Behavior Transformers: Cloning k modes with one stone

Nur Muhammad (Mahi) Shafiullah [*] **Zichen Jeff Cui**

Ariuntuya Altanzaya **Lerrel Pinto**

New York University

Abstract

While behavior learning has made impressive progress in recent times, it lags behind computer vision and natural language processing due to its inability to leverage large, human-generated datasets. Human behaviors have wide variance, multiple modes, and human demonstrations typically do not come with reward labels. These properties limit the applicability of current methods in Offline RL and Behavioral Cloning to learn from large, pre-collected datasets. In this work, we present Behavior Transformer (BeT), a new technique to model unlabeled demonstration data with multiple modes. BeT retrofits standard transformer architectures with action discretization coupled with a multi-task action correction inspired by offset prediction in object detection. This allows us to leverage the multi-modal modeling ability of modern transformers to predict multi-modal continuous actions. We experimentally evaluate BeT on a variety of robotic manipulation and self-driving behavior datasets. We show that BeT significantly improves over prior state-of-the-art work on solving demonstrated tasks while capturing the major modes present in the pre-collected datasets. Finally, through an extensive ablation study, we analyze the importance of every crucial component in BeT. Videos of behavior generated by BeT are available here: `https://mahis.life/bet`.

1 Introduction

Creating agents that can behave intelligently in complex environments has been a longstanding problem in machine learning. Although Reinforcement Learning (RL) has made significant advances in behavior learning, its success comes at the cost of high sample complexity [57, 23, 1]. Without priors on how to behave, state-of-the-art RL methods require online interactions on the order of 1-10M 'reward-labeled' samples for benchmark control tasks [81]. This is in stark contrast to vision and language tasks, where pretrained models and data-driven priors are the norm [19, 11, 32, 6], which allows for efficient downstream task solving.

So how do we learn behavioral priors from pre-collected data? One option is offline RL [47], where offline datasets coupled with conservative policy optimization can learn task-specific behaviors. However, such methods have yet to tackle domains where task-specific reward labels are not present. Without explicit reward labels, imitation learning, particularly behavior cloning, is a more fitting option [68, 9, 77]. Here, given behavior data $\mathcal{D} \equiv \{s_t, a_t\}$, behavior models can be trained to predict actions $f_\theta(s_t) \to a_t$ through supervised learning. When demonstration data is plentiful, such approaches have found impressive success in a variety of domains from self-driving [68, 14] to robotic manipulation [84, 62]. Importantly, it requires neither online interactions nor reward labels.

[*]Corresponding author, email: `mahi@cs.nyu.edu`

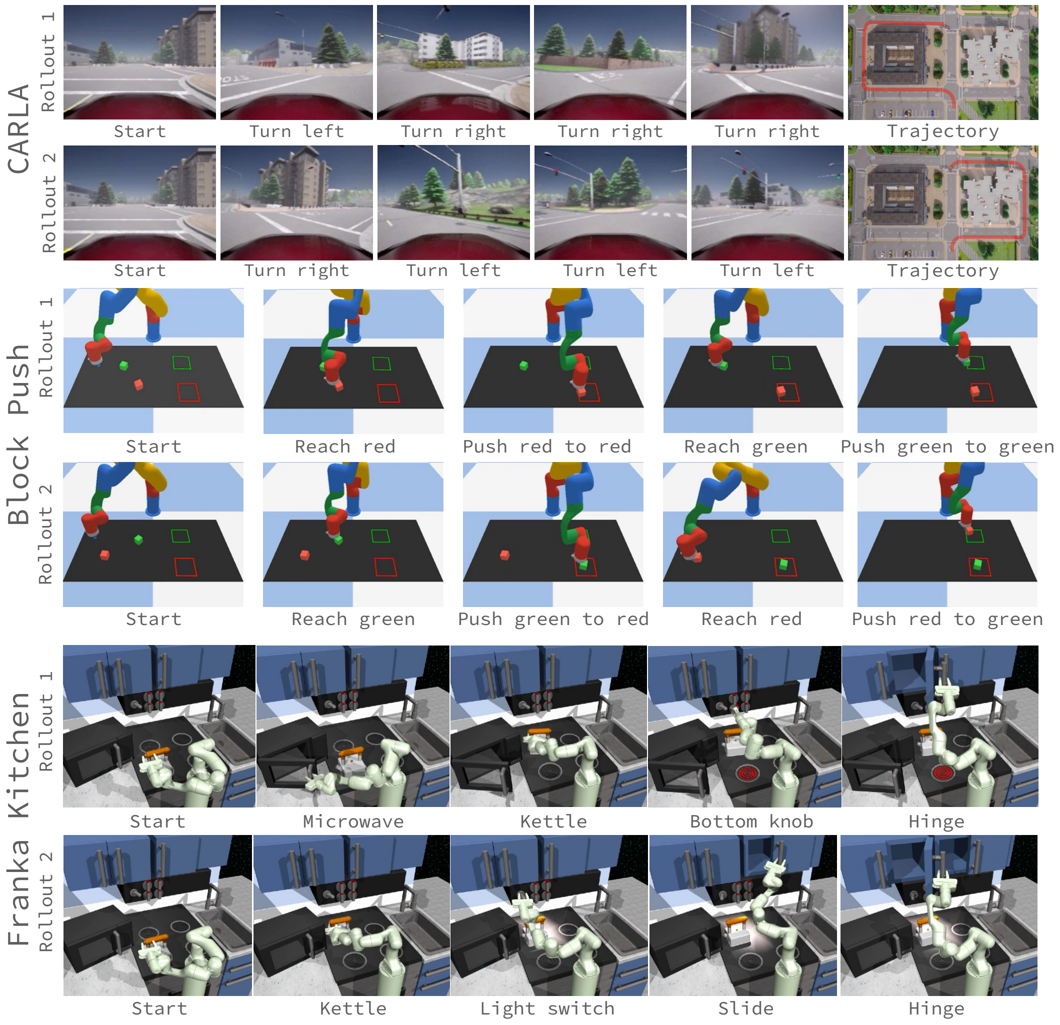

Figure 1: Unconditional rollouts from BeT models trained from multi-modal demonstartions on the CARLA, Block push, and Franka Kitchen environments. Due to the multi-modal architecture of BeT, even in the same environment successive rollouts can achieve different goals or the same goals in different ways.

However, state-of-the-art behavior cloning methods often make a fundamental assumption – that the data is drawn from a unimodal expert solving a single task. This assumption is often baked in to the architecture design, such as using a Gaussian prior. On the other hand, natural pre-collected data is sub-optimal, noisy, and contains multiple modes of behavior, all entangled in a single dataset. This distributionally multi-modal experience is most prominent in human demonstrations. Not only do we perform a large variety of behaviors every day, our personal biases result in significant multi-modality even for the same behavior [31, 52]. Current approach for behavior cloning from such datasets primarily focus on learning goal-conditioned policies, where each goal implies a single mode of behavior [35, 34, 52, 16]. However, even after goal-conditioning, an important question remains: How do we train models that can natively "clone" multi-modal behavior data?

In this work, we present Behavior Transformers (BeT), a new method for learning behaviors from rich, distributionally multi-modal data. BeT is based of three key insights. First, we leverage the context based multi-token prediction ability of transformer-based sequence models [78] to predict multi-modal actions. Second, since transformer-based sequence models are naturally suited to predicting discrete classes, we cluster continuous actions into discrete bins using k-means [53]. This allows us to model high-dimensional, continuous multi-modal action distributions as categorical distributions without learning complicated generative models [42, 20]. Third, to ensure that the actions sampled from BeT are useful for online rollouts, we concurrently learn a residual action corrector to produce continuous actions for a sampled action bin.

We experimentally evaluate BeT on five datasets ranging from simple diagnostic toy datasets to complex datasets that include simulated robotic pushing [25], sequential task solving in kitchen environments [34], and self-driving with visual observations in CARLA [21]. The two main findings from these experiments can be summarized as:

1. On multi-modal datasets, BeT achieves significantly higher performance during online rollouts compared to prior behavior modelling methods.
2. Rather than collapsing or latching onto one mode, BeT is able to cover the major modes present in the training behavior datasets. Unconditional rollouts from this model can be seen in Fig. 1.

