RESCIENCE C

Replication / ML Reproducibility Challenge 2022

[Re] Reproducibility Study of Behavior Transformers

Skander Moalla[1,3], Manuel Madeira[1,3], Lorenzo Riccio[1,3], and Joonhyung Lee[2,3]

[1]EPFL, Switzerland – [2]Independent Researcher – [3]Equal contributions

Edited by
Koustuv Sinha,
Maurits Bleeker,
Samarth Bhargav

Received
04 February 2023

Published
20 July 2023

DOI
10.5281/zenodo.8173757

Reproducibility Summary

Scope of Reproducibility – In this work, we analyze the reproducibility of "Behavior Transformers: Cloning k modes with one stone" [1]. In assessing the Behavior Transformer (BeT) model, we analyze its ability to generate performant and diverse rollouts when trained on data containing multi-modal behaviors, the relevance of each of its components, and its sensitivity to critical hyperparameters.

Methodology – We use the open-source PyTorch [2] implementation released by the authors to train and sample rollouts for BeT. However, the implementation does not include all the environments, evaluation metrics, or ablations studied in the paper. Consequently, we extend it by following the details in the paper and filling in the missing parts to have a complete pipeline and support all the experiments performed in this report. We conducted our experiments on an NVIDIA GeForce GTX 780 GPU, requiring 276 GPU hours to train our models.

Results – Running the code released by the authors does not produce an evaluation of BeT according to the metrics reported in the paper. After extending the implementation with the proper evaluation metrics, we obtain results that support the main claims of the paper in a significant subset of the experiments but that also diverge in many of the actual values obtained. Therefore, we conclude that the paper is largely replicable but not readily reproducible.

What was easy – It was easy to identify the main claims of the paper and the experiments supporting them. Moreover, thanks to the open-source implementation released by the authors, training the model and sampling rollouts were straightforward tasks.

What was difficult – Setting up the development environment was hard due to dependencies not being pinned. Not having the code for evaluation metrics available hindered our efforts to achieve similar numbers. Assessing the sources of discrepancies in our numbers was also difficult, as training curves and model weights were not accessible.

Communication with original authors – We communicated via email with the authors throughout the project. They provided clarifications and resources that helped us with our study. However, the communication was insufficient to reach a complete reproduction.

Correspondence should be addressed to Skander Moalla (skander.moalla@epfl.ch)
The authors have declared that no competing interests exist.
Code is available at https://github.com/skandermoalla/bet-reproduction – DOI 10.5281/zenodo.7937169. – SWH swh:1:dir:4a562f75c0fd44672b806498e18b67690a5baabd.
Data is available at https://github.com/skandermoalla/bet-reproduction – DOI 10.5281/zenodo.7937169.
Open peer review is available at https://openreview.net/forum?id=E0qO5dI5aEn.

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. BC trains the agent to mimic the actions performed in the expert demonstrations at every state in a supervised learning fashion. 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 $\pi : \mathcal{O} \rightarrow \Delta(\mathcal{A})$ that specifies a distribution over the actions to be performed by the agent for a given observation. A typical approach is to model the policy by choosing a hypothesis class parameterized by some $\theta \subset \Theta$ 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, a strong assumption that may not hold in large real-world settings or which may restrict the amount of data available to the learner.

Shafiullah et al.[1] propose the 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 window size h to learn 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 [7] architecture for this context-based multi-modal prediction task, adapting the discrete classes predicted by transformers to continuous actions. With this architecture, the authors claim that:

1. BeT generates rollouts with higher performance than other BC baselines when trained on multi-modal datasets.
2. BeT captures the multi-modality of these datasets instead of collapsing onto single modes.

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 the model during the rollouts, e.g., the number of completed tasks. This directly supports their first claim (1).
2. *Diversity* metrics to measure the diversity of the rollouts, e.g., the entropy of the completed tasks. This directly supports their second claim (2).

To assess the claims above, we reproduce the experiments performed in three of the four environments using BeT with hyperparameters specified by the authors. We find that, although we cannot produce the authors' results for all experiments, BeT is a generally robust method that achieves the stated goals of the paper.
In addition to reproducing the previous work, we also assess critical design choices and claims, addressing the following questions:

3. Are all proposed components of the BeT architecture relevant to *both* the performance and the 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 environment, they use a dataset of trajectories with different modes and a gym [8] environment to deploy trained models and generate rollouts. We briefly summarize the datasets below and refer the reader to appendix B for more details.
Point mass is a toy dataset to showcase the limitations of different baselines and the representational power of BeT in faithfully capturing the different modes.
CARLA [9] has similar multi-modality complexity to Point mass but features higher-dimensional observations. We do not use the CARLA self-driving environment [9] due to limitations in computational resources and explain our choice in appendix B.
Blockpush [10] is a multi-modal dataset whose stochasticity adds extra complexity.
Kitchen [11] is the most complex environment due to its large action space and longer trajectories coming from real human demonstrations.
The authors provide all the datasets and the implementations of their environments, except for the Point mass dataset and environment. In all experiments, we follow the original paper in using 95% of the dataset for training and the remaining 5% for testing.

