Behavior Generation with Latent Actions

Seungjae Lee[1,2], Yibin Wang[1], Haritheja Etukuru[1], H. Jin Kim[2], Nur Muhammad Mahi Shafiullah[1,*], Lerrel Pinto[1,*]
[1]New York University [2]Seoul National University
https://sjlee.cc/vq-bet

***Abstract*—Generative modeling of complex behaviors from labeled datasets has been a longstanding problem in decision-making. Unlike language or image generation, decision-making requires modeling actions – continuous-valued vectors that are multimodal in their distribution, potentially drawn from uncurated sources, where generation errors can compound in sequential prediction. A recent class of models called Behavior Transformers (BeT) addresses this by discretizing actions using k-means clustering to capture different modes. However, k-means struggles to scale for high-dimensional action spaces or long sequences, and lacks gradient information, and thus BeT suffers in modeling long-range actions. In this work, we present Vector-Quantized Behavior Transformer (VQ-BeT), a versatile model for behavior generation that handles multimodal action prediction, conditional generation, and partial observations. VQ-BeT augments BeT by tokenizing continuous actions with a hierarchical vector quantization module. Across seven environments including simulated manipulation, autonomous driving, and robotics, VQ-BeT improves on state-of-the-art models such as BeT and Diffusion Policies. Importantly, we demonstrate VQ-BeT's improved ability to capture behavior modes while accelerating inference speed 5$\times$ over Diffusion Policies.**

I. Introduction

The presently dominant paradigm in modeling human outputs, whether in language [1], image [38], audio [57], or video [2], follows a similar recipe: collect a large in-domain dataset, use a large model that fits the dataset, and possibly as a cherry on top, improve the model output using some domain-specific feedback or datasets. However, such a large, successful model for generating human or robot actions in embodied environments has been absent so far, and the issues are apparent. Action sequences are semantically diverse but temporally highly correlated, human behavior distributions are massively multi-modal and noisy, and the hard-and-fast grounding in the laws of physics means that unlike audio, language or video-generation, even the smallest discrepancies may cause a cascade of consequences that lead to catastrophic failures in as few as tens of timesteps [42, 39]. The desiderata for a good model of behaviors and actions thus must contain the following abilities: to model long- and short-term dependencies, to capture and generate from diverse modes of behavior, and to replicate the learned behaviors precisely [43, 9].

Prior work by [43] shows how transformers can capture the temporal dependencies well, and to some extent even capture the multi-modality in the data with clever tokenization. However, that tokenziation relies on k-means clustering, a method typically based on an ℓ_2 metric space that unfortunately does not scale to high-dimensional action spaces or temporally extended actions with lots of inter-dependencies. More recent works have also used tools from generative modeling to address the problem of behavior modeling [36, 9, 56], but issues remain, for example in high computational cost when scaling to long-horizons, or failing to express multi-modality during rollouts.

In this work, we propose Vector-Quantized Behavior Transformer (VQ-BeT), which combines the long-horizon modeling capabilities of transformers with the expressiveness of vector-quantization to minimize the compute cost while maintaining high fidelity to the data. We posit that a large part of the difficulty in behavior modeling comes from representing the continuous-valued, multi-modal action vectors. A ready answer is learning discrete representations using vector quantization [49] used extensively to handle the output spaces in audio [12], video [52], and image [41]. In particular, the performance of VQ-VAEs for generative tasks has been so strong that a lot of recent models that generate continuous values simply generate a latent vector in the VQ-space first before decoding or upsampling the result [57, 2, 38].

VQ-BeT is designed to be versatile, allowing it to be readily used in both conditional and unconditional generation, while being performative on problems ranging across simulated manipulation, autonomous driving, and real-robotics. Through extensive experiments across eight benchmark environments, we present the following experimental insights:

1) VQ-BeT achieves state-of-the-art (SOTA) performance on unconditional behavior generation outperforming BC, BeT, and diffusion policies in 5/7 environments (Figure 1 middle). Quantitative metrics of entropy and qualitative visualizations indicate that this performance gain is due to better capture of multiple modes in behavior data (Figure 1 left).
2) On conditional behavior generation, by simply specifying goals as input, VQ-BeT achieves SOTA performance and improves upon GCBC, C-BeT, and BESO in 6/7 environments (Figure 1 right).
3) VQ-BeT directly works on autonomous driving benchmarks such as nuScenes [7], matching and being comparable to task-specific SOTA methods.
4) VQ-BeT is a single-pass model, and hence offers a 5$\times$ speedup in simulation and 25$\times$ on real-world robots over multi-pass models that use diffusion models.
5) VQ-BeT scales to real-world robotic manipulation such

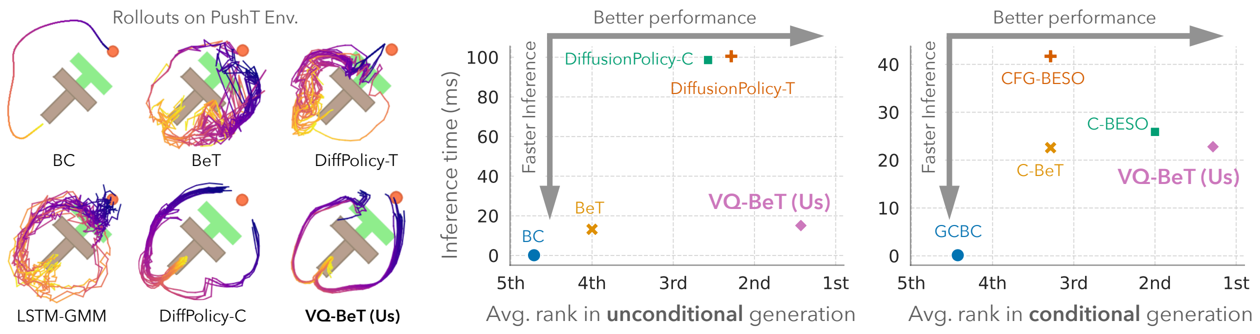


Fig. 1: Qualitative and quantitative comparison between VQ-BeT and relevant baselines. On the left, we can see trajectories generated by different algorithms while pushing a T-block to target, where VQ-BeT generates smooth trajectories covering both modes. On the right, we show two plots comparing VQ-BeT and relevant baselines on unconditional and goal-conditional behavior generation. The comparison axes are (x-axis) relative success represented by average rank on a suite of seven simulated tasks, and (y-axis) inference time.

as pick-and-placing objects and door closing, improving upon prior work by 73% on long-horizon tasks.

II. Background and Preliminaries

A. Behavior cloning

Given a dataset of continuous-valued action and observation pairs $\mathscr{D} = \{(o_t, a_t)\}_t$, the goal of behavior cloning is to learn a mapping π from observation space $\mathscr{O}$ to the action space $\mathscr{A}$. This map is often learned in a supervised fashion with π as a deep neural network minimizing some loss function $\mathscr{L}(\pi(o), a)$ on the observed behavior data pairs $(o, a) \in \mathscr{D}$. Traditionally, $\mathscr{L}$ was simply taken as the MSE loss, but its inability to admit multiple modes of action for an observation led to different loss formulations [30, 14, 43, 9]. Similarly, understanding that the environment may be partially observable led to modeling the distribution $\mathbb{P}(a_t \mid o_{t-h:t})$ rather than $\mathbb{P}(a_t \mid o_t)$. Finally, understanding that such behavior datasets are often generated with an explicit or implicit goal, many recent approaches condition on an (implicit or explicit) goal variable g and learn a goal-conditioned behavior $\mathbb{P}(a \mid o, g)$. Note that such behavior datasets crucially do not contain any "reward" information, which makes this setup different from reward-conditioned learning as a form of offline RL.

B. Behavior Transformers

Behavior transformer (BeT) [43] and conditional behavior transformer (C-BeT) [10] are respectively two unconditional and goal-conditional behavior cloning algorithms built on top of GPT-like transformer architectures. In their respective settings, they have shown the ability to handle temporal correlations in the dataset, as well as the presence of multiple modes in the behavior. While GPT [5] itself maps from discrete to discrete domains, BeT can handle multi-modal continuous output space by a clever tokenization trick. Prior to training, BeT learns a k-means based encoder/decoder that can convert continuous actions into one discrete and one continuous component. Then, by learning a categorical distribution over the discrete component and combining the component mean with a predicted continuous "offset" variable, BeT can functionally learn multiple modes of the data while each mode remains continuous. While the tokenizer allows BeT handle multi-modal actions, the use of k-means means that choosing a good value of k is important for such algorithms. In particular, if k is too small then multiple modes of action gets delegated to the same bin, and if k is too large one mode gets split up into multiple bins, both of which may result in a suboptimal policy. Also, when the action has a large number of (potentially correlated) dimensions, for example when performing action chunking [56], non-parametric algorithms like k-means may not capture the nuances of the data distribution. Such shortcomings of the tokenizer used in BeT and C-BeT is one of the major inspirations behind our work.

C. Residual Vector Quantization

In order to tokenize continuous action, we employ Residual Vector Quantization (Residual VQ) [54] as a discretization bottleneck. Vector quantization is a quantization technique where continuous values are replaced by a finite number of potentially learned codebook vectors. This process maps the input x to an embedding vector z_q in the codebook $\{e_1, e_2, \cdots e_k\}$ by the nearest neighbor look-up:

$$z_q = e_c, \quad \text{where } c = \text{argmin}_j ||x - e_j||_2. \tag{1}$$

Residual VQ is a multi-stage vector quantizer [50] which replaces each embedding of vanilla VQ-VAE [49] with the sum of vectors from a finite layers of codebooks. This approach cascades N_q layers of vector quantizations residually: the input vector x is passed through the first stage of vector quantization to derive z_q^1. The residual, $x - z_q^1$, is then iteratively quantized by a sequence of $N_q - 1$ quantizing layers, passing the updated residual $x - \sum_{i=1}^{p} z_q^i$ to the next layer. The final quantized input vector is then the sum of vectors from a set of finite codebooks $z_q(x) = \sum_{i=1}^{N_q} z_q^i$.

