A. Y. Ng, S. Russell, et al. "Algorithms for inverse reinforcement learning." In: **Icml**. Vol. 1. 2000, p. 2.
6. T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. "An algorithmic perspective on imitation learning." In: **Foundations and Trends® in Robotics** 7.1-2 (2018). Publisher: Now Publishers, Inc., pp. 1–179.
7. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is all you need." In: **Advances in neural information processing systems** 30 (2017).
8. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. **CARLA: An Open Urban Driving Simulator**. arXiv:1711.03938 [cs]. Nov. 2017.
9. P. Florence, C. Lynch, A. Zeng, O. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. **Implicit Behavioral Cloning**. arXiv:2109.00137 [cs]. Aug. 2021.
10. A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. **Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning**. arXiv:1910.11956 [cs, stat]. Oct. 2019.
11. T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. **Focal Loss for Dense Object Detection**. arXiv:1708.02002 [cs]. Feb. 2018.
12. **Cresset: The One Template to Train Them All**. original-date: 2021-09-16T14:22:59Z. Dec. 2022.
13. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. **Optuna: A Next-generation Hyperparameter Optimization Framework**. arXiv:1907.10902 [cs, stat]. July 2019.
14. O. Yadan. **Hydra - A framework for elegantly configuring complex applications**. 2019.

A Model Description

In this section, we present an very helpful diagram to understand the approach proposed by Shafiullah et al. [1] (figure 1).

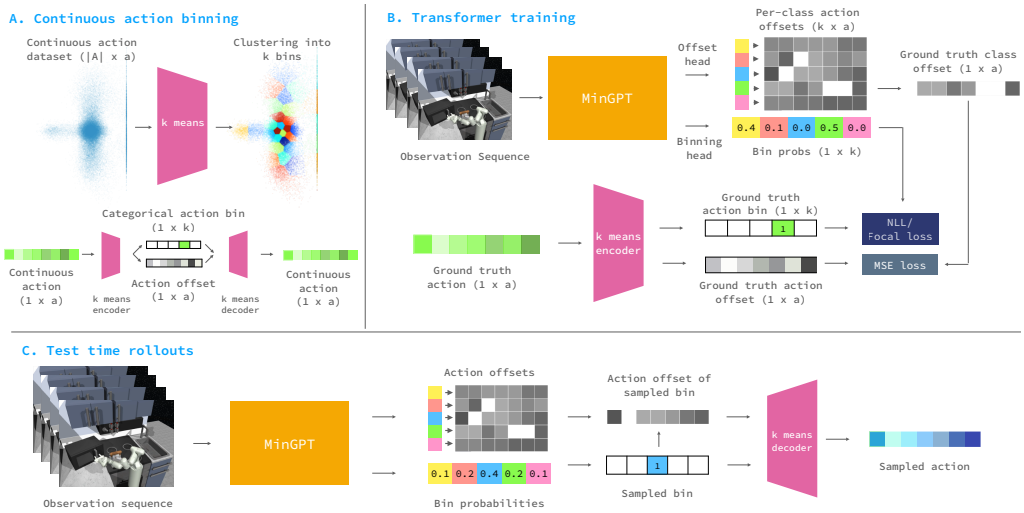

Figure 1. Architecture of BeT pipeline, taken from Shafiullah et al. [1]. (A) Action encoding and decoding process, via clustering. (B) BeT architecture and training. (C) Sampling an action with BeT given observations within the horizon.