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 by first decomposing an action, a, into two components: a *center*, $A_{\lfloor a \rfloor}$, and an *offset*, $\langle a \rangle$, such that $a \coloneqq A_{\lfloor a \rfloor} + \langle a \rangle$. 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 offsets are uniquely defined by assigning the center of a to the closest action center $\lfloor a \rfloor \coloneqq \arg\min_i \|a - A_i\|_2$. 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)).
In 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 $1 \times k$ 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 to the ground truth action centers with the focal loss [12], a simple modification to the standard cross entropy loss that helps model low-probability classes in imbalanced datasets. 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 similar in magnitude at initialization (see Figure 1).
After training, BeT interacts with the environment by first sampling an action center given its observation history so far, 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, neither the scripts required to run the experiments presented in the paper nor those to compute and report the evaluation metrics and run the ablations are available. Moreover, the development environment (a `conda` environment) released by the authors was platform-specific and could not be reproduced due to unpinned dependencies, making it difficult to get started with the reproducibility task and obtain the relevant metrics to verify 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 [13], 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 the 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 that available PyTorch binaries were not compiled for, 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 publicly available for others to reproduce our work.

Second, we aggregate the code in the authors' repository to include the evaluation metrics for every experimental environment in the reproducibility pipeline. We investigated critical design choices made in the evaluation metrics and noticed discrepancies between the metrics reported in the paper and how they were computed in the code. Both of these findings are described in appendix C, addressing point 5 of our reproducibility scope.

Finally, the numbers reported in the paper are all measured for 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 train for the number of epochs specified by the authors and evaluate the model which attained the lowest test loss, computed at the end of every epoch. The authors mention using early stopping for some baselines but do not mention whether they perform such validation for BeT. In addition, we perform a hyperparameter search in the pipeline, which involved either a grid search/sweep or using the *Optuna* [14] library. These changes were supported by refactoring the design of the *Hydra* [15] configurations.

A notable contribution we make is to vectorize the rollout generation code to have several copies of the environment running asynchronously in parallel, all consulting the same model to leverage batching. With this, we achieve a speedup of Nx (20x 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, rolling 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 and describe the results 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 addressing 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 BeT with an LSTM trunk. Whereas Shafiullah et al.[1] reported it having a very low performance, we believe it to be a strong candidate for this context-based multi-modal prediction task, as it has a similar representational power to a Transformer trunk and test it on the Point mass environment.
Unlike the single-run numbers reported by the authors, we report averages (with standard deviation) across cross-validation runs. Every command needed to run any of the experiments is readily available as a shell script 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][16]. 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's architectural design choices (*binning*, *offsets*, *history*, and *focal loss*) and representational power (replacing its Transformer trunk with an 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's performance, recovering the standard Markovian model in BC.
[new] No focal loss. In the original paper, the focal loss is presented as a relevant component for appropriate learning of the action center Transformer head. However, no ablations were conducted in the original work to confirm this. We test this claim by setting the focal loss parameter γ to zero, recovering the standard binary cross entropy loss. This addresses point 3 in our reproducibility scope.
Trunk MLP. The model has an MLP trunk instead of a Transformer. This tests the relevance of the Transformer architecture in the final performance. We follow the original paper in setting the MLP with 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 underparameterized 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 [14] 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 shown in Table 7. From that exploration, we pick the best-performing model with the smallest total loss in the validation set.

3.5 Hyperparameters

We adopt the hyperparameters proposed in the original paper in all the experiments. This includes BeT and its ablations, except for the MLP ablation, for which we performed a hyperparameter search described in the ablation study. 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.

[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]**.

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 trained model in different environments is 1 minute for Point mass, 1 hour for Blockpush, and 1 hour for Kitchen. Considering all experiments, this takes 276 GPU hours. We performed the training runs for Blockpush and Kitchen in parallel on the same GPU, amounting to 128h in wall-clock 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 10 minutes for Kitchen (speedup: 20x). Hence, it takes 55 CPU hours in total. Again, we divided the work into 2 groups of 20 threads, yielding a total of 27.5h of actual run time.

4 Results

4.1 Results reproducing original paper

We report the results of our runs with BeT and its ablations, using the same table format as Shafiullah et al.[1] and show BeT and its ablations in the same table for each experiment. We highlight the differences between our results and the authors' and how they relate to their claims. Discussions of the overall interpretation of the results, the sources of discrepancies, and the global success of the reproduction are deferred to section 5.

Performance — Assessing the authors' first claim (1).

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 pushing one (P1) and two blocks (P2) to the respective target squares. Best in bold.