III. Vector-Quantized Behavior Transformers

In this section, we introduce VQ-BeT, which has capability to solve both conditional and non-conditional tasks from uncurated behavior dataset. VQ-BeT is composed of two stages: Action discretization phase (stage 1 in Figure 2) and VQ-BeT learning phase (stage 2 in Figure 2). Each stage is explained in Section III-B and III-C, respectively.

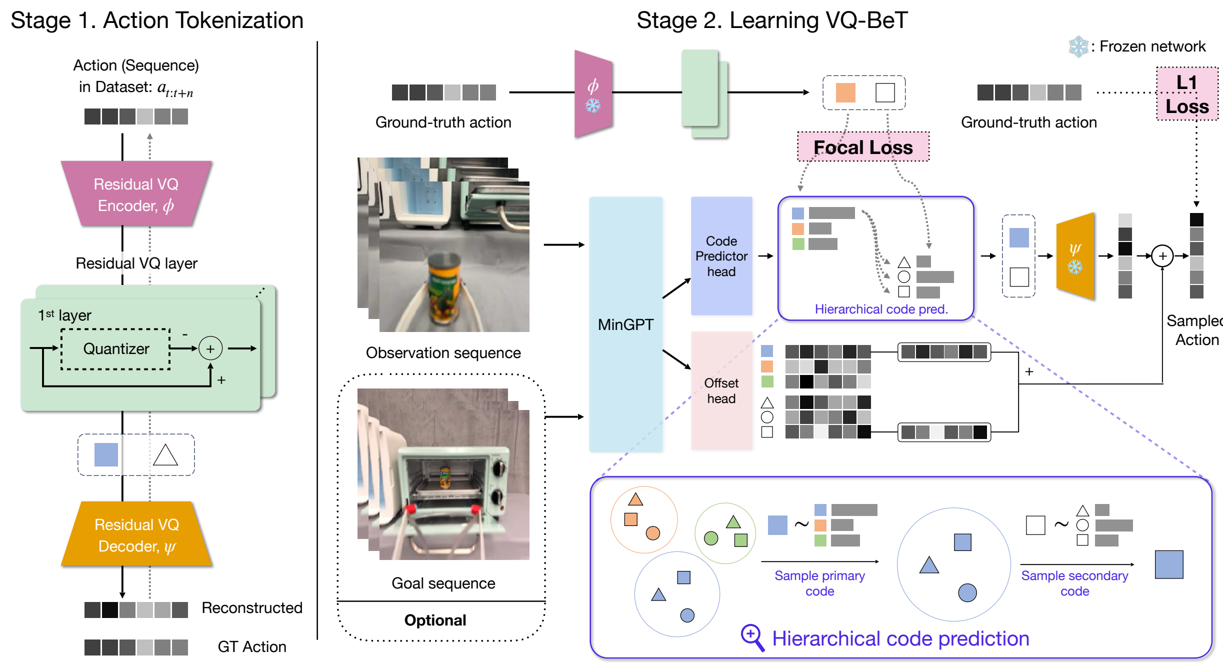


Fig. 2: Overview of VQ-BeT, broken down into the residual VQ encoder-decoder training phase and the VQ-BeT training phase. The same architecture works for both conditional and unconditional cases with an optional goal input. In the bottom right, we show a detailed view of the hierarchical code prediction method.

A. Sequential prediction on behavior data

Binning actions to tokenize them and predicting the tokenized class has been successfully applied for learning multi-modal behavior [43, 10]. However, these k-means binning approaches face issues while scaling, as disucssed in Section II-B.

As such, we propose instead to learn a discrete latent embedding space for action or action chunks, and modeling such action latents instead. Note that, such latent models in the form of VQ-VAEs and latent diffusion models are widely used in multiple generative modeling subfields, including image, music, and video [2, 57, 38]. With such discrete tokenziation, our model can directly predict action tokens from observation sequences optionally conditioned on goal vectors.

B. Action (chunk) discretization via Residual VQ

We employ Residual VQ-VAE [54] to learn a scalable action discretizer and address the complexity of action spaces encountered in the real world. The quantization process of an action (or action chunk, where $n > 1$) $a_{t:t+n}$ is learned via learning a pair of encoder and decoder networks; ϕ, ψ. We start with passing $a_{t:t+n}$ through the encoder ϕ. The resulting latent embedding vector $x = \phi(a_{t:t+n})$ is then mapped to an embedding vector in the codebook of the first layer $z_q^1 \in \{e_1^1, \cdots e_k^1\}$ by the nearest neighbor look-up, and the residual is recursively mapped to each codebook of the remaining $N_q - 1$ layers $z_q^i \in \{e_1^i, \cdots e_k^i\}$, where $i = 2, \cdots, N_q$. The latent embedding vector $x = \phi(a_{t:t+n})$ is represented by the sum of vectors from codebooks $z_q(x) = \sum_{i=1}^{N_q} z_q^i$, where each vector $z_q^{i=1:N_q}$ works as the centroid of hierarchical clustering.

Then, the discretized vector $z_q(x) = \sum_{i=1}^{N_q} z_q^i$ is reconstructed as $\psi(z_q(x))$ by passing through the decoder ψ. We train Residual VQ-VAE using a loss function, as shown in Eq 3. The first term represents the reconstruction loss, and the second term is the VQ objective that shifts the embedding vector e towards the encoded action $x = \phi(a_{t:t+n})$. To update the embedding vectors $e_{1:k}^{1:N_q}$, we use moving averages rather than direct gradient updates following [22, 34]. In all of our experiments, it was sufficient to use $N_q := 2$ VQ-residual layers, and keep the commitment loss $\lambda_{\text{commit}} := 1$ constant.

$$\mathcal{L}_{\text{Recon}} = \left\| a_{t:t+n} - \psi(z_q(\phi(a_{t:t+n}))) \right\|_1 \tag{2}$$

$$\begin{aligned}\mathcal{L}_{\text{RVQ}} =& \mathcal{L}_{\text{Recon}} + \|\text{SG}[\phi(a_{t:t+n})] - e\|_2^2 \\ &+ \lambda_{\text{commit}} \|\phi(a_{t:t+n}) - \text{SG}[e]\|_2^2, \quad (\text{SG}: \text{stop gradient})\end{aligned} \tag{3}$$

We indicate the codes of the first quantizer layer as *primary code*, and the codes of the remaining layers as *secondary codes*. Intuitively, the primary codes in Residual VQ performs coarse clustering over a large range within the dataset, while the secondary codes handle fine-grained actions. (Decoded centroids are visualized in Appendix Figure 8.)

C. Weighted update for code prediction

After training Residual VQ, we train GPT-like transformer architecture to model the probability distribution of action or action chunks from the sequence of observations. One of the main differences between BeT and VQ-BeT stems from using a learned latent space.

Since our vector quantization codebooks let us freely translate between an action latent $z_q(\phi(a_{t:t+n})) = \sum_{i=1}^{N_q} z_q^i$ and the sequence of chosen codes at each codebook, $\{z_q^i\}_{i=1}^{N_q}$, we

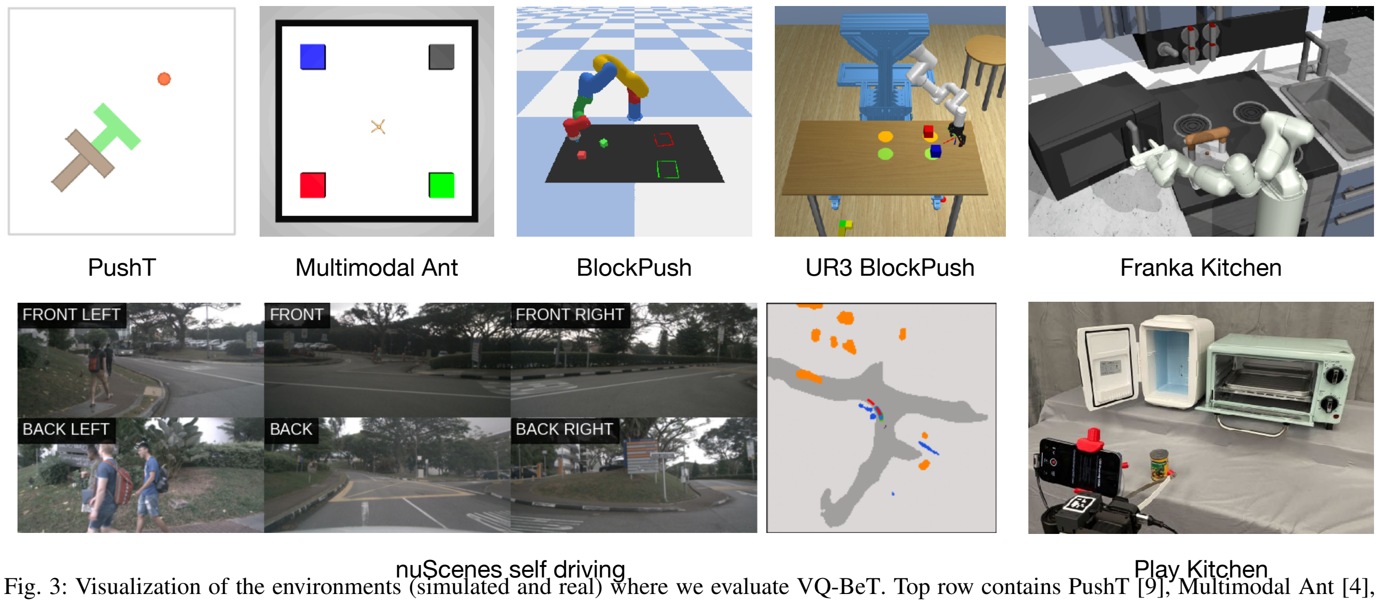


Fig. 3: Visualization of the environments (simulated and real) where we evaluate VQ-BeT. Top row contains PushT [9], Multimodal Ant [4], BlockPush [14], UR3 BlockPush [27], Franka Kitchen [15], and bottom row contains nuScenes self-driving [7], and our real robot environment.