All of our datasets, code, and trained models will be made publicly available.

2 Behavior Transformers

Given a dataset of continuous observation and action pairs $\mathcal{D} \equiv \{(o, a)\} \subset \mathcal{O} \times \mathcal{A}$ that contains behaviors we are interested in, our goal is to learn a behavior policy $\pi : \mathcal{O} \mapsto \mathcal{A}$ that models this data without any online interactions with the environment or reward labels. This setup follows the Behavior Cloning formulation, where policies are trained to model demonstrations from expert rollouts. Often, such policies are chosen from a hypothesis class parametrized by parameter set θ. Following this convention, our objective is to find the parameter θ that maximizes the probability of the observed data

$$\theta^* := \arg\max_\theta \prod_t \mathbb{P}(a_t \mid o_t; \theta) \tag{1}$$

When the model class is restricted to unimodal isotropic Gaussians, this maximum likelihood estimation problem leads to minimizing the Mean Squared Error (MSE), $\sum_t \|a_t - \pi(o_t; \theta)\|^2$.

Limitations of traditional MSE-based BC: While MSE-based BC has been able to solve a variety of tasks [9, 77], it assumes that the data distribution is unimodal. Clean data from an expert demonstrator solving a particular task in a particular way satisfies this assumption, but pre-collected intelligent behavior often may not [52, 34]. While more recent behavior generation models have sought to address this problem, they often require complex generative models [76], an exponential number of bins for actions [54], complicated training schemes [66], or time-consuming test-time optimization [25]. An experimental analysis of some of these prior works is presented in Section 3.

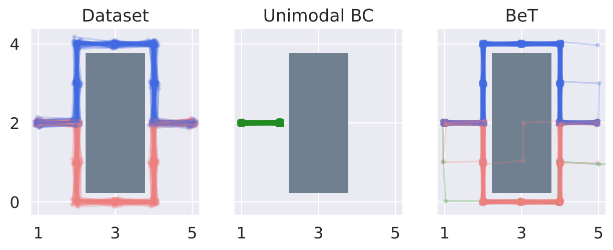

Figure 2: Comparison between a regular MSE-based BC model and a BeT models that can capture multi-modal distributions. The MSE-BC model takes 0 action to minimize MSE.

Overview of Behavior Transformers (BeT): We address two critical assumptions in regular BC. First, we relax the assumption that the behavior we are cloning is purely Markovian, and instead model $P(a_t \mid o_t, o_{t-1}, \cdots, o_{t-h+1})$ for some horizon h. Second, instead of assuming that actions are generated by a unimodal action distribution, we model our action distribution as a mixture of gaussians. However, unlike previous efforts similar to Mixture Density Networks (MDN) to do so, whose limitations have been explored in Florence et al. [25], we do not explicitly predict mode centers, which significantly improves our modeling capacity. To operationalize these two features in a single behavior model, we make use of transformers since (a) they are effective in utilizing prior observational history, and (b) they are naturally suited to output multi-modal tokens through their architecture.

2.1 Action discretization for distribution learning

Although transformers have become standard as a backbone for sequence-to-sequence models [19, 11], they are designed to process discrete tokens and not continuous values. In fact, modeling multi-modal

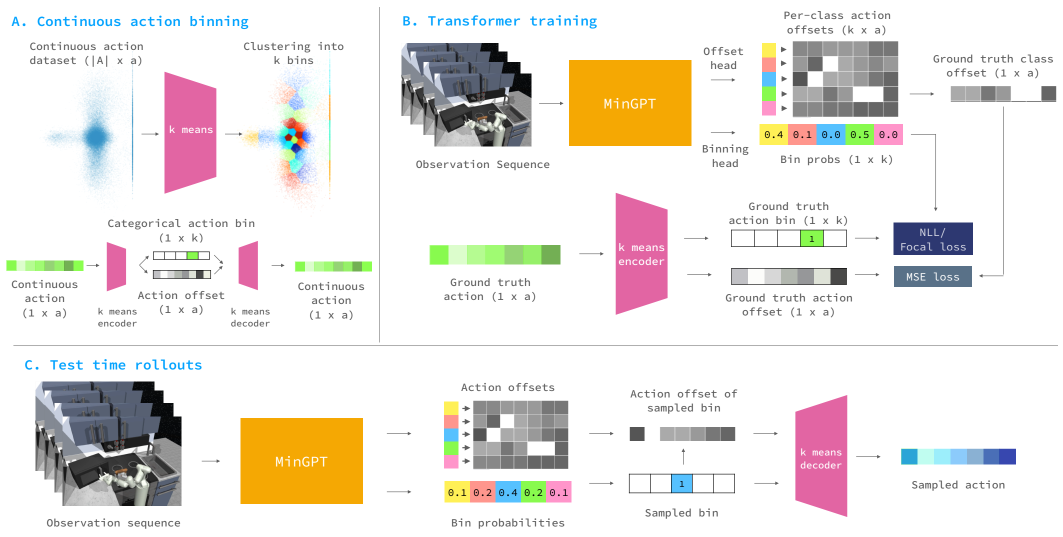

Figure 3: Architecture of Behavior Transformer. (A) The continuous action binning using k-means algorithm that lets BeT split every action into a discrete bin and a continuous offset, and later combine them into one full action. (B) Training BeT using demonstrations offline; each ground truth action provides a ground truth bin and residual action, which is used to train the minGPT trunk with its binning and action offset heads. (C) Rollouts from BeT in test time, where it first chooses a bin and then picks the corresponding offset to reconstruct a continuous action.

distributions of high-dimensional continuous variables in a tractable manner is in itself a challenging problem, especially if we want the trained behavior model to cover the modes present in the dataset. To address this, we propose a new factoring of the action prediction task by dividing each action in two parts: a categorical variable denoting an 'action center', and a corresponding 'residual action'.

To this end, given the actions in our dataset, we first optimize for a set of k action centers, $\{A_1, A_2, \cdots, A_k\} \subset \mathcal{A}$. We then decompose each action into two parts: a categorical variable representing the closest action bin, $\lfloor a \rfloor := \arg\min_i \|a - A_i\|_2$, and a continuous residual action $\langle a \rangle := a - A_{\lfloor a \rfloor}$. If we are given the set of action centers $\{A_i\}_{i=1}^k$, an action bin index $\lfloor a \rfloor$ and the residual action $\langle a \rangle$, we can deterministically reconstruct the true action $a := A_{\lfloor a \rfloor} + \langle a \rangle$. Once learned, these k-means based encoder and decoders for this action factorization process are fixed for the rest of the train and testing phases. The action factorization procedure is illustrated in Fig. 3 (A).

2.2 Attention-based behavior mode learning

Once we have the clustering based autoencoder learned from the actions in the dataset, we model our demonstration trajectories with BeT. We use a transformer decoder model, namely minGPT [11], with minor modifications, as our backbone. The transformer $\mathcal{T}$ takes in a sequence of continuous observations $(o_i, o_{i+1}, \cdots, o_{i+h-1})$ and learns a sequence-to-sequence model mapping each observation to a categorical distribution over k discrete action bins. The predicted probability sequence is then compared with the ground truth labels, $(\lfloor a_i \rfloor, \lfloor a_{i+1} \rfloor, \lfloor a_{i+2} \rfloor, \cdots, \lfloor a_{i+h-1} \rfloor)$. We use a negative log-likelihood-based Focal loss [49] between the predicted categorical distribution probabilities and the ground truth labels to train the transformer head. Focal loss is a simple modification over the standard cross entropy loss. While the standard cross entropy loss for binary classification can be thought of $\mathcal{L}_{ce}(p_t) = -\log(p_t)$, Focal loss adds a term $(1-p_t)^\gamma$ to this, to make the new loss

$$\mathcal{L}_{focal}(p_t) = -(1-p_t)^\gamma \log(p_t)$$

This loss has the interesting property that its gradient is more steep for smaller values of p_t, while flatter for larger values of p_t. Thus, it penalizes and changes the model more for making errors in the low-probability classes, while is more lenient about making errors in the high probability classes. The model is illustrated in Fig. 3 (B).