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 R1, R2, and P1 numbers. However, our P2 results diverge from those of the paper. Yet, those numbers still beat the baselines reported by the authors, supporting their first (1) 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 in terms of relative cumulative reward. For the MLP ablation, the authors report training collapse for P1 and P2, where performance is computed from the reward. With our detailed reporting of the metrics, we can see that this ablated model, nonetheless, reaches the blocks and pushes one block to a target in a few rollouts. Finally, the focal loss ablation performs on par with BeT but with much higher variance. This is consistent with its desired effects, and addresses point 3 in our reproducibility scope.
Kitchen (table 2) Our results diverge from the authors' when BeT is evaluated on achieving more than one task. Although the performance is still good, they make some baselines reported by the authors stronger than BeT, thus refuting their first (1) claim. Our

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 task	2 tasks	3 tasks	4 tasks	5 tasks
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

no-history and MLP trunk ablations results 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. Moreover, we can observe that the focal loss performs on par with BeT and even outperforms it on its best random seed.
Point mass We show qualitatively in figure 5 that BeT produces trajectories with different modes without collapsing, supporting the authors' first (1) claim.

Diversity — Assessing the authors' second (2) claim. When the performance results diverge, so will the diversity results. However, we can still compare those numbers relative to the performance obtained and thus assess claim 2 independently of claim 1.
Blockpush (table 3) In our experiments for BeT, we obtain results consistent with the authors'. BeT has equal probabilities of reaching and pushing blocks with different colors. This result supports the authors' second (2) claim. The authors did not provide diversity metrics for the ablations, but we can see from our numbers that the no-binning and MLP ablations do not generate balanced rollouts, supporting the architectural choices of BeT and addressing point 3 of our reproducibility scope.
Kitchen (table 4) In appendix C, we describe sources of divergence between our metrics and the authors'. Our results support the authors' claim 2 that BeT captures multi-modality. The relative entropy between BeT and the demonstrations is high and comparable to the authors' for the task entropies, as are the sequence entropies, a new metric that we compute. We also observe that ablations with low performance have low entropy as they achieve fewer tasks. The no-history ablation exhibits a significant decrease in the diversity of its rollouts, supporting the authors' architectural choices and addressing claim 3 of our reproducibility scope. The no-binning and no-focal loss ablations, however, achieve a higher entropy than BeT, but this cannot be disentangled from the fact that they also perform better than BeT. We can only say that these ablations are not detrimental to BeT's ability to capture multi-modality on the Kitchen dataset.
Point mass We show qualitative rollout plots in figure 5, showing that BeT exhibits different modes, though not exactly as reported by the authors.

Sensitivity to critical hyperparameters — In Figure 6 of Shafiullah et al.[1], the authors show a sweep over the number of action centers and claim that the range for near-optimum performance is quite wide. However, the figure does not fully support this claim as they use a logarithmic scale over k. 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 claims concerning the sensitivity of BeT near high-performing hyperparameters. Results are in figure 4. The results do not match the numbers obtained by the Shafiullah et al.[1]. However, we can see that there is indeed a sweet spot around high-performing hyperparameters, supporting the authors' claims.

Table 3. Metrics to measure the ability of the models to capture the different modes in the datasets. 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. These metrics are described in further detail in appendix C.

		1st Block Reached		Push: Red Block Target		Push: Green Block Target	
Models	# Runs	Red	Green	Red	Green	Red	Green
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
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
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
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
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
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
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
BeT (paper)	1	0.54	0.46	0.43	0.44	0.41	0.40
Demonstrations		0.50	0.50	0.50	0.50	0.50	0.50

Table 4. Metrics to measure the ability of the models to capture the different modes in the datasets. For Kitchen, we measure the task and sequence entropies of the distribution of tasks performed during an episode. These metrics are also described in further detail in appendix C.

	Sequence Entropy	Task Entropy
Models		
BeT	3.98 ± 0.65	2.63 ± 0.07
No offsets	1.16 ± 0.23	1.79 ± 0.10
No binning	**4.26 ± 0.61**	**2.71 ± 0.04**
No history	3.51 ± 0.43	2.39 ± 0.06
Trunk MLP	1.00 ± 0.85	1.59 ± 0.37
Trunk MLP (Opt)	0.98 ± 0.27	1.66 ± 0.09
No focal loss	4.16 ± 0.66	2.67 ± 0.05
BeT (paper)	/	2.47*
Demonstrations	4.69	2.88

4.2 Results beyond the original paper

A limitation in BeT's representation power — 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" to give the bin centers. However, a model only needs to distinguish between the different modes at the specific state it is considering, so there are actually fewer actions "competing" for action centers at a given output step. As the idea is to capture the different modes with these centers, BeT assumes that to clone k different modes, one can use k action bins, but this may be insufficient. 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, those 3 actions must be captured with only 2 centers. In this case, 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 case in figure 6. We believe that considering a state-dependent, or perhaps timestep-dependent, binning can be a valid solution to overcome this problem.