Environment	Metric	GCBC	C-BeT	C-BESO	CFG-BESO	VQ-BeT
PushT	Final IoU	0.02	0.02	0.30	0.25	**0.39**
Image PushT	$(\cdot/1)$	0.02	0.01	0.02	0.01	**0.10**
Kitchen	Goals	0.15	3.09	3.75	3.47	**3.78**
Image Kitchen	$(\cdot/4)$	0.64	2.41	2.00	1.59	**2.60**
Multimodal Ant	Goals	0.00	1.68	1.14	0.92	**1.72**
UR3 BlockPush	$(\cdot/2)$	0.19	1.67	**1.94**	1.91	**1.94**
BlockPush	Success $(\cdot/1)$	0.01	0.87	**0.93**	0.88	0.87

TABLE I: Comparing different algorithms in goal-conditional behavior generation. The seven simulated robotic manipulation and locomotion environments used here are described in Section IV-A.

Environment	Metric	BC	BeT	DiffPolicy-C	DiffPolicy-T	VQ-BeT
PushT	Final IoU	0.65	0.39	0.73	0.74	**0.78**
Image PushT	$(\cdot/1)$	0.13	0.01	0.66	0.45	**0.68**
Kitchen	Goals	0.18	3.07	2.62	3.44	**3.66**
Image Kitchen	$(\cdot/4)$	0.75	2.48	**3.11**	3.01	2.98
Multimodal Ant		0.01	2.73	3.12	2.90	**3.22**
UR3 BlockPush	Goals	0.11	1.59	1.83	1.82	**1.84**
BlockPush	$(\cdot/2)$	0.01	1.67	0.47	**1.93**	1.79

TABLE II: Performance of different algorithms in unconditional behavior generation tasks. We evaluate over seven simulated robotic manipulation and locomotion tasks as described in Section IV-A.

use them as a labels in the code prediction $\mathscr{L}_{\text{code}}$ loss to learn the categorical prediction head ζ^i_{code} for given sequence of observations $o_{t-h:t}$. Following [43, 10], we employ Focal loss [28] to train the code prediction head by comparing the probabilities of the predicted categorical distribution with the actual labels z^i_q. We adjust the weights between the primary code and secondary code learning losses, leveraging our priors about the latent space.

$$\mathscr{L}_{\text{code}} = \mathscr{L}_{\text{focal}}(\zeta^{i=1}_{\text{code}}(o_t)) + \beta \mathscr{L}_{\text{focal}}(\zeta^{i>1}_{\text{code}}(o_t)) \tag{4}$$

Finally, the quantized behavior is obtained by passing the sum of the predicted residual embeddings through the decoder as follows.

$$\lfloor a_{t:t+n} \rfloor = \psi\Big(\sum_{j,i} e^i_j \cdot \mathbb{I}[\zeta^i_{\text{code}} = j)]\Big) \tag{5}$$

We adopt additional offset head ζ_{offset} to maintain full fidelity, adjusting the centers of discretized actions based on observations. The total VQ-BeT loss is shown in Eq. 7.

$$\mathscr{L}_{\text{offset}} = \Big| a_{t:t+n} - \big(\lfloor a_{t:t+n} \rfloor + \zeta_{\text{offset}}(o_t) \big) \Big|_1 \tag{6}$$

$$\mathscr{L}_{\text{VQ-BeT}} = \mathscr{L}_{\text{code}} + \mathscr{L}_{\text{offset}} \tag{7}$$

D. *Conditional and non-conditional task formulation*

To provide a general-purpose behavior-learning model that can predict multi-modal continuous actions in both conditional and unconditional tasks, we introduce conditional and non-conditional task formulation of VQ-BeT.

a) Non-conditional formulation: For a given dataset $\mathscr{D} = \{o_t, a_t\}$, we consider a problem of predicting the distribution of possible action sequences $a_{t:t+n}$ conditioned on a sampled sequence of observations $o_{t-h:t}$. Thus, we formulate the behavior policy as $\pi : \mathscr{O}^h \to \mathscr{A}^n$, where $\mathscr{O}$ and $\mathscr{A}$ denotes the observation space and action space, respectively.

b) Conditional formulation: For goal-conditional tasks, we extend the formulation above to take a goal conditioning vector in the form of one or more observations. Given current observation sequence and future observation sequence, we now consider an extended policy model that predicts the distribution of sequential behavior $\pi : \mathscr{O}^h \times \mathscr{O}^g \to \mathscr{A}^n$, where $o_{t-h:t} \in \mathscr{O}^h$ and $o_{N-g:N} \in \mathscr{O}^g$ are current and future observation sequences.

IV. Experiments

With both conditional and unconditional VQ-BeT, we run experiments to understand how well they can model behavior on different datasets and environments. We focus on two primary properties of VQ-BeT's generated behaviors: quality, as evaluated by how well the generated behavior achieves some task objective or goal, and the diversity, as evaluated by the entropy of the distribution of accomplished subtasks or goals. Concretely, through our experiments, we try to answer the following questions:

1) How well do VQ-BeT policies perform on the respective environments in both conditional and unconditional behavior generation?
2) How well does VQ-BeT capture the multi-modality present in the dataset?
3) Does VQ-BeT scale beyond simulated tasks?
4) What design choices of VQ-BeT make the most impact in its performance?

A. *Environments, datasets, and baselines*

Across our experiments, we use a variety of environments and datasets to evaluate VQ-BeT (Figure 3). In simulation, we evaluate the wider applicability of VQ-BeT on eight benchmarks; namely, six manipulation tasks including two image-based tasks: (a) PushT, (b) Image PushT, (c) Kitchen, (d) Image Kitchen, (e) UR3 BlockPush, (f) BlockPush; a locomotion task, (g) Multimodal Ant; and a self-driving benchmark, (h) NuScenes. The environments are visualized in Figure 3, and a detailed descriptions of each task is provided in Appendix A. We also evaluate on a real-world environment with twelve tasks (five single-phase, three multi-phase tasks and four long-horizon tasks) described in Section IV-G.

a) Baselines: We compare VQ-BeT against the SOTA methods in behavior modeling in both conditional and unconditional categories. In both of these categories, we compare against transformer- and diffusion-based baselines.

For unconditional behavior generation, we compare against MLP-based behavior cloning, the original Behavior Transformers (BeT) [43] and Diffusion Policy [9]. The BeT architecture uses a k-means tokenization as explained in Section II-B. Diffusion policy [9], on the other hand, uses a denoising diffusion head [18] to model multi-modality in the behaviors. We use both the convolutional and transformer variant of the diffusion policy as baselines for our work since they excel in different cases.

For goal-conditional behaviors, we compare against simple goal conditioned BC, Conditional Behavior Transformers (C-BeT) [10] and BESO [40]. C-BeT uses k-means tokenization but otherwise has a similar architecture to ours. BESO uses denoising diffusion, and has a conditioned variant (C-BESO) and a classifier-free guided variant (CFG-BESO) that we compare against.

B. *Performance of behavior generated by VQ-BeT*

We evaluate VQ-BeT in a set of goal-conditional tasks in Table I and a set of unconditional tasks in Table II. On the PushT environments, we look at final and max coverage, where the coverage value is the IoU between the T block and the target T position. For the unconditional Kitchen, BlockPush, and

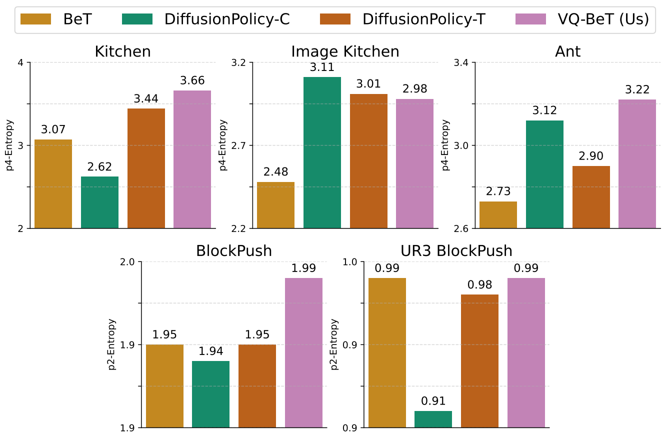


Fig. 4: A comparison between the behavior entropy of the algorithms, calculated based on their task completion order, on five of our simulated environments.

Ant tasks, we look at the total number of tasks completed in expectation, where the maximum possible number of tasks is 4, 2, and 4 respectively. For the conditional environments, we report the expected number of successes given a commanded goal sequence, where the numbers of commanded goals are 4 in Kitchen, 2 in Ant, and 2 in BlockPush. Across all of these metrics, a higher number designates a better performance.

From Tables I and II, we see that in both conditional and unconditional tasks, VQ-BeT largely outperforms or matches the baselines. First, on the conditional tasks, we find that VQ-BeT outperforms all baselines in all tasks except for BlockPush. In BlockPush, VQ-BeT performs on par with BeT, while C-BESO and CFG-BESO performs slighly better. Note that BlockPush has one of the simplest action spaces (2-D $\Delta x, \Delta y$) in the dataset while also having the largest demonstration dataset, and thus the added advantage of having vector quantized actions may not have such a strong edge. Next, in unconditional tasks, we find that VQ-BeT outperforms all baselines in Franka Kitchen (state), Ant Multimodal, UR3 Multimodal, and both PushT (state and image) environments. In BlockPush environment, VQ-BeT is outperformed by DiffusionPolicy-T, while in Image Kitchen it is outperformed by DiffusionPolicy-C. However, VQ-BeT empirically shows stable performances on all tasks, while DiffusionPolicy-T struggles in Image PushT environments, and DiffusionPolicy-C underperforms in Kitchen and BlockPush environments.