B Datasets

A detailed description of the datasets used in Shafiullah et al. [1] is provided below:

Point mass. Point mass is a synthetic toy dataset used to isolate and showcase the benefits of BeT's representational power. I.e. that a history-dependent policy implemented with a transformer does not collapse when trained on a dataset of trajectories with multiple modes and can represent them faithfully in rollouts. Observations and actions in this environment are 2-dimensional (x, y) coordinates. The dataset has 2 versions one with two modes and trajectories of lengths 8 and one with three modes and trajectories of lengths between 8 and 16.
Unfortunately, although this environment was used throughout the paper as a proof of concept, the authors did not provide the dataset of demonstrations they used in this environment. We found bits of code that we glued together to generate the dataset at runtime during training, but some hyperparameters such as the number of demonstrations and noise level do not coincide with what is described in the paper. In particular, the authors do not mention any source of stochasticity in this dataset in the paper, however, their figures demonstrating point mass do show stochastic behavior. This significantly reduces the reproducibility level of the paper.
CARLA. The authors use the CARLA self-driving environment [8] to test BeT's capability in learning from high dimensional observations (224,224,3)-dimensional RGB images. We do not include this dataset in our experiments as we do not have the computational requirements to get rollouts for it[3]. Nevertheless, we believe that as the dataset only contains 2 modes, uses a pre-trained ResNet-18 for observation embeddings, and 2-dimensional actions, excluding it is not detrimental to assessing the paper's main claims.

[3]Ideally an 8-GB GPU in addition to the one used for training. (Requirements reference)

In particular, excluding high-dimensional observations, point mass has a similar complexity, and the following datasets have more complexity.
Blockpush. The multi-modal block-pushing environment [9] features a robotic arm moving two blocks of different colors in two targets of different colors. The observations are 16-dimensional and the actions 2-dimensional. The dataset has 1000 demonstrations generated from a deterministic controller and its multi-modality comes from the different combinations of reaching which color first and putting it in which target. The added complexity of the environment comes from the stochasticity in the start positions of the blocks and the trajectory lengths varying between 85 and 201.
Kitchen. The Franka kitchen environment [10] features human demonstrations recorded via VR headsets in a virtual kitchen environment. They were instructed to perform different sequences of four tasks from a list of seven possible tasks. The observation and actions are 30 and 9-dimensional, respectively. The dataset contains 566 demonstrations in total, with trajectories of lengths between 161 and 409. This is the most complex environment due to its larger action space and longer trajectories coming from human demonstrations which may differ from the simpler synthetic demonstrators in the previous datasets. Although the authors do not mention it, this environment has a stochastic starting position.

For more details, the datasets are comprehensively described in the original paper [1] (see Section 3.1 and appendix, Section A). The authors of the original paper provide a link to download all but the point mass dataset in the readme of their repository.

C Metrics

C.1 Metrics and Threshold Analysis

Franka kitchen environment — In this section, we describe the metrics to analyze performance and diversity in the Franka kitchen environment, from this point on referred to as Kitchen. We also delve into a more granular analysis of a parameter, the **threshold**, which defines the task completion in this environment.

Metric definition The Franka kitchen environment uses an array of observations to track the position of the Franka Emika Panda robotic arm and various household items. At each step, is computed the distance between these observations and a set of predefined goals. If this distance falls below a predefined threshold, the goal is achieved. Therefore, in this environment, we define the following metrics to assess model performance and diversity in Kitchen:

- Reward **[Performance]**: The reward is calculated based on the number of tasks accomplished during each rollout. Longer sequences lead to a higher reward.
- Number of tasks completed **[Performance]**: There are seven tasks in total, the agent has 280 timesteps to complete them. The demonstration dataset has always four tasks completed each rollout, while BeT in a subset of rollouts is able to complete more than four in the allotted time.
- Task entropy **[Diversity]**: Entropy computed on the distribution of achieved tasks during 1000 rollouts. Is the main metrics used by the authors.
- Sequence entropy **[Diversity] [New]**: Entropy computed on the distribution of sequences of achieved tasks during 1000 rollouts. It takes into account the number of unique and chronologically recorded task sequences. As the authors did not differentiate between different permutations of the same achieved tasks when completed in a different order, we propose this additional metric to better measure the model's multimodality.

Discrepancies in metrics computation As the authors kindly provided us with the code used to compute their tables, in this subsection we describe the two main different choices we made regarding metrics computation.