An LSTM trunk — Shafiullah et al.[1] report a surprisingly poor performance with an LSTM trunk on the Blockpush and Kitchen environments. Although such recurrent neural network architectures are known to be slower and harder to train than a Transformer, we are interested in showing that an LSTM can be a strong candidate for the context-based

multi-modal prediction task BeT is addressing, as both models have similar representational power. We run BeT with an LSTM trunk on the simple Point mass 2 environment to avoid issues related to training and show in figure 5 that it provides performant rollouts capturing the multi-modality of the dataset. The hyperparameters of the model were manually tuned and are listed in Table 8. The performance of the LSTM is actually better than that of the Transformer using the hyperparameters provided by the authors in that environment, which suggests that practitioners with relatively easy environments may also consider using an LSTM trunk.

5 Discussion

In the following section, we use the terms *reproduction* and *replication* as defined in ReScience C to evaluate the reproducibility of the BeT paper. In short, we use

- *reproduction* to mean "running the same computation on the same input data, and then checking if the results are the same... Reproduction can be considered as software testing at the level of a complete study... [It] verifies that a computation was recorded with enough detail that it can be analyzed later or by someone else".
- *replication* to mean "repeating a published protocol, respecting its spirit and intentions but varying the technical details [...] that everyone believes shouldn't matter, and see if the scientific conclusions are affected or not".

Overall, we conclude that the work presented in "Behavior Transformers: Cloning k modes with one stone" is largely replicable but is not readily reproducible.
Indeed, although the authors open source their code and data and describe how to use their model, they do not provide a reproducible way to obtain the numbers reported in the paper. Using their code is not enough, and extending it according to the specification in the paper yields numbers that diverge from theirs in multiple experiments, hence our conclusion on the reproducibility of the paper.
However, most of our experiments support the paper's main claims. From the Blockpush and Kitchen experiments, the proposed method exhibits the representational power desired for multi-model behavior cloning. Indeed, our Blockpush results beat all baselines reported by the authors.
Regarding the ablations, we have observed that the model without offsets performed almost as well as the base configuration when the action dimension space is low, such as in Blockpush. However, offsets proved essential in Kitchen, where the action space is higher dimensional, and the action centers are much less expressive. For the no-binning ablation where only a single bin is used, we obtain precisely the inverse situation: the model underperforms in Blockpush but is the best model for Kitchen, both in performance and diversity. We conjecture that this behavior is due to the model overfitting to a single mode and thus being less prone to confound different modes and go out-of-distribution. The runs where the model was given no history performed well in Blockpush, as they stick to one performant modality, but less well in Kitchen when history is needed to stick to one task. They also show a decrease in diversity. For the two MLP ablations, we see that having a MinGPT trunk is critical as the MLPs fail to exhibit the representation power to correctly model the sequence of received observations despite being fine-tuned. Finally, we found that the focal loss decreased the variance of the reported metrics, confirming the authors' claim that it would help with unbalanced splits, but at the same time, the model with no focal loss outperformed BeT in some runs. Practitioners must, therefore, carefully consider the tradeoff between variance reduction and performance.

5.1 Limitations

Our replication is limited by sources of discrepancies that we could not mitigate, such as a slightly different development environment due to the unreproducible environment provided by the authors, different ways of computing evaluation metrics not provided by the authors, and in one case, incorrectly described, and limited computational budget, which limited our ability to investigate more experiments.

5.2 What was easy

It was very easy to identify and understand the main claims of the paper and the experiments which 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 datasets and providing the code for the data loaders, model definition, and training routine. Generating rollouts for the trained models with the provided code was also easy.

5.3 What was difficult

The authors provide a development environment specification via a conda environment configuration file but only pin the direct dependencies. However, these dependencies have sub-dependencies whose versions were unpinned and made the environment unusable when we started the reproduction, with some libraries crashing at runtime. This in itself made the computations made in the paper non-reproducible. Building our own environment created an initial source of divergence between our methods and made it difficult to isolate this divergence. Moreover, it was 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 exact formulas or the design choices of the metrics in the paper. We had to get those from hints in the codebase and clarifications from the authors, making the replication much more time-consuming. Finally, the paper did not provide model weights or training curves and reported single-run numbers without error bars, making it very hard to judge the extent of the discrepancy in our results and investigate its sources. As computations in machine learning are subject to randomness and non-determinism, it is crucial to report evaluation metrics on multiple runs with a central tendency (e.g. mean) & variations (e.g. error bars) [17] to hope for similar numbers during a reproducibility test.

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 the Jupyter notebooks they used to compute their numbers which allowed us to identify discrepancies between how they were computing and reporting them. We greatly appreciate their effort in helping us reproduce their paper. However, we feel that many of their clarifications should have been documented in the code or the paper.