C. *How well does VQ-BeT capture multimodality?*

One of the primary promises of behavior generation models is to capture the diversity present in the data, rather than simply copying a single mode of the existing data very well. Thus, for a quantitative measure we examine the behavior entropy of the models in the unconditional behavior generation task. Behavior entropy here tries to captures the diversity of a model's generated long horizon behaviors. We compare the final-subtask entropy as a balanced metric between performance and diversity. We see that VQ-BeT outperforms all baselines in all tasks except for Image Kitchen, where it's outperformed by DiffusionPolicy-T. However, behavior diversity is hard to capture properly in a single number, which is why we also present the diversity of generated behavior on the PushT task in Figure 1 (left). There, we can see how VQ-BeT captures both modes of the dataset in rollouts, while also generating overall smooth trajectories.

D. *Inference-time efficiency of VQ-BeT*

Unconditional	C-BeT	C-BESO	CFG-BESO	VQ-BeT
Single step	22.6ms	25.9ms	41.7ms	22.8ms
Multi step	✗	✗	✗	23.3ms
Conditional	**BeT**	**DiffusionPolicy-C**	**DiffusionPolicy-T**	**VQ-BeT**
Single step	13.2ms	100.5ms	98.6ms	15.1ms
Multi step	✗	100.7ms	98.6ms	15.2ms

TABLE III: Inference times for VQ-BeT and baselines in kitchen environment. For DiffusionPolicy we rolled-out with 10-iteration diffusion, following their real-world settings. The methods that only support single-step action prediction are marked with ✗.

Denoising diffusion based models such as DiffusionPolicy and BESO require multiple forward passes from the network to generate a single action or action chunk. In contrast, VQ-BeT can generate action or action chunks in a single forward pass. As a result, VQ-BeT enjoys much faster inference times, as shown in Table III. Receding horizon control using action chunking can speed up some of our baselines, but VQ-BeT can take advantage of the same, speeding up the method proportionally. Moreover, receding horizon control is not a silver bullet; it can be problematic in affordable, inaccurate hardware, as we show in Section IV-G in our real world experiments.

E. *Adapting VQ-BeT for autonomous driving*

While our previous experiments showed robotic manipulation or locomotion results, learning from multi-modal behavior datasets has wider applications. We evaluate VQ-BeT in one such case, in a self-driving trajectory planning task using the nuScenes [7] dataset. In this task, given a few frames of observations, the model must predict the next six frames of an car's location. While nuScenes usually require the trajectory be predicted from the raw images, we adapted the GPT-Driver [32] framework which uses pretrained models to extract vehicle and obstacle locations and velocities. However, this

Method	Access to information	Avg. L_2 (m) (↓)	Avg. collision (%) (↓)
FF [19]	Full	1.43	0.43
EO [26]		1.6	0.33
UniAD [21]		1.03	0.31
Agent-Driver [33]		<u>0.74</u>	<u>0.21</u>
GPT-Driver [32]	Partial	0.84	0.44
Diffusion-based traj. model		0.96	0.49
VQ-BeT		**0.73**	**0.29**

TABLE IV: *(Lower is better)* Trajectory planning performance on the nuScenes environment. We **bold** the best partial-information model and <u>underline</u> the best full-information model. Even with partial information about the environment, VQ-BeT can match or beat the SOTA models on the L_2 error metric.

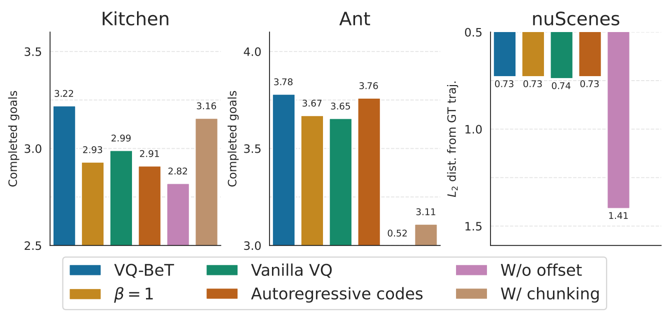


Fig. 5: Summary of our ablation experiments. The five different axes of ablation is described in Section IV-F.

processing also discards road lane and shoulder informations, which makes collision avoidance hard.

In Table IV, we show the performance of VQ-BeT in this task, measured by how closely it followed the ground truth trajectory in test scenes, as well as how likely the generated trajectory was to collide with the environment. Note that collision avoidance is especially difficult for agents with partial information since they do not have any lane information. We find that VQ-BeT outperforms all other methods in trajectory following, achieving the lowest average L_2 distance between the ground truth trajectories and generated trajectories. Moreover, VQ-BeT achieves a collision probability that is better or on-par with older self-driving methods, while not being designed for self-driving in particular.

F. *Design decisions that matter for VQ-BeT*

In this section, we examine how changes in each module of VQ-BeT affect its performance. We ablate the following components: using residual vs. vanilla VQ, using an offset head, using action chunking, predicting the VQ-codes autoregressively, and weighing primary and secondary codes equally by setting $\beta = 1$ in Eq. 4. We perform these ablation experiments in the conditional Kitchen, unconditional Ant, and the nuScenes self-driving task, and the result summary is presented in Figure 5.

We note that performance-wise, not using a residual VQ layer has a significant negative impact, which we believe is because of the lack of expressivity from a single VQ-layer. A similar drop in performance shows up when we weigh the two VQ layers equally by setting $\beta = 1$, in Eq. 4. Both experiments seems to provide evidence that important expressivity is conferred on VQ-BeT using residual VQs. Next, we note that predicting the VQ-codes autoregressively has a negative impact on the kitchen environment. This performance drop is anomalous, since in the real world, we found that the autoregressive (and thus causal) prediction of primary and secondary codes is important for good performance. In the environments where it is possible, we also tried action chunking [56]; however the performance for such models were lacking. Since VQ-BeT models are small and fast, action chunking isn't necessary even when running it on a real robot in real time. Finally, we found that the offset prediction is quite important for VQ-BeT, which points to how important full action fidelity is for sequential decision making tasks that we evaluate on.

G. Adapting VQ-BeT to real-world robots

While our previous experiments evaluated VQ-BeT in simulated environments, one of the primary potential applications of it is in learning robot policies from human demonstrations. In this section, we set up a real robot environment, collect some data, and evaluate policies learned using VQ-BeT.

a) Environment and dataset: For single-phase and two-phase tasks, we run our experiments in a kitchen-like environment with a toaster oven, a mini-fridge, and a small can in front of the robot as shown in Figure 3. For long-horizon scenarios consisting of more than three tasks, we also test on a real kitchen environment as shown in Figure 6. We use a similar robot and data collection setup as Dobb·E [44], and use the Hello Robot: Stretch [25] for policy rollouts. We create a set of single-phase and multi-phase tasks on this environment (See Table V, or Appendix B for details). While the single-phase tasks can only be completed in one way, some multi-phase tasks have multi-modal solutions in the benchmark and the datasets.

b) Baselines: In this environment, we use MLP-BC and BC with Depth as our simple baselines, and DiffusionPolicy-T as our multi-modal baseline. To handle visual inputs, all models are prepended with the HPR encoder from Shafiullah et al. [44] which is then fine-tuned during training.

Method	Open Toaster	Close Toaster	Close Fridge	Can to Toaster	Can to Fridge	Total
VQ-BeT	**8/10**	**10/10**	**10/10**	**10/10**	9/10	**47/50**
DiffPol-T†	**8/10**	9/10	8/10	**10/10**	**10/10**	45/50
BC w/ Depth	0/10	7/10	**10/10**	8/10	2/10	27/50
BC	0/10	8/10	7/10	9/10	5/10	29/50

Method	Can to Fridge → Close Fridge	Can to Toaster → Close Toaster	Close Fridge and Toaster	Total
VQ-BeT	**6/10**	**8/10**	5/10	**19/30**
DiffPol-T†	4/10	1/10	**6/10**	11/30
BC w/ Depth	2/10	0/10	2/10	4/30
BC	2/10	1/10	4/10	7/30

TABLE V: Real world robot experiments solving a number of standalone tasks (top) and two-task sequences (bottom). Here, † denotes that we modified DiffusionPolicy-T to improve its performance; see Section IV-G paragraph "Practical concerns".

c) Results: We present the experiment results from the real world environment in Table V and Table VI. Table V is split in two halves for single-phase and two-phase tasks. On the single-phase tasks, we see that, simple MLP-BC models are able to perform almost all tasks with some success, which shows that the subtasks are achievable, and the baselines are implemented well. On these single-phase tasks, VQ-BeT marginally outperforms DiffusionPolicy-T, while both algorithms achieve a $\geq 90\%$ success rate. However, the more interesting comparison is in the two-phase, longer horizon tasks. Here, VQ-BeT outperforms all baselines, including DiffusionPolicy, by a relative margin of 73%.

Besides comparisons with baselines, we also notice multi-modality in the behavior of VQ-BeT. Especially in the task "Close Fridge and Toaster", we note that our model closes the doors in both possible orders during rollouts rather than collapsing to a single mode of behavior.

Additionally, we present results from long-horizon real world experiments consisting of a sequence of three or more subtasks in Figure 6 and Table VI. We consider interactions with a wider variety of environments (communal kitchen and conference room) and objects (bread, box, bag, and drawer) compared to the single- or two-phase tasks in order to evaluate VQ-BeT in more general scenes. Overall, we see that VQ-BeT has at least thrice the success rate of DiffusionPolicy at the end of all four tasks. For Task 1 and 2, we observe that VQ-BeT gains a performance advantage toward the end of the episode, although VQ-BeT and DiffusionPolicy perform similarly at the beginning of the episodes. Also note that Task 2 is difficult in our ego-only camera setup, since the bag is out of the view while grabbing the bread. For Tasks 3 and 4, we observe that VQ-BeT outperforms DiffusionPolicy in all subtasks and notably, the performance difference is even more pronounced toward the end of the episode. These long-horizon task results continue to suggest that VQ-BeT may overfit less and learn more robust behavior policies in longer horizons tasks.

d) Practical concerns: In practice, we noticed that receding-horizon control as used by Chi et al. [9] fails completely in our environment (See Appendix Table XI for comparison to closed loop control). Our low-cost mobile manipulator robot lacks precise motion control unlike more expensive robot arms like Franka Panda. This controller noise causes models to go out of distribution during even a short period (three timesteps) of open-loop rollout. To resolve this, we rolled out every policy fully closed-loop, which resulted in a much larger inference time gap (25×) between VQ-BeT and Diffusion Policy as presented in Table VII.