- They determine task entropy by the following steps: sampling uniformly at random 100 task sequences out of the 566 completed in each rollout and computing entropy for the tasks completed in each subset. This process is repeated 5000 times, and the final entropy value is the average of the sum of subset entropies. However, this approach has a stochastic component, and as there was no seed to make the process deterministic each entropy calculation will result in a slightly different value. Since precise values reproducibility was a priority and the random sampling was not considered essential for computing this metric, the decision was made to calculate task entropy directly on all tasks completed in the 566 rollouts, eliminating the element of stochasticity.
- Additionally, we acquire demonstration sequences directly from the training observations in the "bet-data-release" folder provided by the authors, as opposed to the authors' method of obtaining them directly from the ground truth dataset found in the Relay Policy Learning repository by [10]. It was noticed that there is a slight difference between the sequences obtained through these two methods, with 13 rollouts having only 3 completed tasks when obtained from the training observations, while every sequence in the ground truth of 566 has 4 completed tasks.

Threshold analysis An appropriate threshold is a pivotal component to obtaining a correct success metric, as an incorrect threshold can result in significant performance misalignment. In fact, a threshold that is too high for any single task can result in the agent receiving a reward for an incomplete run, even if the obtained configuration would not adhere to the standard expected by a human. Conversely, an excessively low threshold can lead to an unrealistically null success rate.
In the original paper, this value is setup to 0.3 for all the seven tasks. To validate their choice, we computed the achievement rate for 24 different threshold values. For each tentative value, we computed the number of rollouts where each task would be considered completed over the human demonstration dataset. The results are then normalised. As shown in figure 2, a threshold of 0.3 is, indeed, an optimal value as it results in a balanced achievement rate for each task, and it avoids the problem of having single tasks achieved every time.

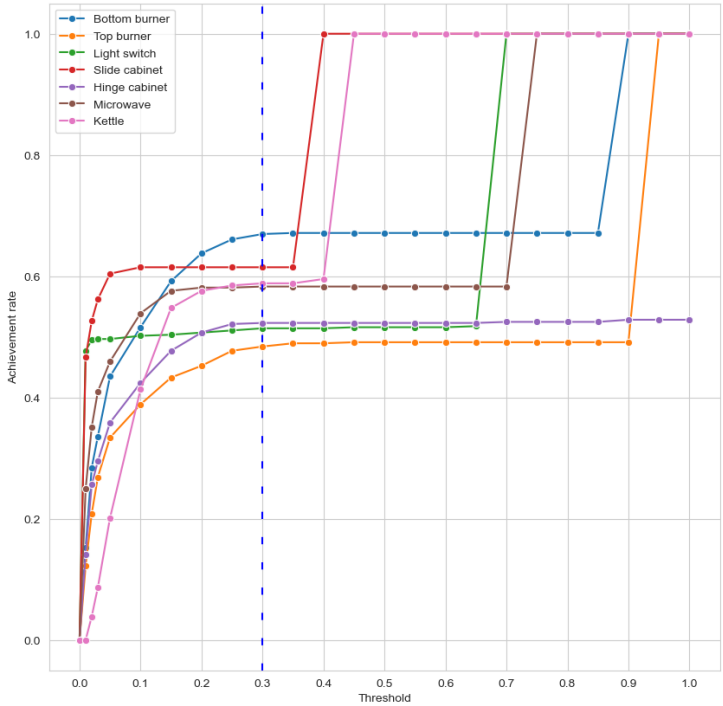

Figure 2. Achievement rate of the different tasks from observations in the demonstration dataset for different threshold values in the Kitchen. The value 0.3 results in a balanced rate for each task.