2.3 Action correction: from coarse to finer-grained predictions

Using a transformer allows us to model multi-modal actions. However, discretizing the continuous action space in any way invariably causes loss of fidelity [39]. Discretization error may cause online rollouts of the behavior policy to go out of distribution from the original dataset [73], which can in turn cause critical failures. To predict the complete continuous action, we add an extra head to the transformer decoder that offsets the discretized action centers based on the observations.

For each observation o_i in the sequence, the head produces a $k \times \dim(A)$ matrix with k proposed residual action vectors, $\left(\langle a_i^{(j)} \rangle\right)_{j=1}^{k} = (\langle \hat{a}_i^{(1)} \rangle, \langle \hat{a}_i^{(2)} \rangle, \langle \hat{a}_i^{(3)} \rangle, \cdots, \langle \hat{a}_i^{(k)} \rangle)$, where the residual actions correspond to bin centers $A_1, A_2, A_3, \cdots, A_k$. These residual actions are trained with a loss akin to the *masked multi-task loss* [30] from object detection. In our case, if the ground truth action is $\mathbf{a}$, the loss is:

$$\text{MT-Loss}\left(\mathbf{a}, \left(\langle \hat{a}_i^{(j)} \rangle\right)_{j=1}^{k}\right) = \sum_{j=1}^{k} \mathbb{I}[\lfloor \mathbf{a} \rfloor = j] \cdot \| \langle \mathbf{a} \rangle - \langle \hat{a}^{(j)} \rangle \|_2^2 \quad (2)$$

Where $\mathbb{I}[]$ denotes the Iverson bracket, ensuring the offset head of BeT only incurs loss from the ground truth class of action $\mathbf{a}$. This mechanism prevents the model from trying to fit the ground truth action using the offset at every index.

2.4 Test-time sampling from BeT

During test time, at timestep t we input the latest h observations $(o_t, o_{t-1}, \cdots, o_{t-h+1})$ to the transformer, combining the present observation o_t with $h-1$ previous observations. Our trained MinGPT model gives us $h \times 1 \times k$ bin center probability vectors, and $h \times k \times \dim(A)$ offset matrix. To sample an action at timestep t, we first sample an action center according to the predicted bin center probabilities on the t^{th} index. Once we have chosen an action center $A_{t,j}$, we add the corresponding residual action $\langle \hat{a}_t^{(j)} \rangle$ to it to recover a predicted continuous action $\hat{\mathbf{a}}_{\mathbf{t}} = A_{t,j} + \langle \hat{a}_t^{(j)} \rangle$. This sampling procedure is illustrated in Fig. 3 (C).

3 Experiments

We now study the empirical performance of BeT on a variety of behavior learning tasks. Our experiments are designed to answer the following questions: (a) Is BeT able to imitate multi-modal demonstrations? (b) How well does BeT capture the modes present in behavior data? (c) How important are the individual components of BeT?

3.1 Environments and datasets

We experiment with five broad environments. While full descriptions of these environments, dataset creation procedure, and overall statistics are in Appendix A, a brief description of them are as follows.

(a) **Point mass environment #1:** Our first set of experiments in Fig. 2, used to get a qualitative understanding of BeT, were performed in a simple Pointmass environment with a 2D observation and action space with two hundred demonstrations. The pre-collected demonstrations start at a fixed point, and then make their way to another point while avoiding a block in the middle. The two primary modes in this dataset are taking a left turn versus a right turn.

(b) **Point mass environment #2:** The setup is similar to the previous environment with the exception of one straight line and two complicated prolonged 'Z' shaped modes of demonstration (Fig. 5.)

(c) **CARLA self-driving environment:** CARLA [21] uses the Unreal Engine to provide a simulated driving environment in a visually realistic landscape. The agent action space is 2D (accelerate/brake and left/right steer), while the observation space is (224,224,3)-dimensional RGB image from the car. A hundred total demonstrations drive around a building block in two distinct modes. This environment highlights the challenge of behavior learning from high-dimensional observations as shown in Fig. 1 (a). For visual observations with BeT, we use a frozen ResNet-18 [36] pretrained on ImageNet [18] as an encoder.

(d) **Multi-modal block-pushing environment:** For more complicated interaction data, we use the multi-modal block-pushing environment from Implicit Behavioral Cloning (IBC) [25], where an XArm robot needs to push two blocks into two squares in any order. The blocks and target squares are colored red and green. The positions of the blocks are randomized at episode start. We collect 1,000 demonstrations using a deterministic controller with two independent axes of multi-modality: (a) it starts by reaching for either the red or the green block, with 50% probability, and (b) it pushes the blocks to (red, green) or (green, red) squares respectively with 50% probability.

(e) **Franka kitchen environment:** To highlight the complexity of performing long sequences of actions, we use the Relay Kitchen Environment [34] where a Franka robot manipulates a virtual kitchen environment. We use the relay policy learning dataset with 566 demonstrations collected by human participants wearing VR headsets. The participants completed a sequence of four object-interaction tasks in each episode [34]. There are a total of seven interactable objects in the kitchen: a microwave, a kettle, a slide cabinet, a hinge cabinet, a light switch, and two burner knobs. This dataset contains two different kinds of multi-modality: one from the inherent noise in human demonstrations, and another from the demonstrators' intent.

3.2 Baseline behavior learning methods

While a full description of our baselines are in Appendix B.1, a brief description of them is here:

(a) **Multi-layer Perceptron with MSE (RBC):** We use MLP networks trained with MSE loss as our first baseline, since this is the standard way of performing behavioral cloning for a new task [77]. A comparison with transformer-based behavior cloning is discussed in Section 3.5.

(b) **Nearest neighbor (NN):** Nearest neighbor based algorithms are easy to implement, and has recently shown to have strong performance on complicated behavioral cloning tasks [3].

(c) **Locally Weighted Regression (LWR):** This non-parametric approach provides better regularization compared to NN and is a strong alternative to parametric BC [4, 62].

(d) **Variational auto-encoders (VAE):** Inspired by SPiRL [66], where behavioral priors are learned through a VAE [42], we compare with continuous actions generated from the VAE and the prior.

(e) **Normalizing Flow (Flow):** Inspired by PARROT [76], where state-conditioned action priors are learned through a Flow model [20], we compare with actions generated from the Flow model.

(f) **Implicit Behavioral Cloning (IBC):** Instead of modeling the conditional distribution $P(a \mid o)$, IBC models the joint probability distribution $P(a, o)$ using energy-based models [25]. While IBC is slower than explicit BC models because of their sampling requirements, they have been shown to learn well on multi-modal data, and outperform earlier work such as MLP-MDNs [8].

3.3 Is BeT able to imitate multi-modal demonstrations?

The first question we ask is whether BeT can actually clone behaviors given a mixed dataset of unlabeled, multi-modal behaviors. To examine that, we look at the performance of our model in CARLA, Block push, and Kitchen environments compared with our baselines in Table 1.

We see that BeT outperforms all other methods in all environments except CARLA, where it is narrowly outperformed by LWR. Since the models are all behavioral cloning algorithms, they share the failure mode of failing once the observations go out of distribution (OOD). However, they vary in the tolerance. For example, BeT shines in the Block push environment, where alongside extreme environment randomness and multi-modality, the models also have to learn significant long-term behaviors and commit to a single mode over a long period. While all baselines can somewhat successfully reach one block, they fail to complete the long-horizon, multi-modal task of pushing two blocks into two different bins. On the other hand, we observe that BeT's primary failure mode is not realizing a block has not completely entered the target yet, while other methods either go OOD quickly, or keep switching between modes. We also observe that BeT performs well even in complex observation and action spaces. In the CARLA environment, the model takes in visual observations, while in the Franka Kitchen environment, the action space corresponds to a 9-DOF torque controlled robot. BeT handles both cases with the same ease as it does environments with lower-dimensional observation or action spaces.

Table 1: Performance of BeT compared with different baselines in learning from demonstrations. For CARLA, we measure the probability of the car reaching the goal successfully. For Block push, we measure the probability of reaching one and two blocks, and the probabilities of pushing one and two blocks to respective squares. For Kitchen, we measure the probability of n tasks being completed by the model within the allotted 280 timesteps. Evaluations are over 100 rollouts in CARLA and 1,000 rollouts in Block push and Kitchen environments.