V. Related Works

a) Deep generative models for modeling behavior: VQ-BeT builds on a long line of works that leveraged tools from generative modeling to learn diverse behaviors. The earliest examples are in inverse RL literature [24, 53, 13, 17], where such tools were used to learn a reward function given example behavior. Using generative priors for action generationi, such as GMM by Lynch et al. [30] or EBMs by Florence et al. [14], or simply fitting multi-modal action distributions [45, 37] became more common with large, human collected behavior datasets [31, 15]. Subsequently, a large body of work [43, 10, 36, 9, 40, 8] used generative modeling tools for generalized behavior learning from multi-modal datasets.

b) Action reparametrization: While Shafiullah et al. [43] is the closest analogue to VQ-BeT, the practice of reparametrizing actions for easier or better control goes back to "bang-bang" controllers [6, 3] replacing continuous actions with extreme discrete values. Discretizing each action dimension separately, however, may exponentially explode the action space, which is generally addressed by assuming each action dimension as independent [48] or causally dependent [35]. Without priors on the action space, each of these assumptions

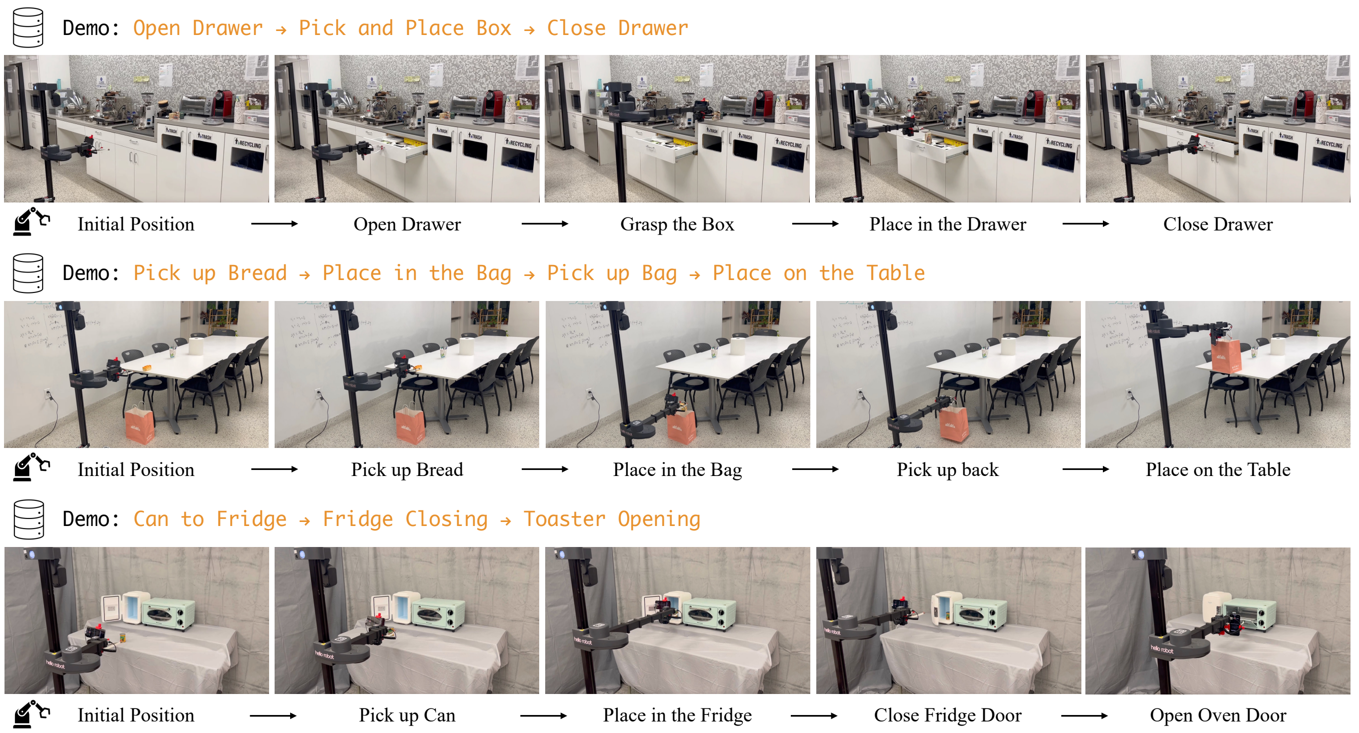


Fig. 6: Visualization of the trajectory VQ-BET generated in a long-horizon real world environment. Each demo consists of three to four consecutive tasks. Please refer to Table VI for the success rates for each task.

Task 1	Approach Handle	Grasp Handle	Open Drawer	Let Handle Go	Approach the Box	Grasp the Box	Move to Drawer	Place Box inside	Go in front of Drawer	Close Drawer
VQ-BeT	8/10	7/10	7/10	7/10	7/10	7/10	7/10	6/10	6/10	6/10
DiffPol-T†	10/10	9/10	9/10	9/10	8/10	3/10	3/10	3/10	3/10	2/10
Task 2	Approach Bread	Grasp the Bread	Move to the Bag	Place Bread inside	Approach the Handle	Grasp the Handle	Lift Bag up	Place on the table	Let Handle go	
VQ-BeT	10/10	10/10	10/10	4/10	3/10	3/10	3/10	3/10	3/10	
DiffPol-T†	9/10	9/10	9/10	9/10	2/10	2/10	2/10	1/10	1/10	
Task 3	Grasp Can	Pick up Can	Can into Fridge	Let Go of Can	Move Left of Fridge Door	Close Fridge Door	Go in Front of Toaster	Grasp Toaster Handle	Open Toaster	Return to Home Pos.
VQ-BeT	10/10	10/10	10/10	8/10	8/10	8/10	8/10	7/10	7/10	7/10
DiffPol-T†	5/10	5/10	5/10	4/10	2/10	2/10	2/10	2/10	2/10	2/10
Task 4	Grasp Can	Pick up Can	Can into Toaster	Drops Can on Tray	Goes Below Toaster Door	Close Toaster Door	Backs up	Move Left of Fridge Door	Close Fridge	Return to Home Pos.
VQ-BeT	10/10	10/10	8/10	8/10	8/10	6/10	6/10	6/10	6/10	6/10
DiffPol-T†	9/10	9/10	8/10	8/10	8/10	1/10	2/10	2/10	2/10	1/10

TABLE VI: Long-horizon real world robot experiments (Figure 6). Each task consists of three to four sequences; Task 1 (Open Drawer → Pick and Place Box → Close Drawer), Task 2 (Pick up Bread → Place in the Bag→ Pick up Bag → Place on the Table), Task 3 (Can to Fridge → Fridge Closing → Toaster Opening), and Task 4 (Can to Toaster → Toaster Closing → Fridge Closing). Here, † denotes that we modified DiffusionPolicy-T to improve its performance as explained in Section IV-G paragraph "Practical concerns".

	RTX A4000 GPU	4-Core Intel CPU
VQ-BeT	18.06	207.25
DiffusionPolicy-T	573.49	5243.82
BC w/ Depth	5.66	87.28
BC	4.73	83.28

TABLE VII: Average inference time for real robot (in milliseconds). The GPU column is calculated on our workstation while the CPU column is calculated on the Hello Robot's onboard computer.

may be limiting, which is why later work opted to learn the reparametrization [45, 11, 29] similar to VQ-BeT. On another hand, options [47, 46] abstract actions temporally but can be challenging to learn from data. Many applications instead hand-craft primitives as a parametrized action space [16] which may not scale well for different tasks.

VI. Conclusion

In this work, we introduce VQ-BeT, a model for learning behavior from open-ended, multi-modal data by tokenizing the action space using a residual VQ-VAE, and then using a transformer model to predict the action tokens. While we show that VQ-BeT performs well on a plethora of manipulation, locomotion, and self-driving tasks, an exciting application of such models would be in scaling them up to large behavior datasets containing orders of magnitude more data, environments, and behavior modes. Finding a shared latent space of actions between different embodiments may let us "translate" policies between different robots or even from human to robots. Finally, a learned, discrete action space may also make real-world RL application faster, which we would like to explore in the future.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Omer Bar-Tal, Hila Chefer, Omer Tov, Charles Herrmann, Roni Paiss, Shiran Zada, Ariel Ephrat, Junhwa Hur, Yuanzhen Li, Tomer Michaeli, et al. Lumiere: A space-time diffusion model for video generation. *arXiv preprint arXiv:2401.12945*, 2024.

[3] Richard Bellman, Irving Glicksberg, and Oliver Gross. On the "bang-bang" control problem. *Quarterly of Applied Mathematics*, 14(1):11–18, 1956.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Donald Wayne Bushaw. *Differential equations with a discontinuous forcing term*. PhD thesis, Princeton University, 1952.

[7] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[8] Lili Chen, Shikhar Bahl, and Deepak Pathak. Playfusion: Skill acquisition via diffusion from language-annotated play. In *Conference on Robot Learning*, pages 2012–2029. PMLR, 2023.

[9] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[10] Zichen Jeff Cui, Yibin Wang, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. From play to policy: Conditional behavior generation from uncurated robot data. *arXiv preprint arXiv:2210.10047*, 2022.

[11] Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous control with action quantization from demonstrations. *arXiv preprint arXiv:2110.10149*, 2021.