Blockpush environment – Similarly to Kitchen, we describe the metrics used to analyze performance and diversity of models in the Blockpush environment [9]. Moreover, we also investigate the influence of the **threshold** parameter in the task completion in this environment

Metric definition For the Blockpush tasks, Shafiullah et al. [1] adopted the PyBullet-based environment implemented by Florence et al. [9]. The objective of the XArm robot agent used is to successfully push a block into a target, defined as a square area of the plane. If both blocks are successfully pushed into the two separate targets, the rollout is considered complete and the maximum reward is awarded. In particular, a block is considered correctly pushed if at any moment in time during the rollout the distance between its center and a target falls below a predetermined threshold. Therefore, in this environment, we define the following metrics to assess model performance and diversity in Blockpush, each probability is obtained normalizing the number of occurrences over 1000 rollouts.:

- Reward **[Performance]**: The reward is based on how many blocks are pushed into the target: 0.49 when only one block is recorded as pushed and 1 when both are pushed.
- R1 **[Performance]**: The probability of reaching one block.
- R2 **[Performance]**: The probability of reaching both blocks.
- P1 **[Performance]**: The probability of pushing one block into a target.
- P2 **[Performance]**: The probabilty of pushing both blocks into the two different targets.

- 1st block reached **[Diversity]**: The probability of which block, the red or green, is reached first.
- Push: red block target **[Diversity]**: The probability of determining the target into which the red block is pushed.
- Push: green block target **[Diversity]**: The probability of determining the target into which the green block is pushed.

Proposed metrics To further investigate BeT performance, we implemented additional metrics that differ from the ones of Shafiullah et al. [1]. The authors determine if a block is reached by measuring the distance between the initial block position and its position at each subsequent time step. If this distance exceeds 10^{-3}, the block is considered reached. We decided to recompute this metric by checking the distance between the arm and the block and to determine whether a block is reached if it falls below 0.05, a value already used by the authors to record pushes. This decision is substantiated by the fact that while pushing the first block to a target, the first block can cross and interact with the second block, barely moving it, resulting in what we deem an unintentionally recorded reach of the second block. Our proposed metric seeks to reduce the occurrence of these false positives, therefore having an direct effect on the computed values of R1 and R2. Moreover, by observing the rendered rollouts we noticed that sometimes a block is only momentarily pushed into a target and then pushed out, which would be considered a valid push by the original metrics, even if the intended goal is not properly achieved. A pattern that we observed (and that is considered BeT's primary failure mode by the authors) occurs when the first block is accurately reached but pushed just outside the target; then, the arm moves to push the second block into the target. However, since the rollout does not stop (due to the first block still being misaligned), the arm continues to push the second block further away from the target, following its initial path. In this scenario, both blocks are correctly recorded as reached but the second block is wrongly counted as successfully pushed, even if by the end of the rollout its position is outside the intended target. To avoid these false positives we decided to compute the distances between the blocks and the targets on the last timestep of each rollout and only count as correct pushes the ones where the block actually stays in the target, disregarding the runs where the block is pushed in and then out the target. This directly affects the computation of the values for P1 and P2. The differences in performance between the original way of implementing the metrics and the ones we propose can be visualized in table 4.

Table 4. Comparison of the probabilities of reaching and pushing one or two blocks to the targets, first using the metrics established by the author and then using our proposed metrics. Some values are rounded to 3 decimal cases (contrarily to the rest of the paper) to better illustrate the effect of the different metrics.

	Distance between block positions at t and t+1		Distance between arm and block		Block enters the target		Block stays in the target	
	R1	R2	R1	R2	P1	P2	P1	P2
BeT	1.0	0.976	1.0	0.975	0.940	0.180	0.682	0.002
Demonstrations	1.0	1.0	1.0	1.0	1.0	0.996	0.997	0.983

It is possible to conclude that while the new metric regarding R1 and R2 does not cause a significant deviation, the same does not hold for P1 and P2. In this paper, for the sake of consistency, we considered the same metrics as the original paper. Nevertheless, it is clear that, in some cases, they are not correctly capturing the primary objectives of the considered environment and allowing the aforementioned false positives.