5.5 Acknowledgements

We thank the teaching team of the EPFL Machine Learning course (CS-433) for encouraging us to conduct this project and for their review of a preliminary version of this work. We thank Prof. Tanja Käser for providing CPUs to run the evaluation. We thank Nur Muhammad (Mahi) Shafiullah for the multiple exchanges about details in the BeT project.

References

1. N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto. "Behavior Transformers: Cloning k modes with one stone." In: **Advances in Neural Information Processing Systems**. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 22955–22968. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/90d17e882adbdda42349db6f50123817-Paper-Conference.pdf.
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. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.
3. R. Sutton and A. Barto. **Reinforcement Learning, second edition: An Introduction**. Adaptive Computation and Machine Learning series. MIT Press, 2018. URL: https://books.google.ch/books?id=uWV0DwAAQBAJ.
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), 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. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. "Openai gym." In: **arXiv preprint arXiv:1606.01540** (2016).
9. A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. "CARLA: An open urban driving simulator." In: **Conference on robot learning**. PMLR. 2017, pp. 1–16.
10. P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. "Implicit behavioral cloning." In: **Conference on Robot Learning**. PMLR. 2022, pp. 158–168.
11. A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning." In: **arXiv preprint arXiv:1910.11956** (2019).
12. T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. "Focal Loss for Dense Object Detection." In: **2017 IEEE International Conference on Computer Vision (ICCV)**. 2017, pp. 2999–3007. DOI: 10.1109/ICCV.2017.324.
13. J. Lee. **Cresset: The One Template to Train Them All**. URL: https://github.com/cresset-template/cresset.
14. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. "Optuna: A next-generation hyperparameter optimization framework." In: **Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining**. 2019, pp. 2623–2631.
15. O. Yadan. **Hydra - A framework for elegantly configuring complex applications**. Github. 2019. URL: https://github.com/facebookresearch/hydra.
16. L. Biewald. **Experiment Tracking with Weights and Biases**. Software available from wandb.com. 2020. URL: https://www.wandb.com/.
17. J. Pineau et al. "The machine learning reproducibility checklist." In: **McGill** (2020).

A Model Description

In figure 1, we present a diagram to visualize the approach proposed by Shafiullah et al.[1].

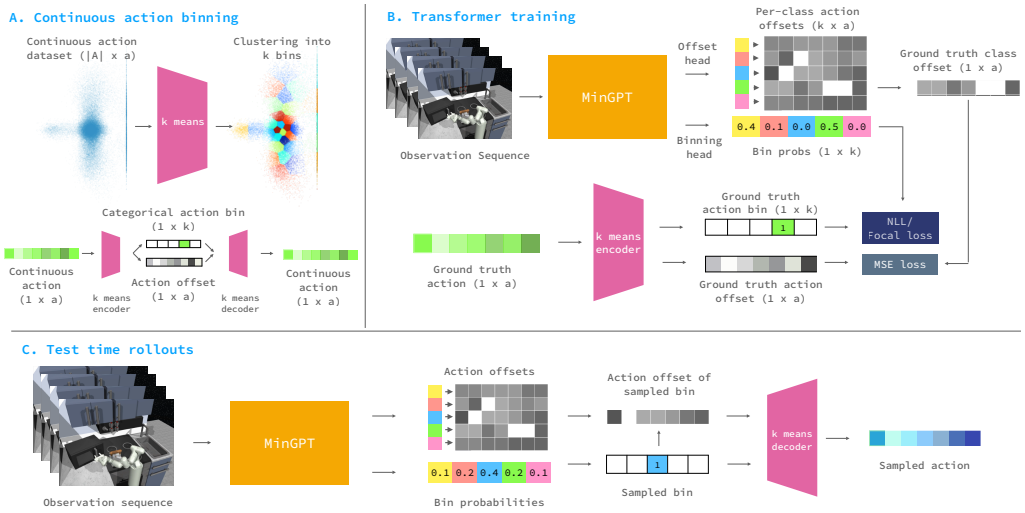

Figure 1. Architecture of the BeT pipeline, taken from Shafiullah et al.[1]. (A) Action encoding and decoding process via clustering. (B) The BeT architecture and training. (C) Sampling an action with BeT, given observations within its history.

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 demonstrate the benefits of BeT's representational power by showing 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 two 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 fragments of code that we used to generate the dataset during training, but some hyperparameters, such as the number of demonstrations and noise level, differ from 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, significantly impacting the reproducibility of the paper.

CARLA. The authors use the CARLA self-driving environment [9] to test BeT's capability in learning from high dimensional observations of (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 greater complexity.

Blockpush. The multi-modal block-pushing environment [10] 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 starting block and target colors. The greater complexity of the environment comes from the stochasticity in the starting positions of the blocks and the trajectory lengths varying between 85 and 201.