[12] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[13] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR, 2016.

[14] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[15] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[16] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[17] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[19] Peiyun Hu, Aaron Huang, John Dolan, David Held, and Deva Ramanan. Safe local motion planning with self-supervised freespace forecasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12732–12741, 2021.

[20] Shengchao Hu, Li Chen, Penghao Wu, Hongyang Li, Junchi Yan, and Dacheng Tao. St-p3: End-to-end vision-based autonomous driving via spatial-temporal feature learning. In *European Conference on Computer Vision*, pages 533–549. Springer, 2022.

[21] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862, 2023.

[22] Riashat Islam, Hongyu Zang, Anirudh Goyal, Alex Lamb, Kenji Kawaguchi, Xin Li, Romain Laroche, Yoshua Bengio, and Remi Tachet Des Combes. Discrete factorial representations as an abstraction for goal conditioned reinforcement learning. *arXiv preprint arXiv:2211.00247*, 2022.

[23] Bo Jiang, Shaoyu Chen, Qing Xu, Bencheng Liao, Jiajie Chen, Helong Zhou, Qian Zhang, Wenyu Liu, Chang Huang, and Xinggang Wang. Vad: Vectorized scene representation for efficient autonomous driving. *arXiv preprint arXiv:2303.12077*, 2023.

[24] Mrinal Kalakrishnan, Peter Pastor, Ludovic Righetti, and Stefan Schaal. Learning objective functions for manipulation. In *2013 IEEE International Conference on Robotics and Automation*, pages 1331–1336. IEEE, 2013.

[25] Charles C Kemp, Aaron Edsinger, Henry M Clever, and Blaine Matulevich. The design of stretch: A compact, lightweight mobile manipulator for indoor human envi-

ronments. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3150–3157. IEEE, 2022.

[26] Tarasha Khurana, Peiyun Hu, Achal Dave, Jason Ziglar, David Held, and Deva Ramanan. Differentiable raycasting for self-supervised occupancy forecasting. In *European Conference on Computer Vision*, pages 353–369. Springer, 2022.

[27] Jigang Kim, J hyeon Park, Daesol Cho, and H Jin Kim. Automating reinforcement learning with example-based resets. *IEEE Robotics and Automation Letters*, 7(3): 6606–6613, 2022.

[28] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[29] Jianlan Luo, Perry Dong, Jeffrey Wu, Aviral Kumar, Xinyang Geng, and Sergey Levine. Action-quantized offline reinforcement learning for robotic skill learning. In *Conference on Robot Learning*, pages 1348–1361. PMLR, 2023.

[30] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

[31] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR, 2018.

[32] Jiageng Mao, Yuxi Qian, Hang Zhao, and Yue Wang. Gpt-driver: Learning to drive with gpt. *arXiv preprint arXiv:2310.01415*, 2023.

[33] Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. A language agent for autonomous driving. *arXiv preprint arXiv:2311.10813*, 2023.

[34] Pietro Mazzaglia, Tim Verbelen, Bart Dhoedt, Alexandre Lacoste, and Sai Rajeswar. Choreographer: Learning and adapting skills in imagination. *arXiv preprint arXiv:2211.13350*, 2022.

[35] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

[36] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, et al. Imitating human behaviour with diffusion models. *arXiv preprint arXiv:2301.10677*, 2023.

[37] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR, 2021.

[38] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[39] Nived Rajaraman, Lin Yang, Jiantao Jiao, and Kannan Ramchandran. Toward the fundamental limits of imitation learning. *Advances in Neural Information Processing Systems*, 33:2914–2924, 2020.

[40] Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*, 2023.

[41] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[42] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[43] Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[44] Nur Muhammad Mahi Shafiullah, Anant Rai, Haritheja Etukuru, Yiqian Liu, Ishan Misra, Soumith Chintala, and Lerrel Pinto. On bringing robots home. *arXiv preprint arXiv:2311.16098*, 2023.

[45] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[46] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation: 5th International Symposium, SARA 2002 Kananaskis, Alberta, Canada August 2–4, 2002 Proceedings 5*, pages 212–223. Springer, 2002.

[47] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[48] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of the aaai conference on artificial intelligence*, volume 32, 2018.

[49] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[50] A Vasuki and PT Vanathi. A review of vector quantization techniques. *IEEE Potentials*, 25(4):39–47, 2006.

[51] Bob Wei, Mengye Ren, Wenyuan Zeng, Ming Liang, Bin Yang, and Raquel Urtasun. Perceive, attend, and drive: Learning spatial attention for safe self-driving. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4875–4881. IEEE, 2021.

[52] Chenfei Wu, Lun Huang, Qianxi Zhang, Binyang Li, Lei Ji, Fan Yang, Guillermo Sapiro, and Nan Duan. Godiva: Generating open-domain videos from natural descriptions. *arXiv preprint arXiv:2104.14806*, 2021.

[53] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[54] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:495–507, 2021.

[55] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8660–8669, 2019.

[56] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[57] Alon Ziv, Itai Gat, Gael Le Lan, Tal Remez, Felix Kreuk, Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. Masked audio generation using a single non-autoregressive transformer. *arXiv preprint arXiv:2401.04577*, 2024.

Appendix

A. *Simulated environments*

Across our experiments, we use a variety of environments and datasets to evaluate VQ-BeT. We give a short descriptions of them here, and depiction of them in Figure 3:

- **Franka Kitchen:** We use the Franka Kitchen robotic manipulation environment introduced in [15] with a Franka Panda arm with a 7 dimensional action space and 566 human collected demonstrations. This environment has seven possible tasks, and each trajectory completes a collection of four tasks in some order. While the original environment is state-based, we create an image-based variant of it by rendering the states with the MuJoCo renderer as an 112 by 112 image. In the conditional variant of the environment, the model is conditioned with future states or image goals (Image Kitchen).
- **PushT:** We adopt the PushT environment introduced in [9] where the goal is to push a T-shaped block on a table to a target position. The action space here is two-dimensional end-effector velocity control. Similar to the previous environment, we create an image based variant of the environment by rendering it, and a goal conditioned variant of the environment by conditioning the model with a final position. This dataset has 206 demonstrations collected by humans.
- **BlockPush:** The BlockPush environment was introduced by Florence et al. [14] where the goal of the robot is to push two red and green blocks into two (red and green) target squares in either order. The conditional variant is conditioned by the target positions of the two blocks. The training dataset here consists of 1,000 trajectories, with an equal split between all four possibilities of (block target, push order) combinations, collected by a pre-programmed primitive.
- **UR3 BlockPush:** In this task, an UR3 robot tries to move two blocks to two goal circles on the other side of the table [27]. Each demonstration is multimodality, since either block can move first. In the non-conditional setting, we evaluate whether each block reaches the goal, while in the conditional setting, we evaluate in which order the blocks get to the given target point.
- **Multimodal Ant:** We adopt a locomotion task that requires the MuJoCo Ant [4] robot to reach goals located at each corner of the map. The demonstration contains trajectories that reach the four goals in different orders. In the conditional setting, the performance is evaluated by reaching two goals given by the environment, while in the unconditional setting, the agent tries to reach all four goals.
- **nuScenes self-driving:** Finally, to evaluate VQ-BeT on environments beyond robotics, we use the nuScenes [7] self-driving environment as a test setup. We use the preprocessed, object-centric dataset from Mao et al. [32] with 684 demonstration scenes where the policy must predict the next six timesteps of the driving trajectory. In this environment, the trajectories are all goal-directed, where the goal of which direction to drive is given to the policy at rollout time. In Appendix Section E, we detail how we process the GPT-Driver Mao et al. [32] dataset for use in our method.

B. *Real-world environments*

We run our experiments on a kitchen-like environment, with a toaster oven, a mini-fridge, and a small can in front of them, as seen in Fig. 3. In this environment, we define the tasks as opening or closing the fridge or toaster, and moving the can from the table to the fridge or toaster and vice versa. During data collection and evaluation, the starting position for the gripper and the position of the cans are randomized within a predefined area, while the location of the fridge and the toaster stays fixed. We use a similar robot and data collection setup as Dobb·E [44], using the Stick to collect 45 demonstrations for each task, using 80% of them for training and 20% for validation, and using the Hello Robot: Stretch [25] for policy rollouts. While some of the single tasks can only be completed in one way, the we also test the model on sequences of two tasks, for example closing oven and fridge, which can be completed in multiple ways. This task multi-modality is also captured in the dataset: tasks that can be completed in multiple ways have multi-modal demonstration data.

		C-BeT	C-BESO	CFG-BESO	**VQ-BeT**
Kitchen	Full	3.09	3.75	3.47	**3.78**
	1/4	2.77	2.62	3.07	**3.46**
	1/10	2.59	2.67	2.73	**2.95**
Image Kitchen	Full	2.41	2.00	1.59	**2.60**
Ant Multimodal	Full	1.68	1.14	0.92	**1.72**
	1/4	0.85	0.58	0.52	**1.23**
	1/10	0.35	0.39	0.40	**1.06**
BlockPush Multimodal	Full	0.87	**0.93**	0.88	0.87
	1/4	0.48	0.52	0.47	**0.62**
	1/10	0.10	**0.29**	0.17	0.13
UR3 Multimodal	$-\ell_1$	-0.129	-0.090	-0.091	**-0.085**
	p1	**1.00**	0.98	0.97	**1.00**
	p2	0.67	**0.96**	0.94	0.94
PushT	Final Coverage	0.02	0.30	0.25	**0.39**
	Max Coverage	0.11	0.41	0.38	**0.49**
Image PushT	Final Coverage	0.01	0.02	0.01	**0.10**
	Max Coverage	0.02	0.02	0.02	**0.12**

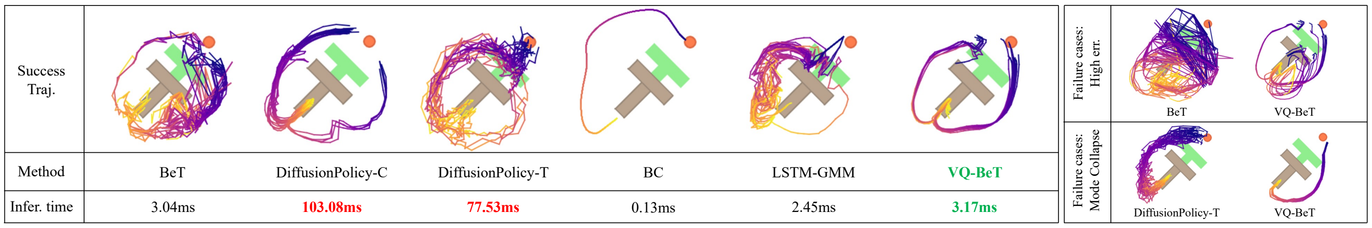
TABLE VIII: Quantitative results of VQ-BeT and related baselines on conditional tasks.