	CARLA	Block push				Kitchen				
	Driving	Reach		Push		# Tasks completed				
Baselines	Success	R1	R2	P1	P2	1	2	3	4	5
RBC	0.98	0.67	0	0	0	0	0	0	0	0
1-NN	0.99	0.49	0.05	0.01	0	0.90	0.72	0.44	0.17	0
LWR	**1**	0.50	0.06	0	0	**1**	0.83	0.52	0.21	0
VAE	0	0.60	0.05	0	0	**1**	0	0	0	0
Flow	0.03	0.59	0.02	0	0	0.04	0	0	0	0
IBC	0.25	0.98	0.04	0.01	0	0.99	0.87	0.61	0.24	0
BeT (Ours)	0.98	**1**	**0.99**	**0.96**	**0.71**	0.99	**0.93**	**0.71**	**0.44**	**0.02**

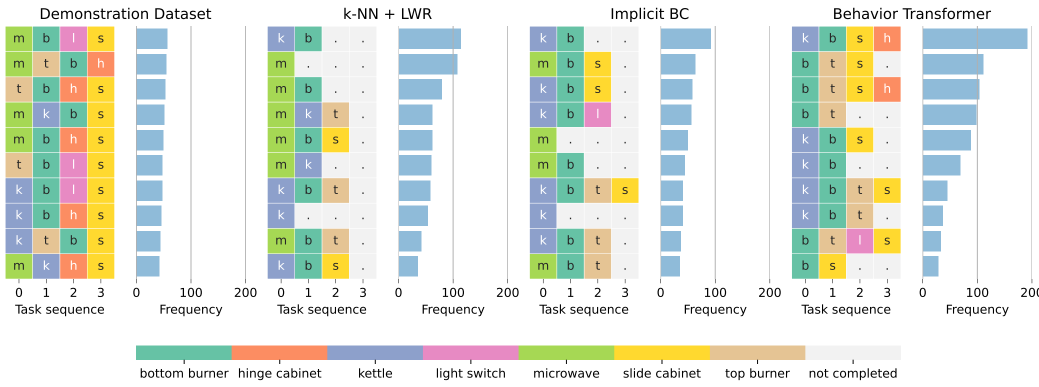

Figure 4: Distribution of most frequent tasks completed in sequence in the Kitchen environment. Each task is colored differently, and frequency is shown out of a 1,000 unconditional rollouts from the models.

3.4 Does BeT capture the modes present in behavior data?

Next, we examine the question of whether, given a dataset where multi-modal behavior exists, our model learns behavior that is also multi-modal. Here, we are interested in seeing the variance of the behavior of the model over different rollouts. In each of our environments, the demonstrations contain different types of multi-modality. As a result, we show a comprehensive analysis of multi-modality seen in our agent behaviors.

Table 2: Multimodality learned from the multimodal demonstrations by different algorithms. In CARLA, we consider the probability of turning left vs. right at the intersection, ignoring OOD rollouts. In Block push, we consider two set of probabilities, (a) which block was reached first, and (b) what was the pushing target for each block. Finally, in Franka Kitchen, we consider the empirical entropy for the task sequences, considered as strings, sampled from the model. We highlight the values closest to the corresponding demonstration values.

	CARLA		Block: first block reached		Push: red block target		Push: green block target		Kitchen
Baselines	Left	Right	Red	Green	Red	Green	Red	Green	Task entropy
RBC	0	0.98	0.41	0.25	0	0	0	0	0
1-NN	0	0.99	0.24	0.25	0	0	0	0.01	2.12
LWR	0	1	0.26	0.26	0.01	0	0.01	0.01	2.29
VAE	0	0	0.27	0.33	0	0	0	0	0.72
Flow	0	0	0.31	0.29	0	0	0	0	0.08
IBC	0.12	0.13	**0.48**	**0.50**	0	0	0.01	0.01	2.41
BeT (Ours)	**0.34**	**0.64**	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	0.50	0.50	2.96

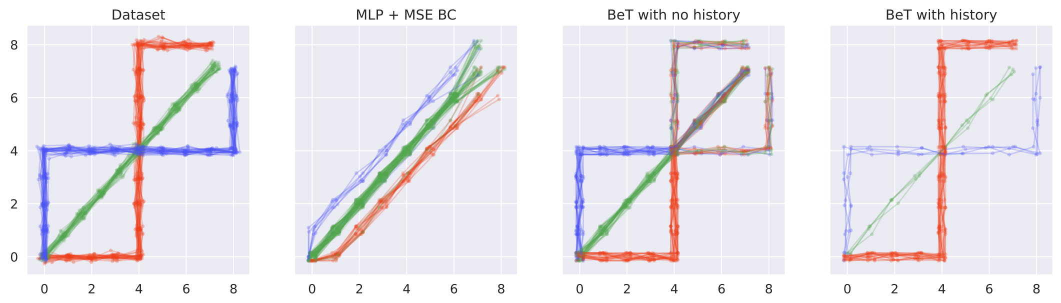

Figure 5: Comparison between an RBC model and two BeT models, trained with and without historical context on a dataset with three distinct modes. BeT with history is better able to capture the context-dependant behavior in the demonstrations.

We see in Table 2 that in CARLA and Block push, BeT covers all the modes of the demonstration data, even in the few cases where it does not perfectly match the demonstrated task probabilities. For the Kitchen environment, we see in Fig. 4 that BeT visits certain strings of tasks more frequently than in the original demonstrations. However, compared to other strong baselines, BeT generates longer task strings more often while maintaining diversity and not collapsing to a single mode.

3.5 How important are the individual components of BeT?

There are four key differences between BeT architecture and standard BC: (a) binning actions into discrete clusters, (b) using offsets to faithfully reconstruct actions later, (c) learning sequentially to use historical context, and (d) using an attention-based MinGPT trunk. In this section, we discuss the impacts they have in BeT performance.

Impact of discrete binning: Intuitively, having discrete options for bin centers is what enables BeT to express multi-modal behavior even when starting from an identical starting state. Indeed, if there is no binning, we see from Table 3 that the performance of BeT drops significantly. More tellingly, in the Franka Kitchen environment, the model only ever completed a subsequence of (kettle, top/bottom burner, light switch, slide cabinet) tasks after 100 random rollouts. This result shows us that having discrete bins helps BeT achieve multi-modality. We also experiment with the Mixture density networks (MDN) [8] and uniform quantization, as shown in previous works [25, 39]. We see that they may perform well sometimes but overall still fall short of our k-means binning approach.

Table 3: Relative performance of ablated variants of BeT, normalized by average BeT successes at the task

Ablations	CARLA	Block push	Kitchen
No offsets	0.94	0.95	0.78
No binning	0.94	0.25	0.68
No history	0.65	0.95	0.88
MLP	0.90	0	0.05
Temp. Conv	0.72	0.01	0.26
LSTM	0.03	0.03	0.04
GPT-MDN	0.30	0.83	0.86
Unif. quant.	0.90	0.96	0.90

Necessity of action offsets: An important feature of BeT is the residual action offset that corrects the discrete actions coming from the bins. While the bin centers may be quite expressive, Table 3 shows that the inability to correct them causes a performance degrade. Interestingly, the largest degradation comes in the Kitchen environment, which also has the highest dimensional action space. Intuitively, we can understand how in higher dimension the loss of fidelity from discretizing would be higher, and the relative performance loss across three environments support that hypothesis.

Importance of historical context: While RL algorithms traditionally assume environments are Markovian, human behavior in an open-ended environment is rarely so. Thus, using historical context helps BeT to perform well. We show a simple experiment in Fig. 5 on the second point mass environment. Here, training and evaluating with some historical context allows BeT to follow the demonstrations better. We experience the same in the CARLA, Block push, and Kitchen environments, where training with some historical context raises performance across the board as seen in Table 3.

Importance of transformer architecture: Despite transformers' success in other fields of machine learning, it is natural to wonder whether the tasks BeT solves here really requires one. We ablated

BeT by replacing the MinGPT trunk with an MLP, Temporal Convolution, and LSTMs, and found that they have lower performance while also being difficult to train stably. This performance reduction remains even if the MLP is given some historical context by stacking h observations before passing it to the MLP. See Table. 3 for results and Appendix C.3 for further details.