Threshold analysis Similarly to Kitchen, the threshold parameter choice is crucial to compute correct success metrics. Thus, we investigated how the probabilities of push-

ing one and two blocks to respective squares would change when dealing with different threshold values. We observed that Shafiullah et al. [1] used a higher threshold value when compared to the one of the original implementation. The threshold value used by Florence et al. [9] is 0.04, while the authors used 0.05. Naturally, with a higher value, we expected a higher success rate.
This was accomplished by calculating success metrics for the two different thresholds on the 1000 demonstration rollouts from the training dataset and on 1000 evaluation rollouts using BeT with its original hyperparameter configuration. We also tested for other values of threshold and we noticed that: **i)** a threshold ≤ 0.03 is excessively rigorous to the models, being excessively difficult to have success; and **ii)** for a threshold ≥ 0.06, it becomes excessively easy to achieve the goal and this criterion no long is a good success discriminator.
We observed that in the demonstration dataset, the probabilities remain unchanged when computed using either a threshold of 0.04 or 0.05. In only one instance (out of 1000), a block was considered pushed with a tolerance of 0.05 but not with 0.04. During the evaluation rollouts using a BeT agent trained with the original hyperparameters, the differences in probabilities are more pronounced, as shown in table 5. We can see that, even though 0.05 is still a reasonable value for the threshold, it clearly simplifies the problem for the model. Therefore, unless the threshold is the same for both cases, direct comparisons to other models also running in the Blockpush environment should be carefully carried out and interpreted with a grain of salt.

Table 5. Comparison on the metrics proposed by the authors for the threshold values of 0.04, used in the original implementation by Florence et al. [9], and 0.05, used by Shafiullah et al. [1]. The two P2 values for "Demonstrations" are rounded to 3 decimal cases (contrarily to the rest of the paper) to better illustrate the influence of the different threshold values.

	Demonstrations		BeT	
	Threshold 0.05	Threshold 0.04	Threshold 0.05	Threshold 0.04
R1 (block-block distance)	1.00	1.00	1.00	1.00
R2 (block-block distance)	1.00	1.00	0.98	0.98
P1 (block enters the target)	1.00	1.00	0.94	0.86
P2 (block enters the target)	0.996	0.995	0.18	0.11

D Hyperparameter Sweeps

D.1 MLP hyperprameter optimization

In this section, we present the domain explored for the MLP trunk ablation model, using the Optuna framework for hyperparameter optimization. In particular, for each environment, we trained the MLP trunk ablation model for 25 different sets of parameters proposed by the Optuna framework. The explored domain is shown in table 6.

D.2 Sensitivity sweep

In this section, we present the results concerning the sweeps performed on the number of action centers and on the window-sizes (see figure 3).

Table 6. MLP hyperparameter domain explored by the Optuna algorithm. For the learning rate, Optuna sampled values in the continuous interval provided. All the other parameters only could take the discrete values mentioned. The hyperparameters used in the original paper are underlined, except for batchnorm, whose original adopted value is not clear. The best hyperparameter configurations found for both environments are in bold.

Hyperparameters	Blockpush	Kitchen
Learning Rate	1e-5 to 1e-1 (1e-4, **3.05e-4**)	1e-5 to 1e-1 (1e-4, **5.3e-4**)
Gradient Norm Clipping	**None**, 1	**None**, 1
Weight Decay	0.01, **0.05**, 0.1	**0.01**, 0.05, 0.1
Number Hidden Layers	**4**, 6, 8	**6**, 8, 10, 12
Hidden Layers Width	**72** , 128, 144	**120**, 132
Batchnorm	**True**, False	**True**; False

E Point mass figures

In this section, we present the results obtained by running models on the Point mass environment. In particular, we present the rollouts obtained for BeT (hyperparameteres as specified in the original paper [1]) and for the trunk LSTM ablation (manually fine-tuned) in figure 4. Moreover, in figure 5, we present the breaking point of BeT exposed in section 4.2.