		BeT	DiffusionPolicy-C	DiffusionPolicy-T	**VQ-BeT**
PushT	Final Coverage	0.39	0.73	0.74	**0.78**
	Max Coverage	0.73	**0.86**	0.83	0.80
Image PushT	Final Coverage	0.01	0.66	0.45	**0.68**
	Max Coverage	0.01	**0.82**	0.71	0.73
Kitchen	p1	0.99	0.94	0.99	**1.00**
	p2	0.93	0.86	**0.98**	**0.98**
	p3	0.71	0.56	0.87	**0.91**
	p4	0.44	0.26	0.60	**0.77**
	p3-Entropy	**3.44**	3.18	3.38	3.42
	p4-Entropy	4.01	3.62	3.89	**4.07**
Image Kitchen	p1	0.97	0.99	0.97	**1.00**
	p2	0.73	**0.95**	0.90	0.93
	p3	0.51	0.73	**0.75**	0.67
	p4	0.27	**0.44**	0.39	0.38
	p3-Entropy	3.03	2.36	3.01	**3.20**
	p4-Entropy	2.77	2.93	**3.55**	3.32
Ant Multimodal	p1	0.91	**0.96**	0.87	0.94
	p2	0.79	0.81	0.78	**0.83**
	p3	0.67	0.73	0.69	**0.75**
	p4	0.36	0.62	0.56	**0.70**
	p3-Entropy	3.89	4.26	**4.27**	4.19
	p4-Entropy	3.55	4.18	4.11	**4.20**
BlockPush Multimodal	p1	0.96	0.36	**0.99**	0.96
	p2	0.71	0.11	**0.94**	0.83
	p2-Entropy	1.95	1.94	1.95	**1.99**
UR3 Multimodal	p1	0.84	**1.00**	**1.00**	**1.00**
	p2	0.75	0.83	0.82	**0.84**
	p2-Entropy	**0.99**	0.91	0.98	**0.99**

TABLE IX: Quantitative results of VQ-BeT and related baselines on non-conditional tasks.

		L2 (↓)				Collision (%) (↓)			
		1s	2s	3s	Avg.	1s	2s	3s	Avg.
ST-P3 metrics	ST-P3 [20]	1.33	2.11	2.90	2.11	0.23	0.62	1.27	0.71
	VAD [23]	0.17	0.34	**0.60**	**0.37**	0.07	0.10	0.24	0.14
	GPT-Driver [32]	0.20	0.40	0.70	0.44	0.04	0.12	0.36	0.17
	Agent-Driver [33]	**0.16**	0.34	0.61	**0.37**	**0.02**	**0.07**	**0.18**	**0.09**
	Diffusion-based Traj. Prediction	0.21	0.43	0.80	0.48	**0.01**	**0.07**	0.35	0.14
	VQ-BeT	0.17	**0.33**	**0.60**	**0.37**	**0.02**	0.11	0.34	0.16
UniAD metrics	NMP [55]	-	-	2.31	-	-	-	1.92	-
	SA-NMP [51]	-	-	2.05	-	-	-	1.59	-
	FF [19]	0.55	1.20	2.54	1.43	0.06	0.17	1.07	0.43
	EO [26]	0.67	1.36	2.78	1.60	0.04	0.09	0.88	0.33
	UniAD [21]	0.48	0.96	1.65	1.03	0.05	0.17	0.71	0.31
	GPT-Driver [32]	0.27	0.74	1.52	0.84	0.07	0.15	1.10	0.44
	Agent-Driver [33]	**0.22**	0.65	**1.34**	0.74	0.02	**0.13**	**0.48**	**0.21**
	Diffusion-based Traj. Prediction	0.27	0.78	1.83	0.96	**0.00**	0.27	1.21	0.49
	VQ-BeT	**0.22**	**0.62**	**1.34**	**0.73**	0.02	0.16	0.70	0.29

TABLE X: *(Lower is better)* Trajectory planning performance on the nuScenes [7] self-driving environment. We **bold** the best performing model. Note that while Agent-Driver outperforms us in some Collision avoidance benchmarks, it is because they use a lot more information than what is available to our agent, namely the road lanes and the shoulders information, without which avoiding collision is difficult for our model or GPT-Driver [32]. Even with such partial information about the environment, VQ-BeT can match or beat the SOTA models in predicting L2 distance from ground truth trajectory.



Method	BeT	DiffusionPolicy-C	DiffusionPolicy-T	BC	LSTM-GMM	VQ-BeT
Infer. time	3.04ms	103.08ms	77.53ms	0.13ms	2.45ms	3.17ms

Fig. 7: Multi-modal behavior visualization on pushing a T-block to target. On the left, we can see trajectories generated by different algorithms and their inference time per single step, where VQ-BeT generate smooth trajectories to complete the task with both modes with short inference time. On the right, we can see failure cases of VQ-BeT and related baselines due to high error and mode collapse.

Control method	Close Toaster	Close Fridge	Can to Toaster	Can to Fridge	Can to Fridge → Close Fridge	Close Fridge and Toaster	Total
Closed loop ($n = 1$)	9/10	8/10	10/10	10/10	4/10	6/10	47/60
Receding horizon ($n = 3$)	0/5	0/5	0/5	0/5	0/5	0/5	0/30

TABLE XI: Quantitative results of running diffusion policy [9] with closed-loop vs. receding horizon control in real-world robot experiments, where n is the number of actions executed at each timestep. We select four single-phase tasks and two two-phase tasks in which diffusion policy does well with closed-loop control, and compare with the same policy with receding horizon control by executing multiple predicted actions at each timestep. We see the diffusion policy with an action sequence executed per timestep goes out of distribution quite easily and fails to complete any tasks on this set of experiments.

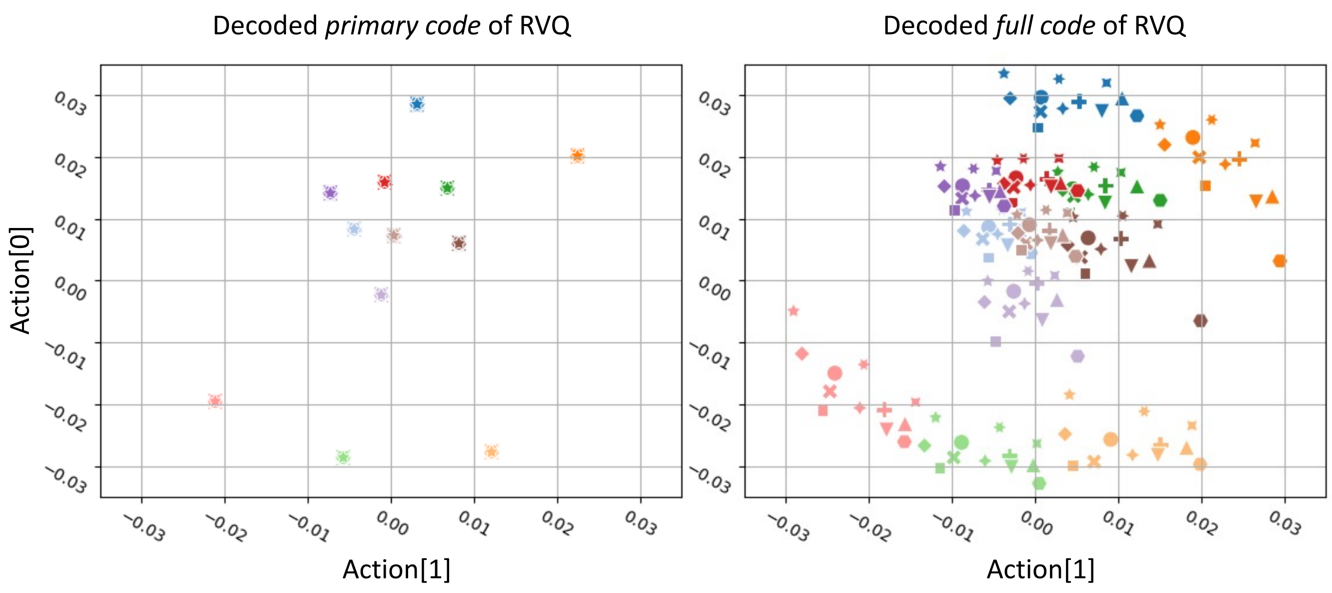


Fig. 8: Action centroids of primary codes and full combination of the codes. On the left, we represent centroids of the raw action data obtained by decoding (total of 12) primary codes learned from Blockpush Multimodal dataset. On the right, we show the decoded action of the centroids corresponding to all 144 possible combinations of full the codes. We can see that the primary codes, represented by different colors in each figure, are responsible for clustering in the coarse range, while full-code representation provides further finer-grained clusters with secondary codes.