Computation considerations: While transformers in usual contexts are large models, we downscale them for our application in BeT (See Appendix B.4). Our models contain on the order of 10^4-10^6 parameters, and even with a small batch size trains within an hour for our largest datasets (Block push) on a single desktop GPU. In contrast, for the same task, our strongest baseline IBC takes about 14 hours. Evaluation rollouts on the same environment take 1.65 seconds with BeT, as opposed to 17.70 seconds with IBC.

4 Related Work

This paper builds upon a rich literature in imitation learning, offline learning, generative models, and transformer architectures. The most relevant ones to our work are discussed here.

Learning from offline data: Since Pomerleau [67] showed the possibility of driving an autonomous vehicle using offline data and a neural network, learning behavior from offline data has been a continuous topic of research for scalable behavior learning [2, 7, 75]. The approaches can be divided into two broad classes: Offline RL [27, 44, 45, 80, 47, 26], focusing on learning from datasets of a mixed quality that also have reward labels; and imitation learning [61, 63, 64, 37], focusing on learning behavior from a dataset of expert behavior without reward labels. BeT falls under the second category, as it is a behavior cloning model. Behavior cloning is a form of imitation learning that tries to model the action of the expert given the observation which is often used in real-world applications [84, 85, 84, 69, 24, 83]. As behavior cloning algorithms are generally solving a fully supervised learning problem, they tend to be faster and simpler than reinforcement learning or offline RL algorithms and in some cases show competitive results [26, 33].

Generative models for behavior learning: One approach for imitation learning is Inverse Reinforcement Learning or IRL [74, 59], where given expert demonstrations, a model tries to construct the reward function. This reward function is then used to generate desirable behavior. GAIL [37], an IRL algorithm, connects generative adversarial models with imitation learning to construct a model that can generate expert-like behavior. Under this IRL framework, previous works have tried to predict multi-modal, multi-human trajectories [46, 38]. Similarly, other works have tried Gaussian Processes [71] for creating dynamical models for human motion [79]. Another class of algorithms learn a generative action decoder [65, 52, 76] from interaction data to make downstream reinforcement learning faster and easier, which inspired BeT's action factorization. Finally, a class of algorithms, most notably [50, 25, 43, 58] do not directly learn a generative model but instead learn energy based models. These energy based models can then be sampled to generate desired behavior. Since [25] is a BC model capable of multi-modality, we compare against it as a baseline in Sec. 3.

Transformers for control: With the stellar success of transformer models [78] in natural language processing [19, 11] and computer vision [22], there has been significant interest in using transformer models to learn behavior and control. Among those, [12, 39] applies them to Reinforcement Learning and Offline Reinforcement Learning, respectively, while [13, 16, 54] use them for imitation learning. [16, 54] use transformers mostly to summarize historical visual context, while [13] relies on their long-term extrapolation abilities to collect human-in-the-loop demonstrations more efficiently. BeT is inspired by both of these use cases, as we use a transformer to summarize historical context while leveraging its generative abilities. Architecturally, BeT is most closely related to the imitation learning variant of [39], with a significant difference that while [39] learns the joint state, action distribution, BeT learns the conditional distribution of action given state, which allows BeT to tackle much more complicated state spaces.

Datasets for distributionally multi-modal data: Similar to computer vision [18, 48, 51] and natural language processing [10, 70], there has been a recent interest in collecting behavior datasets that may aid in downstream behavior learning. Some of them are labeled with agent goals or rewards for downstream tasks [55, 26, 56], while others are more open ended [34, 52, 82] and come without reward or task labels. In our work, we focus towards the latter class. The lack of labeled goal or reward labels in the second category implies that there is more multi-modality in the action distributions compared to action distributions of goal or reward conditioned datasets, which is the same reason a lot of work learning from multi-modal datasets try to learn a goal-conditioned model [35, 34, 52, 16]. Finally, the lack of labelling requirements mean that the unlabelled datasets are cheaper to obtain, which should help BeT scale further in the future.

5 Discussions

In this work, we introduce Behavior Transformers (BeT), which uses a transformer-decoder based backbone with a discrete action mode predictor coupled with a continuous action offset corrector to model continuous actions sequences from open-ended, multi-modal demonstrations. While BeT shows promise, the truly exciting use of it would be to learn diverse behavior from human demonstrations or interactions in the real world. In parallel, extracting a particular, unimodal behavior policy from BeT during online interactions, either by distilling the model or by generating the right ‘prompts’ [72], would make BeT tremendously useful as a prior for online Reinforcement Learning.

Acknowledgments

We thank Ben Evans, David Brandfonbrener, Abitha Thankaraj, Jyo Pari, and Anthony Chen for valuable feedback and discussions. This work was supported by grants from Honda, and ONR award numbers N00014-21-1-2404 and N00014-21-1-2758.

References

[1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[3] S. P. Arunachalam, S. Silwal, B. Evans, and L. Pinto. Dexterous imitation made easy: A learning-based framework for efficient dexterous manipulation. *arXiv preprint arXiv:2203.13251*, 2022.

[4] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Lazy learning*, pages 11–73, 1997.

[5] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[6] A. Bardes, J. Ponce, and Y. LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2021.

[7] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Survey: Robot programming by demonstration. *Handbook of robotics*, 59(BOOK_CHAP), 2008.

[8] C. M. Bishop. Mixture density networks. *Neural Computing Research Group Report*, 1994.

[9] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[10] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[11] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[12] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *CoRR*, abs/2106.01345, 2021. URL `https://arxiv.org/abs/2106.01345`.

[13] H. M. Clever, A. Handa, H. Mazhar, K. Parker, O. Shapira, Q. Wan, Y. Narang, I. Akinola, M. Cakmak, and D. Fox. Assistive tele-op: Leveraging transformers to collect robotic task demonstrations. *arXiv preprint arXiv:2112.05129*, 2021.

[14] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

[15] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub Repository*, 2016.

[16] S. Dasari and A. Gupta. Transformers for one-shot visual imitation. *arXiv preprint arXiv:2011.05970*, 2020.

[17] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, pages 4171–4186, 2018. doi: 10.18653/v1/N19-1423.

[20] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[21] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[22] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[23] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1329–1338. JMLR.org, 2016.

[24] P. Florence, L. Manuelli, and R. Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 5(2):492–499, 2019.

[25] 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 preprint arXiv:2109.00137*, 2021.

[26] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[27] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, PMLR, 2018.

[28] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.

[29] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[30] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[31] K. Grauman, A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. *arXiv preprint arXiv:2110.07058*, 3, 2021.

[32] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

[33] C. Gulcehre, Z. Wang, A. Novikov, T. Le Paine, S. Gomez Colmenarejo, K. Zolna, R. Agarwal, J. Merel, D. Mankowitz, C. Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv e-prints*, pages arXiv–2006, 2020.

[34] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[35] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, and J. J. Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. *Advances in neural information processing systems*, 30, 2017.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[37] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, volume 29, pages 4565–4573, 2016.

[38] B. Ivanovic, E. Schmerling, K. Leung, and M. Pavone. Generative modeling of multimodal multi-human behavior. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3088–3095. IEEE, 2018.

[39] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34, 2021.

[40] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

[41] A. Karpathy. GitHub - karpathy/minGPT: A minimal PyTorch re-implementation of the OpenAI GPT (Generative Pretrained Transformer) training, 08 2020. URL https://github.com/karpathy/minGPT.

[42] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[43] I. Kostrikov, J. Tompson, R. Fergus, and O. Nachum. Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*, 2021.

[44] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

[45] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[46] N. Lee and K. M. Kitani. Predicting wide receiver trajectories in american football. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9. IEEE, 2016.

[47] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[48] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[49] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[50] M. Liu, T. He, M. Xu, and W. Zhang. Energy-based imitation learning. *arXiv preprint arXiv:2004.09395*, 33, 2020.

[51] Z. Liu, P. Luo, X. Wang, and X. Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15(2018):11, 2018.

[52] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132. PMLR, 2020.

[53] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[54] Z. Mandi, F. Liu, K. Lee, and P. Abbeel. Towards more generalizable one-shot visual imitation learning. *arXiv preprint arXiv:2110.13423*, 2021.