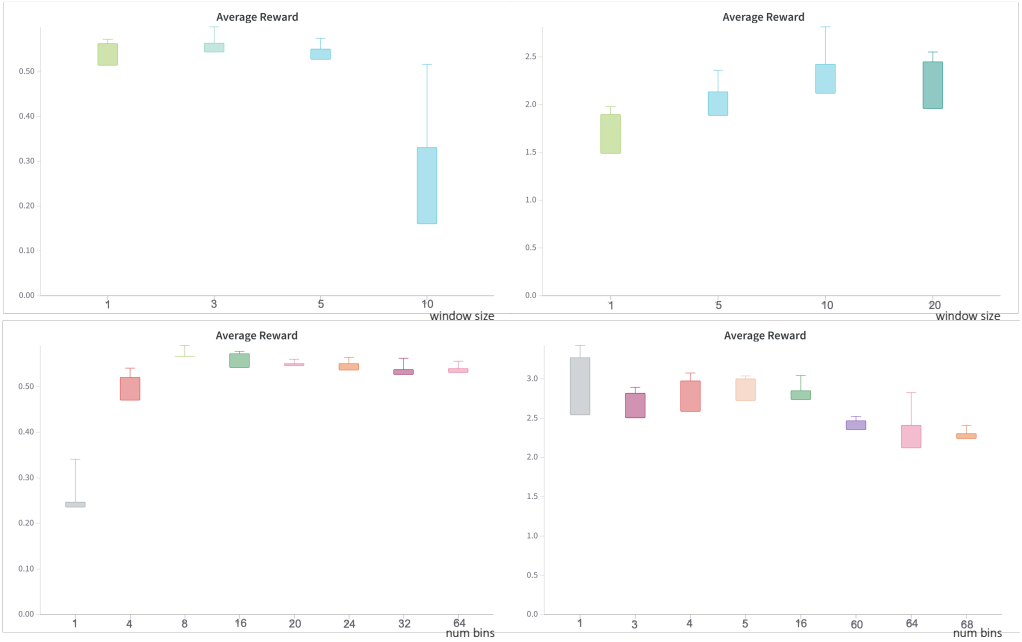

Figure 3. Boxplot for the average reward across rollouts for models swept in the number of action centers (bottom) and of window sizes (top). These sweeps were run both for Kitchen (right) and Blockpush (left). The value of the hyperparameter swept is indicated on the x-axis, while the y-axis corresponds to the average reward.

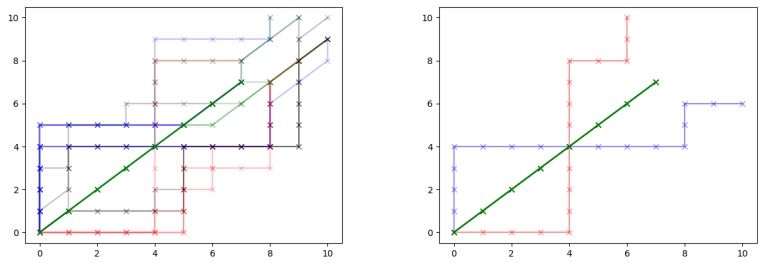

Figure 4. Point mass rollout results with state snapping for models trained on 20,000 training samples without noise. Models used are BeT (left) and an LSTM (right). Hyperparameters for BeT as specified by Shafiullah et al. [1]. Rollouts achieve imperfect performance but have the potential to distinguish between the modes.

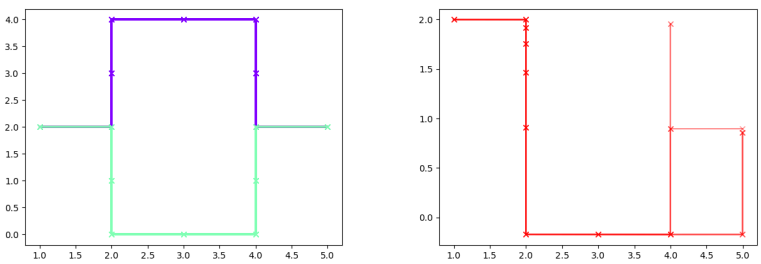

Figure 5. Left: Point mass dataset without noise. There are two modes in the behaviors presented. Right: Rollout exhibiting failure of BeT in a simple scenario. This happens when k-means with $k = 2$, the correct number of modes, converges to vectors that do not allow the model to distinguish the modes.