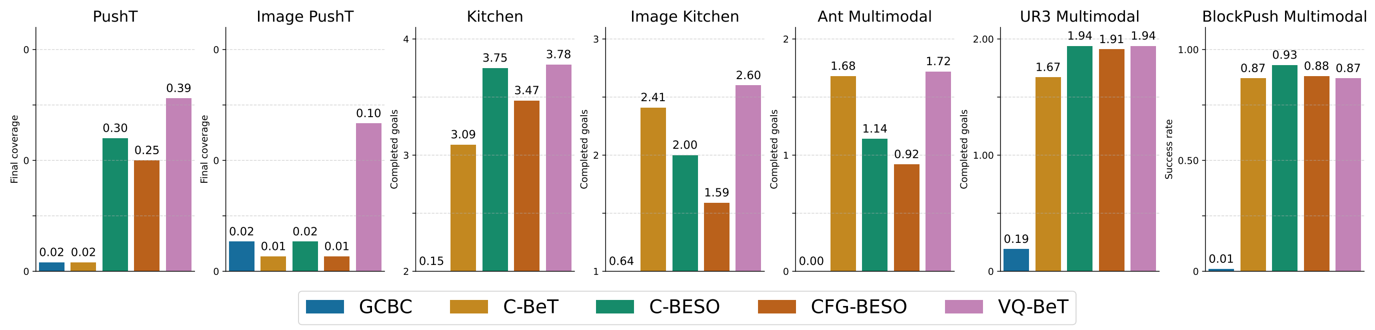


Fig. 9: Evaluation of conditional tasks in simulation environments of VQ-BeT and related baselines. VQ-BeT achieves the best performance in most simulation environments and comparable performance with the best baseline on BlockPush.

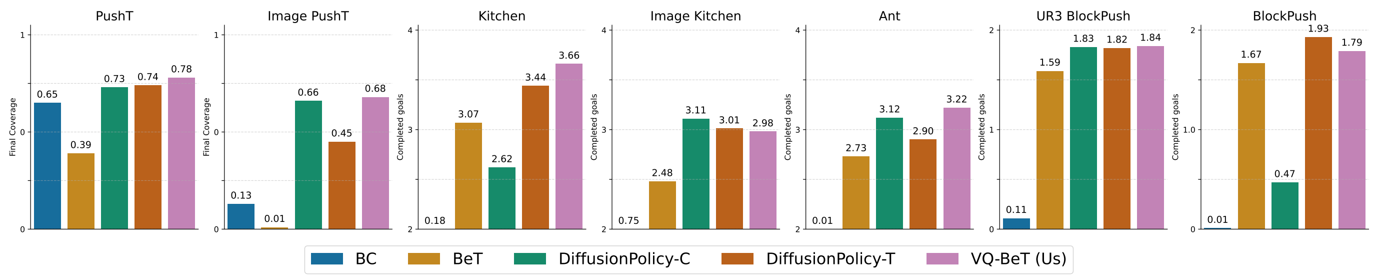


Fig. 10: Evaluation of unconditional tasks in simulation environments of VQ-BeT and related baselines. VQ-BeT achieves the best performance in most simulation environments and comparable performance with the best baseline on BlockPush and Image Kitchen.

C. *VQ-BeT with larger Residual VQ Codebook*

		Original codebook (Vanilla VQ-BeT)	Extended Codebook (Vanilla VQ-BeT)	Extended Codebook (VQ-BeT + Deadcode Masking)
Ant Multimodal (Unconditional)	Codebook Size	10	32	32
	# of Code Combinations	100	1024	1024
	$(\cdot/4)$	**3.22**	3.01	3.11
	p3-Entropy	4.19	4.23	**4.33**
	p4-Entropy	4.20	4.24	**4.32**
Ant Multimodal (Conditional)	Codebook Size	10	48	48
	# of Code Combinations	100	2304	2304
	$(\cdot/2)$	1.72	1.75	**1.81**
Kitchen (Unconditional)	Codebook Size	16	64	64
	# of Code Combinations	256	4096	4096
	$(\cdot/4)$	3.66	**3.75**	3.7
	p3-Entropy	**3.42**	3.01	3.10
	p4-Entropy	**4.07**	3.57	3.74
PushT (UnConditional)	Codebook Size	16	64	64
	# of Code Combinations	256	4096	4096
	Final Coverage	0.78	0.77	**0.79**
	Max Coverage	0.80	0.80	**0.82**
Kitchen (Conditional)	Codebook Size	16	256	256
	# of Code Combinations	256	65536	65536
	$(\cdot/4)$	**3.78**	3.61	3.56

TABLE XII: Evaluation of conditional and unconditional tasks in simulation environments of VQ-BeT with extended size of Residual VQ codebook.

In this section, we present additional results to evaluate the performance of VQ-BeT with larger residual VQ codebooks. While the results of VQ-BeT across the manuscript were obtained using 8 to 16-sized codebooks, resulting in 64 to 256 code combinations (Table XIII), here, VQ-BET was trained on codebooks with 10 to 250 times more combinations, as detailed in Table XII. First, we evaluate VQ-BeT with extended codebook size without any modifications ('Vanilla VQ-BeT'). Next, we test VQ-BeT with an additional technique where the code combinations that do not appear in the dataset are masked with a probability of zero at sampling time to eliminate the possibility of these combinations.

As shown in Table XII, we find that increasing the number of combinations ($\times 10 \sim \times 250$) had little impact on performance in most environments. In environments Ant Multimodal (Conditional) and PushT (Unconditional), overall performance slightly increased as the size of the VQ codebook increased. In environments Ant Multimodal (Unconditional) and Kitchen (Unconditional), we see that there is a performance and entropy trade-off as the size of the codebook increases. The only environment where the performance of VQ-BeT decreased with the extended size of the codebook was Kitchen (Conditional). Also, we see that there is no consistent evidence on whether using masking the deadcode (code combinations that do not appear in the dataset) is better: in Ant and PushT environments, masking led to similar or better performance, while in the Kitchen environment, we find similar or slightly worse performance with masking.

Overall, we conclude that VQ-BeT has robust performance to the size of the codebook if it is enough to capture the major modes in the dataset. We conjecture that this robustness is due to VQ-BeT assigning appropriate roles between primary and secondary codes as the codebook size increases. For example, in the Kitchen (Conditional) environment where we have increased the number of possible combinations by 256, the code prediction accuracy rate has decreased by only $\times 0.08$ of its original accuracy rate, while the primary code prediction retained $\times 0.8$ of its original accuracy rate. Interestingly, Despite this large difference, the performance difference between the two is small, around 4.5% (3.78 vs 3.61). These results suggest that VQ-BeT could rely on the resolution of the primary code in large VQ codebook size, while using less weight on the secondary code to handle the excessive number of code combinations, leading to robust performance to the size of the codebook.

D. Model Design Choises

Hyperparameter	Kitchen	Ant	BlockPush	UR3	PushT	NuScenes	Real-world
Obs window size	10	100	3	10	5	1	6
Goal window size (Conditional Task)	10	10	3	10	5	1	-
Predicted act sequence length	1	1	1	10	5	6	1
Autoregressive code pred.	False	False	False	False	False	True	True
β (Eq. 4)	0.1	0.6	0.1	0.1	0.1	0.1	0.5
Training Epoch	1000	300	1500	300	2000	1000	600
Learning rate	5.5e-5	5.5e-5	1e-4	5.5e-5	5.5e-5	5.5e-5	3e-4
MinGPT layer num	6	6	4	6	6	6	6
MinGPT head num	6	6	4	6	6	6	6
MinGPT embed dims	120	120	72	120	120	120	120
VQ-VAE latent dims	512	512	256	512	512	512	512
VQ-VAE codebook size	16	10	8	16	16	10	8/10/16
Encoder (Image env)	ResNet18	-	-	-	ResNet18	-	HPR

TABLE XIII: Hyperparameters for VQ-BeT

E. VQ-BeT for Driving Dataset

While all the other environments reported in this paper have a fixed observation dimension at one timestep, NuScenes driving dataset, as processed in the GPT-Driver paper [32], could contain the different number of detected objects in each scene. Thus, we make modification to the input types of VQ-BeT to train VQ-BeT with NuScenes driving dataset in response to this change in dimensionality of the obeservation data. The tokens we pass to VQ-BeT are as shown below:

- **Mission Token** indicates the mission that the agent should follow: go forward / turn left / turn right
- **Ego-state Token** contains velocity, angular velocity, acceleration, heading speed, and steering angle.
- **Trajectory History Token** contains ego historical trajectories of last 2 seconds, and ego historical velocities of last 2 seconds.
- **Object Tokens** contains perception and prediction outputs corresponding to current position, predicted future position, and one-hot encoded class indicator of each object. There are total of 15 classes. (‘pushable-pullable’, ‘car’, ‘pedestrian’, ‘bicycle’, ‘truck’, ‘trafficcone’, ‘motorcycle’, ‘barrier’, ‘bus’, ‘bicycle-rack’, ‘trailer’, ‘construction’, ‘debris’, ‘animal’, ‘emergency’)

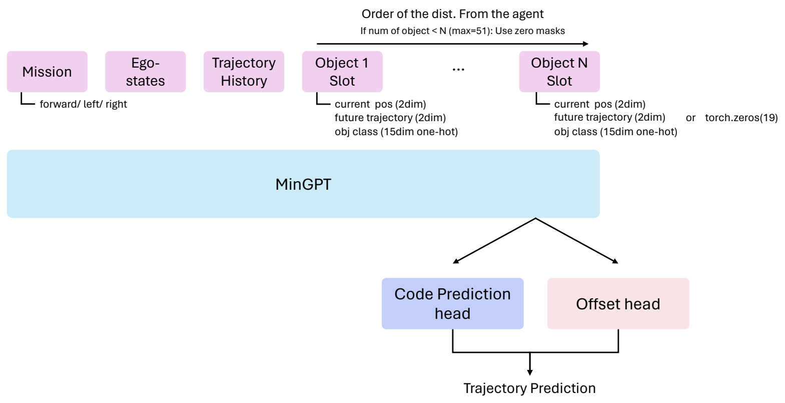


Fig. 11: Overview of VQ-BeT for autonomous driving.