[55] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR, 2018.

[56] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[57] 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. *Nature*, 518(7540):529–533, 2015.

[58] O. Nachum and M. Yang. Provable representation learning for imitation with contrastive fourier features. *arXiv preprint arXiv:2105.12272*, 2021.

[59] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pages 663–670, 2000.

[60] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[61] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018.

[62] J. Pari, N. Muhammad, S. P. Arunachalam, L. Pinto, et al. The surprising effectiveness of representation learning for visual imitation. *arXiv preprint arXiv:2112.01511*, 2021.

[63] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[64] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (TOG)*, 40(4):1–20, 2021.

[65] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. *arXiv preprint arXiv:2010.11944*, 2020.

[66] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. *arXiv preprint arXiv:2010.11944*, 2020.

[67] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[68] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[69] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3758–3765. IEEE, 2018.

[70] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[71] C. E. Rasmussen and H. Nickisch. Gaussian processes for machine learning (gpml) toolbox. *The Journal of Machine Learning Research*, 11:3011–3015, 2010.

[72] L. Reynolds and K. McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2021.

[73] S. Ross, G. Gordan, and A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *arXiv preprint arXiv:1011.0686*, 2010.

[74] S. Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

[75] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3 (6):233–242, 1999.

[76] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[77] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[78] 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: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[79] J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2007.

[80] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[81] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[82] S. Young, J. Pari, P. Abbeel, and L. Pinto. Playful interactions for representation learning. *arXiv preprint arXiv:2107.09046*, 2021.

[83] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, 2020.

[84] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *ICRA*, pages 5628–5635. IEEE, 2018.

[85] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes]
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Sec. 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See abstract.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix B
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Sec 3.5.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See linked code.
   (b) Did you mention the license of the assets? [Yes] See linked code.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

Diverse, multi-modal behaviors generated by our models on different environment are best experienced and understood in a video. We invite you to visit `https://mahis.life/bet` to see BeT models in action.

A Environment and Dataset Details

Point mass environments: In the point mass environment, we have a simple point-mass agent with two-dimensional observation and action spaces. The observation of the agent denotes the (x, y) position of the agent, while the action sets the immediate $(\Delta x, \Delta y)$ displacement of the agent in the next timestep.

To show the effects of unimodal and multimodal behavioral cloning algorithms more cleanly, we also add a “snapping" effect to the environment which moves the agent close to the nearest integer coordinates after each step.

We generate random trajectories for each of our Multipath experiment datasets.

1. In the first one (Fig. 2), our dataset has two modes, which are colored differently in the figure based on the path taken at the fork.
   (a) In the first set of demonstrations, the point mass follows the trajectory $(1, 2), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4), (4, 3), (4, 2), (5, 2)$.
   (b) In the second set of demonstrations, the point mass follows $(1, 2), (2, 2), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2), (5, 2)$.
2. For the second Multipath environment (Fig. 5), there are three modes of demonstration, which are colored in the figure according to their first step direction.
   (a) In the first set of demonstration, the point mass follows $x = y$ from $(0, 0)$ to $(8, 8)$ with $\sqrt{2}$ size step increments.
   (b) In the second set of demonstration, the point mass follows straight lines from $(0, 0) \to (0, 4) \to (4, 4) \to (8, 4) \to (8, 8)$ with step size 1.
   (c) In the third set of demonstration, the point mass follows straight lines from $(0, 0) \to (4, 0) \to (4, 4) \to (4, 8) \to (8, 8)$ with step size 1.

CARLA environment: We use the CARLA [21] self-driving environment to examine BeT performance in environments with high-dimensional observation spaces. CARLA uses the Unreal Engine to provide a photo-realistic driving simulation. We create our environment on the Town04 map in CARLA 0.9.13. The observation space is $224 \times 224 \times 3$ RGB images from the vehicle, which are processed by an ImageNet-pretrained, frozen ResNet-18 to a 512-dimensional real-valued vector. The action space is $[-1, 1]^2$ with an accelerator-brake axis and a steering axis.

The dataset on this environment is collected with the built-in PID agent with minor tuning. We fix waypoints in the trajectory that the demonstration agent needs to follow. The waypoints fork around two central blocks: one set of trajectories thus go to the left, while another set of demonstration trajectories go to the right. While collecting the demonstrations, we add some noise in the environment before executing an action so that there is some variation in the set of 100 total demonstrations that we collect in the environment.

We do not introduce any traffic participants in this environment intentionally as we intend to show the effects of cleanly bi-modal distributions on the learning algorithms in an environment more complicated than the point-mass environments.

Block-push environment: We use a simulated environment similar to Multimodal Push environment described in [25]. We take the environment implementation directly from the PyBullet [15] based implementation provided by Florence et al. [25] in `https://github.com/google-research/ibc/tree/master/environments`.

In our environment, an XArm robot is situated in front of two blocks in a 0.75×1 plane. On the plane there are also two square targets. The goal of the agent is to push the blocks inside of the squares. However, the exact order of the block being pushed, or the combination of which block is pushed in which square doesn't matter. A block is considered successfully pushed if the center of the block is less than 0.05 away from a square.

On initialization, the blocks' positions are randomly shifted within a rectangle of side lengths $(0.2, 0.3)$, while the squares are randomly shifted within a rectangle of size $(0.01, 0.015)$. Additionally, the blocks were rotated at an uniformly arbitrary angle, while the target squares were rotated at an angle between $(\frac{\pi}{6}, -\frac{\pi}{6})$.

The demonstrations in this environments were collected with a hard-coded controller. There are two modes of multimodality inherent in the controller generated demonstartions. The controller:

1. Selects a block to start pushing first,
2. At the same time, independently chooses a target for the block to be pushed into.
3. Once the first block is pushed to a target, it pushes the second block to the remaining target.

Thus combinatorially, the controller is capable of four different modes of behavior. There are additional stochasticity in the controller behavior since there are many ways of pushing the same block into the same target.

The controller pushes the blocks to their targets following specific behavior primitives, such as moving to origin position, moving to a place collinear with a block and its target, and making a straight motion from that position towards the target unless the block rotates too much from its starting position.

Our models were trained on 1,000 demonstrations, all generated from the controller under the above randomized modes.

Franka kitchen environment: For the final set of experiments, we use the Franka Kitchen environment originally introduced in the Relay Policy Learning [34] paper. In that paper, the authors introduce a virtual kitchen environment where human participants in VR manipulated seven different objects in the kitchen: one kettle, one microwave, one sliding door, one hinged door, one light switch, and two burners. In total, we use 566 demonstrations collected by the researchers in that paper, where in each demonstration episode, each participant performed four manipulation task specified by the researchers in advanced.

The manipulator agent in simulator is a Franka Emika Panda robot, which is controlled through a 9-dimensional action space controlling the robot's joint and end-effector position. The 60-dimensional observation space is split into two parts, the first 30 dimension contains information about the current position of the interesting factors in the environment, while the last 30 dimensions contain information about the goal of the demonstrator or the agent. Note that in our demonstrations and our environments, we zero out the last 30 dimensions in all cases since we assume goal is not labelled in the demonstrations and is not specified in the unconditioned rollouts of the model.

One thing to note that, while the D4RL [26] paper also has three versions of the dataset, we chose to use the original version of the collected data from the Relay Policy Learning [34] paper. That is because the relay policy learning dataset is not labeled with intended tasks of the participants or rewards, while the D4RL dataset is geared towards that.

B Implementation Details and Hyperparameters

B.1 Baselines

Multi-layer Perceptron with MSE For our MLP with MSE baselines, we trained fully connected neural networks with optionally BatchNorm layers. In each of our environment, we varied the depth and the width of the MLPs to fit them best according to the bias-variance trade-off, while training them on 95% of the dataset and testing on the remaining 5% on the dataset in terms of MSE loss.

Nearest Neighbor Nearest Neighbor is conceptually the simplest baseline we show in this paper. During training, our Nearest Neighbor model simply stores all the (o, a) pairs. During test time, given a query observation, o, we find the observation o' with the minimum Euclidean distance to that in the representation space, and execute the associated action a' in the environment.