Kitchen. The Franka Kitchen environment [11] features human demonstrations recorded via VR headsets in a virtual kitchen environment. The participants 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. This environment has a stochastic starting position, a detail omitted in the paper.

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

C Metrics

C.1 Metrics and Threshold Analysis

Franka Kitchen environment — This section describes the metrics used to analyze performance and diversity in the Franka Kitchen environment, referred to simply as "Kitchen" below. We also delve into a more granular analysis of the *threshold* parameter, which defines 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. The distance between these observations and a set of predefined goals is computed at each step. If this distance falls below a predefined threshold, the goal is achieved. The following metrics are used to assess model performance and diversity in Kitchen:

- Number of tasks completed **[Performance]**: There are seven tasks in total, and the agent has 280 timesteps to complete as many of them as it can. The demonstration dataset has rollouts with four tasks completed in each, and BeT is able to complete more than four in the allotted time in a subset of rollouts.
- Reward **[Performance]**: The reward is calculated based on the number of tasks completed during each rollout. More tasks completed lead to a higher reward.
- Task entropy **[Diversity]**: Entropy computed on the distribution of achieved tasks during 1,000 rollouts. This is the main metric used by the authors.
- Sequence entropy **[Diversity] [New]**: Entropy computed on the sequence distribution of achieved tasks during 1,000 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 tasks when completed in a

different order, we propose this additional metric to better measure the model's multimodality.

Discrepancies in metric computation The authors kindly shared the Jupyter notebook they used to compute their tables, allowing us to describe the two main choices we made regarding metric computation.
We describe below how their version of the metrics and our version of the metrics are computed on the demonstration dataset. The same applies to computing the metrics on a set of rollouts.

- The original work determines the task entropy by first sampling 100 task sequences uniformly at random out of the 566 sequences in the demonstration dataset and computing the entropy on the subset formed by all the tasks completed in these 100 task sequences. This process is repeated 5,000 times, and the final entropy value is obtained by averaging the subset entropies. There was no seed to make the process deterministic, and each entropy calculation will result in a slightly different value. Since precise values were important for reproducibility and random sampling was not considered essential for computing this metric, we decided to made to calculate task entropy directly on all tasks completed in the 566 rollouts, eliminating the stochasticity.
- Additionally, we acquire demonstration sequences directly from the training observations (same as with sampled rollouts), as opposed to the authors obtaining them from the ground truth dataset found in the Relay Policy Learning repository by [11] creating a discrepancy with how they compute them on the rollouts. We 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.

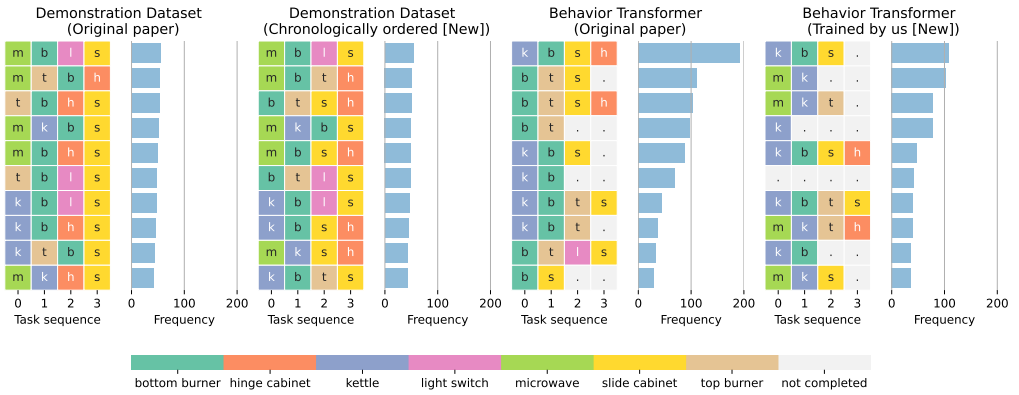

Figure 2. Visualization of the sequence distributions presented by the authors and the ones we obtained. Made using the plotting function from Shafiullah et al.[1].

Threshold analysis An appropriate threshold is pivotal 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.

This value is set to 0.3 for all seven tasks in the original paper. We computed the achievement rate for 24 different threshold values to validate their choice. We also computed the number of rollouts for each tentative value where each task would be considered completed over the human demonstration dataset. The results are then normalized. As shown in figure 3, a threshold of 0.3 is an optimal value as it results in a balanced achievement rate for each task.