While it is a simple baseline, we show that it has a surprisingly high effectiveness in simple environments like CARLA, or dense environments like Kitchen where there is less of a chance in going OOD simply by executing seen actions. On the other hand, in environments like Block-push where the model needs to interpolate or extrapolate more, the NN model fails more.

k-Nearest Neighbor with Locally Weighted Regression A slightly more robust version of NN for regression problems, k-NN with locally weighted regression or LWR, is the next baseline we use. In this baseline, we take the k-nearest neighbors (in all our cases, 5) in the observation representation space, and take a weighted average of their associated actions. The weighting is based on the negative exponent of the distance, or namely, $\exp -||o - o'||$, as seen in [62]. This model is better than simple Nearest Neighbors in interpolations, and thus we see a higher success in the Kitchen environment.

Continuous Generative Model: VAE with Gaussian Prior Following prior works[66], we use variational auto-encoders (VAE) for encoding and decoding sequences of actions into a smaller latent space. The VAE here learns to compress a sequence of $T = 10$ actions into a single latent variable z of 10 dimensions. The hyperparameters for training the VAE has been taken directly from Pertsch et al. [66].

Concurrently with training the VAE, we train a state-conditioned latent prior model that tries to predict $P(z \mid o)$. This latent generator produces a vector of μ and σ which is sampled to find latent z, and we feed a Gaussian distributed variable z back into the decoder network where the action sequence is reconstructed. For the current observation o_t, sequence of reconstructed actions $a_t, \cdots, a_{t+9}$ are performed in a simulated environment.

The design choices of this algorithm has been heavily inspired by [66]. Although this model shows promise in theory, we found in practice that unconditional rollout from this model is not very successful. We believe the shortcoming is a result of random sampling from the z space that does not take into account the recently executed actions, and using a single-mode Gaussian as the state prior similar to [66], and thus this baseline is only slightly better than the MLP-MSE model.

Continuous Generative Models: Normalizing Flow with and without Prior Similar to Singh et al. [76], we use a Normalizing Flow [20] based generative model. We follow the architectural choices and the hyperparameters from [76] in our baseline implementation.

Our observation-conditioned Flow model is trained on the distribution $P(a \mid o)$ to continuously transform it into an identity Gaussian distribution of the same dimensions as a. To find a better prior than simply an identity Gaussian, we also trained a prior model that generates μ, σ of a Gaussian distribution given the observation o. We found that the prior improves the quality of the rollouts, however slightly.

We believe the under-performance of these continuous generative approaches were based on two major problems. One is that they fail to take historical context in concern, and by being a continuous distribution, returned less likely actions that led to more rollouts going OOD. Second, they were designed with a focus of making RL approachable by compressing the action space, which requires having a prior that is not so strict. However, most of BeT's performance comes from having a strong prior over the actions, which is only augmented by the action offset prediction.

Implicit Behavioral Cloning Implicit Behavioral Cloning (IBC) [25] takes a different approach in behavioral cloning, where instead of learning a model $f(o) := a$, we learn an energy based model $E(o, a)$ where the intended action a at any observation is defined as $\arg\min_a E(o, a)$. While this suffers from all the classic issues of training an EBM, like higher sample complexity and higher complexity in sampling, IBC models have been shown to have higher success in learning multi-modal and discontinuous actions.

As a baseline, we use the official implementation provided in `https://github.com/google-research/ibc` For the CARLA environment, we use equivalent hyperparameters from the "pushing from pixels" hyperparameters. For the Block-pushing environment, we use the "pushing from states" hyperparameters. Finally, for the Kitchen environment, we use the "D4RL kitchen" hyperparameters.

While IBC is our strongest baseline, in our experience it is also one that is quite easy to overfit to our datasets. As a result, we monitored test performance over the training and had to employ early stopping for both the CARLA and the Block-pushing tasks.

Trajectory Transformers Trajectory Transformers [39], especially the variant that is trained without any rewards only on states and actions from demonstrations, seem similar to our approach, there are a few crucial differences. While we agree that BeT and Trajectory Transformer based behavior cloning both use some type of discretization to fit demonstration datasets with a minGPT, we believe that is where the similarities end. The primary differences between the algorithms is in our design choices: namely what distributions they model, and consequently how they treat the observations. The differences are explained more thoroughly below.

- **Modeled distribution**: From a provided set of demonstrations, trajectory transformers model the joint distribution P(action, observations). On the other hand, BeT models the conditional distribution P(action | observations). Modeling the joint distribution requires MinGPT to model the forward dynamics of the environment, which can be arbitrarily difficult based on the environment.
- **Observation discretization**: Because trajectory transformers have to model the observations as well, it needs to discretize the observation space. As a result, TT cannot extend to high dimensional observational spaces, such as visual observations. This limitation is also acknowledged by the authors of Trajectory Transformers. BeT, on the other hand, does not model the observations and thus does not need to discretize them. Thus BeT can scale to arbitrarily high dimensional observations, as we show in the CARLA environment experiments, where BeT learns behaviors from high dimensional visual observations.
- **Efficient historical encoding**: Trajectory transformer encodes each (state, action) pair into a total of $|S| + |A|$ input/output tokens, while BeT encodes them into one input/output token. On a base MinGPT implementation that means a $O((|S| + |A|)^2)$ efficiency gain for BeT, or for example 4761x less compute for the same historic context in the Kitchen environment.

As a baseline, we trained and rolled out Trajectory Transformer on the Kitchen environment. It failed to complete any tasks for unconditioned, greedy, or beam search rollouts. We would like to note that the Kitchen environment is more complicated than the MuJoCo environments (HalfCheetah, Hopper, Walker2d, and Ant) that the paper experimented on. At the same time, this environment has an order of magnitude fewer samples on the training set (10^6 vs. approximately 120k). We tried both our own implementation and the implementation from `https://github.com/Howuhh/faster-trajectory-transformer` with the recommended parameters for the AntMaze environment, which is the largest environment used by the authors.

B.2 Algorithm Details

Loss function details: In this paper, we use two loss functions that are inspired by practices in computer vision, in particular object detection. The first of them is the Focal loss [49], and the second one is the Multi-task loss [30].

The Focal loss is a simple modification over the cross entropy loss. While the normal cross entropy loss for binary classification can be thought of $\mathcal{L}_{ce}(p_t) = -\log(p_t)$, the Focal loss adds a term $(1 - p_t)^\gamma$ to this, to make the new loss

$$\mathcal{L}_{focal}(p_t) = -(1 - p_t)^\gamma \log(p_t)$$

This loss has the interesting property that its gradient is more steep for smaller values of p_t, while flatter for larger values of p_t. Thus, it penalizes and changes the model more for making errors in the low-probability classes, while is more lenient about making errors in the high probability classes. Using this error in the object detection world has helped with class imbalance between different classes, and here it helps BeT learn to predict different k-means from the dataset even if their appearance in the dataset is not completely balanced.

For the multi-task loss, we use the formulation

$$\text{MT-Loss}\left(\mathbf{a}, \left(\langle \hat{a}_i^{(j)} \rangle\right)_{j=1}^{k}\right) = \sum_{j=1}^{k} \mathbb{I}[\lfloor \mathbf{a} \rfloor = j] \cdot \|\langle \mathbf{a} \rangle - \langle \hat{a}^{(j)} \rangle\|_2^2$$

This helps us penalize only the offset for the ground truth class, thus making sure the MinGPT is not trying to predict the right action offset through all classes and instead only trying to predict the action offset through the right class.

In practice, we optimize the combined loss, $\mathcal{L}_{focal} + \alpha \mathcal{L}_{mt}$ while α is a hyperparameter that just makes sure at initialization the two losses are of the same order of magnitude.

Compute details: All of our code was run in a single NVIDIA RTX 3080 GPU for state-based environments and RTX 8000 for image-based environments.

Performance measurement details: We measured the performance reported in the Section 3.5 in an NVIDIA RTX 3080 machine with AMD Threadripper 5950x CPUs. We took the average over three runs to minimize inter-run variances, and measured wall-clock time to report in the paper.

In terms of raw computation time to determine one action from the observations, in the Kitchen environment, BeT took 2.8 ms, while IBC took 52 ms and MLP, as the fastest point of comparison, took 0.5 ms. On the same environment, a single step of Trajectory Transformer took 867.86 ms, on an implementation that used more advanced tricks such as attention caching.

Hyperparameters list: We present the BeT hyperparameters in Table 4 below:

Table 4: Environment-dependent hyperparameters in BeT.

Hyperparameter	Point-mass	CARLA	Block-push	Kitchen
Layers	1	3	4	6
Attention heads	2	4	4	6
Embedding width	20	256	72	120
Dropout probability	0.1	0.6	0.1	0.1
Context size	2	10	5	10
Training epochs	10	40	350	50
Batch size	64	128	64	64
Number of bins k	2; 3	32	24	64

However, we have found that as long as the model does not overfit, a wide range of parameters all yield favorable results for BeT; thus, this table should be taken as reference values for reproducing our results rather than the only parameter sets that work.

Apart from that, we have some hyperparameters that are shared across all BeT experiments. They are reproduced in Table 5.

Table 5: Shared hyperparameters for BeT training

Name	Value
Optimizer	Adam
Learning rate	1e-4
Weight decay	0.1
Betas	(0.9, 0.95)
Gradient clip norm	1.0

B.3 Pseudocode

See the pseudocode described on Algorithm 1.

B.4 Architecture and Implementation

For our implementation, we used the MinGPT [41] repository almost as-is. We modified the input token conversion layer to a linear projection layer to handle our continuous, instead of discrete, inputs. Apart from that, we followed the MinGPT architecture quite exclusively, with successive attention layers with a number of attention head and embedding dimensions. Between the layers, we used dropout regularization same as [41].

Algorithm 1 Learning Behavior Transformer from a dataset of behavior sequences.

```
Input: Dataset $(o_{t,i}, a_{t,i})_{t,i}$ for $0 \leq i \leq$ number of demonstrations, $0 \leq t \leq$ maximum episode lengths, intended number of clusters $k$ and context history length $h$.

Initialize: $\theta_M$ the parameters for MinGPT, $\{A_i\}_{i=1}^k$ cluster centers randomly in the action space.

Learn k-means encoder/decoder:
Using all possible $a_{t,i}$, learn the $k$ cluster centers using the $k$ means algorithm.
Set $\{A_i\}_{i=1}^k$ as the learned cluster centers.

Define functions:
$\lfloor a \rfloor := \arg\min_{i=1}^k ||a - A_i||$
$\langle a \rangle := a - \lfloor a \rfloor$
$\text{Enc}(a) := (\lfloor a \rfloor, \langle a \rangle)$
$\text{Dec}(\lfloor a \rfloor, \langle a \rangle) := A_{\lfloor a \rfloor} + \langle a \rangle$

Train MinGPT trunk of BeT:
while Not converged do
    Sample trajectory subsequence $(o_t, a_t), \cdots, (o_{t+h-1}, a_{t+h-1})$ from the dataset.
    Feed in the observations $(o_t, o_{t+1}, \cdots, o_{t+h-1})$ into the MinGPT.
    Get categorical distribution probabilities $p_{\tau,c}$ for $t \leq \tau \leq t+h-1, 1 \leq c \leq k$.
    Compute focal loss $\mathcal{L}_{ce}$ of $p_{\tau,c}$ against ground truth class $\lfloor a_\tau \rfloor$, for all $\tau, c$ .
    Get the residual action offset per class, $\langle a_{\tau,c} \rangle$, for all $\tau, c$ from MinGPT.
    Calculate the multi-task loss, $\mathcal{L}_{mt}$, against true class predicted offset, $\sum_\tau ||\langle a_{\tau,\lfloor a_\tau \rfloor} \rangle - \langle a_\tau \rangle||_2^2$
    Backprop using the normalized loss, $\mathcal{L}_{ce} + \alpha\mathcal{L}_{mt}$ where $\alpha$ makes the losses of equal magnitude.

Running on the environment:
while Episode not completed do
    Stack the last $h$ observations in the environment, $(o_t, o_{t+1}, \cdots, o_{t+h-1})$ and feed into MinGPT.
    Get categorical probabilities $p_{\tau,c}$ for $t \leq \tau \leq t+h-1, 1 \leq c \leq k$ from the MinGPT.
    Sample a class $c$ from $p_{t+h-1,c}$ for $1 \leq c \leq k$.
    Get the associated action offset, $\langle a_{t+h-1,c} \rangle$ from the MinGPT.
    Decode into full continuous action, $\overline{a}_{t+h-1} := \text{Dec}(c, \langle a_{t+h-1,c} \rangle)$
    Execute decoded action $\overline{a}_{t+h-1}$ into environment.
```

For the smallest tasks, like point-mass environments, we used models with approximately 10^4 parameters, which went up to around 10^6 for Kitchen environments.

C Ablation studies

In this section, we provide more details about the ablation studies presented in the main paper, as well as present detailed plots of our ablation studies that compare different versions of the BeT architecture.

C.1 Ablating historical context

One of the reasons why we used transformer-based generative networks in our work is because of our hypothesis that having historical context helps our model learn better behavioral cloning. Our experiments are performed by using the same model and simply providing sequences of length one on training and test time. As we can see on Sec. 3.5, having some historical context helps our model learn much better.

C.2 Ablating the number of discrete bin centers, k

Since BeT is trained with a sum of focal loss for the binning head and MSE loss for the offset head, the number of cluster centers present a trade-off in the architecture. Concretely, as the number of bins go up, the log-likelihood loss goes up but the MSE loss goes down. In Sec. 3.5, we showed that using only one bin ($k = 1$) decreases the performance level of BeT.

In this section, we present the plot of the variation in performance as k value changes.

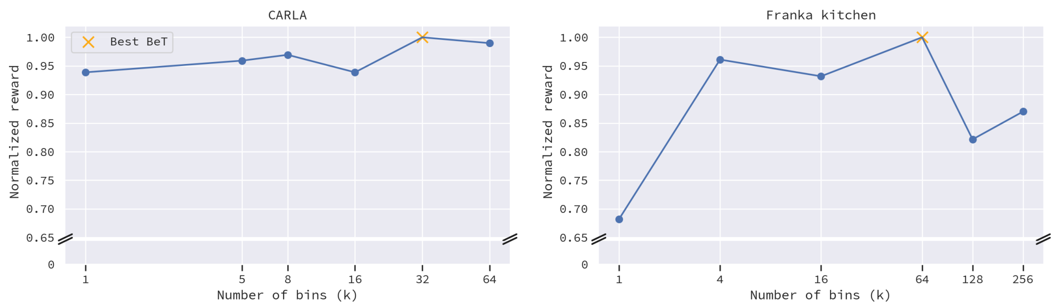

Figure 6: Ablating the number of discrete bin centers k for BeT. Reward is normalized with respect to the best performing model.

C.3 Ablating the core model in the architecture

To ablate the core MinGPT transformer model in the architecture, we run three ablations, where we replace it respectively with a fully-connected multi-layer perceptron (MLP) network, a temporal convolution network, and an LSTM-based recurrent neural network.

Multi-Layer Perceptrons: Since generally MLP networks are not capable of taking in historical context in consideration, we instead stack the last t frames of observation to pass into the MLP network. Near the beginning of a trajectory, the stack of observation is zero-padded to t frames. For the intermediate layers in the MLP, we keep the same width and the number of layers as the corresponding MinGPT.

Temporal Convolution: Convolutions over the sequence length has been used in numerous prior works [60, 40, 17, 28, 5] for sequence modeling. As a baseline, we implement such temporal convolutional network to replace our MinGPT-based trunk. We perform a temporal convolution over the same period of history that is provided to our transformer models. We found that the performance of the temporal convolution models are constantly lower than our MinGPT based models. However, temporal convolutional networks are easier to fit on our data compared to RNNs.

LSTM-based RNN: Recurrent neural networks (RNNs) were the previous state-of-the-art for sequence modeling before transformer-based models. In this work, we compare against an Long-short term memory (LSTM) [29] based RNN instead of a transformer based trunk. We find that even with sufficient model capacity, the RNN based model took significantly longer than our MinGPT model to fit the same dataset. Moreover, the quality of fit was worse, both in training and test time. Finally, in open-ended rollouts, this performance downgrade is reflected in a far lower success rate for completing tasks in the environment (Table. 3).