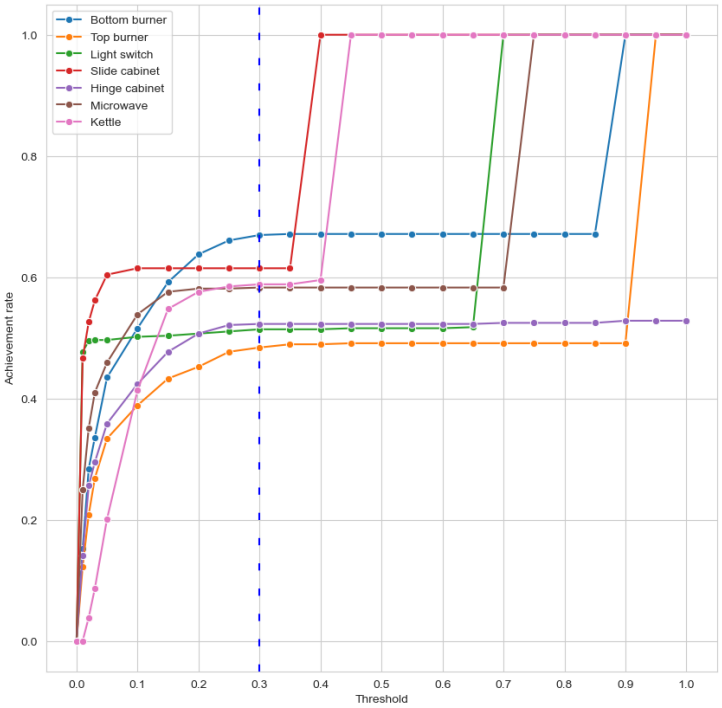

Figure 3. Achievement rate of the different tasks from observations in the demonstration dataset for different threshold values in the Kitchen. The value 0.3 results in a balanced rate for each task.

Blockpush environment — Similarly to Kitchen, we describe the metrics used to analyze performance and diversity of models in the Blockpush environment [10]. Moreover, we also investigate the influence of the *threshold* parameter in this environment.

Metric definition For the Blockpush tasks, Shafiullah et al.[1] adopted the PyBullet-based environment implemented by Florence et al.[10]. The objective of the XArm robot agent 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. A block is considered correctly pushed if the distance between its center and a target falls below a predetermined threshold at any moment in time during the rollout. In this environment, we define the following metrics to assess model performance and diversity in Blockpush, where each probability is obtained by normalizing the number of occurrences over 1,000 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 probability of pushing both blocks into the two different targets.
- 1^{st} block reached (red/green)**[Diversity]**: The probability of the block (red/green) being reached first.
- Push: red block target (red/green) **[Diversity]**: The probability of the red block being pushed to the (green/red) target.
- Push: green block target (red/green) **[Diversity]**: The probability of the green block being pushed to the (green/red) target.

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 has reached its goal by measuring the distance between the initial block position and its position at each subsequent time step. The goal is considered reached if this distance is less than 10^{-3}. 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, directly affecting 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 being misaligned, the arm continues pushing 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 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 5.

Table 5. 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 they are not correctly capturing the primary objectives of the considered environment in some cases 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 pushing 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 than the original implementation. The threshold value used by Florence et al.[10] is 0.04, while the authors used 0.05. With a higher value, we naturally expected a higher success rate.
This was accomplished by calculating success metrics for the two different thresholds on the 1,000 demonstration rollouts from the training dataset and on 1,000 evaluation rollouts using BeT with its original hyperparameter configuration. We also tested other threshold values and noticed that: **i)** a threshold ≤ 0.03 is too strict, making it excessively difficult to succeed; and **ii)** for a threshold ≥ 0.06, it becomes too easy to achieve the goal, and this criterion is no longer a good measure of success.
We observed that in the demonstration dataset, the probabilities remain unchanged when computed using either a threshold of 0.04 or 0.05. A block was considered pushed with a tolerance of 0.05 but not with 0.04 in only one instance out of the 1,000 samples. During the evaluation rollouts using a BeT agent trained with the original hyperparameters, the differences in probabilities are more pronounced, as shown in table 6. We can see that, even though 0.05 is still a reasonable 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 carried out carefully and interpreted with a grain of salt.

Table 6. Comparison of the metrics proposed by the authors for the threshold values of 0.04, used in the original implementation by Florence et al.[10], and 0.05, used by Shafiullah et al.[1]. The two P2 values for "Demonstrations" are rounded to 3 decimal cases to highlight the influence of 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	0.10

D Hyperparameter Sweeps

D.1 MLP hyperparameter optimization

This section presents 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 7.

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 4).

E Point mass figures

This section presents the results obtained by running models on the Point mass environment. In particular, we present the rollouts obtained for BeT (hyperparameters as

Table 7. MLP hyperparameter domain explored by the Optuna algorithm. For the learning rate, Optuna sampled values in a continuous interval. All other parameters were given discrete values. The hyperparameters used in the original paper are underlined except for Batchnorm, as it is unclear whether it was used in the original paper. 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

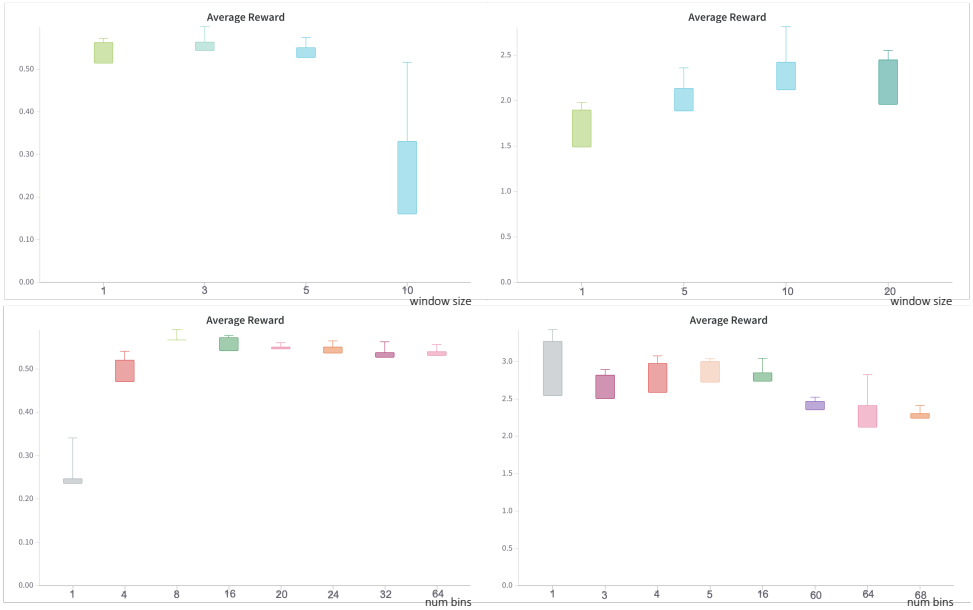

Figure 4. Boxplot for the average reward across rollouts for models swept in the number of action centers (bottom) and of window sizes (top). These sweeps were run both for Blockpush (left) and Kitchen (right). The hyperparameter values are shown on the x-axis, while the y-axis corresponds to the average reward.

specified in the original paper [1]) and for the trunk LSTM ablation (manually fine-tuned in the domain described by Table 8) in figure 5. Moreover, in figure 6, we present the limitation of BeT exposed in section 4.2.

Table 8. Unlike the original work, where the LSTM suffered from training collapse, we find that using a two-layer LSTM trunk is a robust alternative to using a transformer decoder. Hyperparameters for the LSTM model were tuned manually instead of using automated search algorithms on the Point mass environment. The best hyperparameters are in bold.

Hyperparameters	
Optimizer	**Adam**, AdamW
Adam(W) β_2	**0.95**, 0.999
Hidden Width	512, **1024**

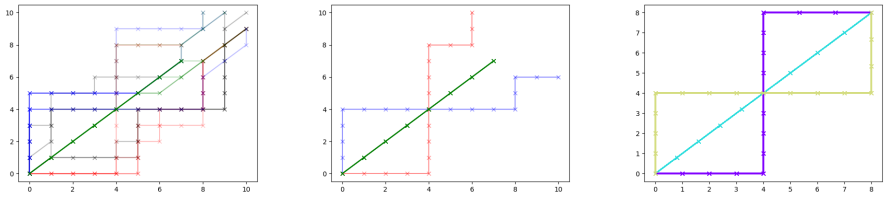

Figure 5. Point mass 2 rollout results with state snapping for models trained on 20,000 training samples without noise. The models used are BeT (left) and an LSTM (middle), both of which target the same dataset (right). Hyperparameters for BeT as specified by Shafiullah et al.[1]. BeT's performance during rollouts is high, but it deviates quite often from the given trajectories. The LSTM's performance is almost perfect, with slight deviations around the last cell. We can clearly distinguish the different modes in the rollouts generated by both models.

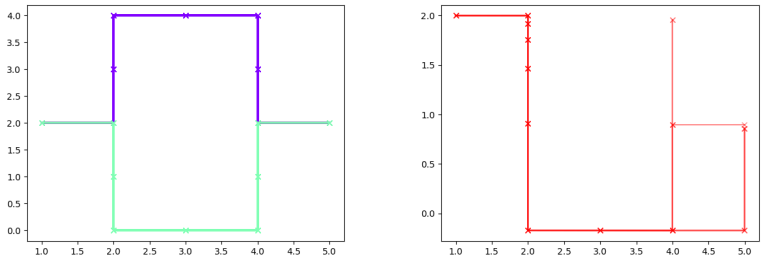

Figure 6. Left: Point mass 1 dataset without noise. There are two modes in the behaviors presented. Right: Rollout exhibiting a limitation of BeT in a simple scenario. It cannot capture the two modes although it has $k = 2$ action center bins, the correct number of modes. This happens when k-means converges to vectors that do not allow the model to distinguish the modes with k bins.