BEAST: Efficient Tokenization of B-Splines Encoded Action Sequences for Imitation Learning

Hongyi Zhou[†] Weiran Liao[†] Xi Huang[†] Yucheng Tang[†] Fabian Otto[§] Xiaogang Jia [†]
Xinkai Jiang [†] Simon Hilber [†] Ge Li [†] Qian Wang [†] Ömer Erdinç Yağmurlu [†]
Nils Blank [†] Moritz Reuss [†] Rudolf Lioutikov[†]
[†] Karlsruhe Institute of Technology [§] Microsoft Research

***Abstract*—We present the B-spline Encoded Action Sequence Tokenizer (BEAST), a novel action tokenizer that encodes action sequences into compact discrete or continuous tokens using B-spline. In contrast to existing action tokenizers based on vector quantization or byte pair encoding, BEAST requires no separate tokenizer training and consistently produces tokens of uniform length, enabling fast action sequence generation via parallel decoding. Leveraging our B-spline formulation, BEAST inherently ensures generating smooth trajectories without discontinuities between adjacent segments. We extensively evaluate BEAST by integrating it with three distinct model architectures: a Variational Autoencoder (VAE) with continuous tokens, a decoder-only Transformer with discrete tokens, and Florence-2, a Vision-Language Model with an encoder-decoder architecture, demonstrating BEAST's compatibility and scalability with large pretrained models. We evaluate BEAST across three established benchmarks consisting of 166 simulated tasks and on three distinct robot settings with a total of 8 real-world tasks. Experimental results demonstrate that BEAST (i) significantly reduces both training and inference computational costs, and (ii) consistently generates smooth, high-frequency control signals suitable for continuous control tasks while (iii) reliably achieves competitive task success rates compared to state-of-the-art methods. Videos and code are available at the project page.**

I. Introduction

Imitation learning has emerged as a powerful paradigm for training robots to perform complex tasks by learning from human demonstrations [21]. Early works [7, 18] in this field primarily focused on predicting single-step actions based on the current observation. However, recent research [34] highlights the importance of learning action sequences to capture the temporal coherence inherent in human demonstrations. Moreover, by modeling action sequences, we can reduce compounding errors [6] and create task demonstrations that more closely align with human methods [14]. Given the success of autoregressive next-token prediction models in natural language processing and other domains [32, 20, 25], it is compelling to explore similar techniques for modeling action sequences, leveraging their ability to predict and generate coherent sequences effectively.

In natural language processing, tokens typically represent words, which are inherently discrete elements. This discrete nature allows for effective next-token prediction, which extends well to the generation and prediction of symbolic actions or in discrete action space. However, a significant challenge arises when attempting to apply these approaches to sub-symbolic, continuous actions, which are not inherently discrete. Discretization addresses this issue by compressing the continuous action sequence while trying to retaining essential information. This process helps in balancing the expressivity of the action representation against computational efficiency.

Despite growing interest in this area, effective strategies to create action sequences of discrete tokens remain underexplored. Existing approaches often focus on single-step tokenization [12, 4, 5], vector quantization [15, 30, 31], or compression-based schemes [23]. However, they require training separate encoder-decoder networks for the tokenizer [15, 29] or produce variable-length token sequences for inputs of the same duration [23], which complicates applying fast token generation techniques such as parallel decoding [13]. Furthermore, existing action tokenizers do not consider the smooth transitions between subsequent action chunks, which could result in undesired jumps at transition.

To address these challenges, we propose the B-spline Encoded Action Sequence Tokenizer (**BEAST**), a novel tokenizer that represents continuous action sequences using B-splines [8]. BEAST offers versatility, allowing for effective integration with both discrete and continuous tokens. Different from tokenizers based on the vector quantization [15, 30, 31], it does not require additional tokenizer training. BEAST compresses action trajectories into fixed-length token sequences enabling efficient parallel decoding for faster token generation, requiring $4-8\times$ fewer tokens than binning-based tokenization. By using B-spline encoded control points as discrete tokens, BEAST ensures the generation of smooth action chunks, as well as the continuous connection between consecutive chunks.

Our contributions are: 1) We introduce BEAST, a novel B-spline-based tokenizer designed for modeling continuous action sequences. 2) We demonstrate the versatility of BEAST by integrating it into diverse model architectures that accommodate both continuous and discrete objectives. 3) We conduct extensive evaluations of simulated and real-world robotic tasks, showcasing its effectiveness. 4) We perform thorough ablation studies to assess the impact of various design choices.

II. Preliminaries

Problem Formulation. Our goal is to train a policy $\pi(\boldsymbol{a}_{1:T}|\ \boldsymbol{s})$ that capable of mapping a given state $\boldsymbol{s}$ to a corresponding sequence of actions $\boldsymbol{a}_{1:T}$ which has T time steps and D Degrees of Freedom (DoF). To make this sequence

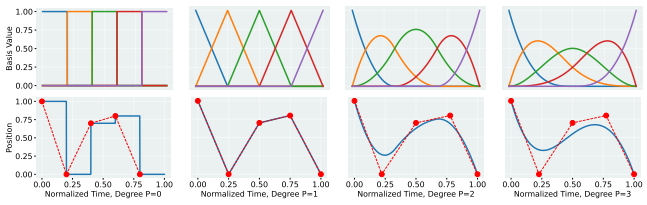

Fig. 1: **From left to right**: Clamped B-Spline Basis $P = 0, 1, 2, 3$ (top) and their generated trajectories (Bottom). Given the same **control points**, a higher degree will lead to smoother trajectories. All generated trajectories start exactly from the first control point and end at the last control point. Notably, action chunk is conceptually equivalent to B-Splines of 0-th degree, i.e., split-wise constants, as shown in the leftmost subplots. This relation is explained in details later in Section III-A.

prediction problem compatible with discrete generative models, we first transform the continuous action sequence into a sequence of discrete tokens. The goal of action sequence tokenization is to obtain a discrete token sequence $\bar{\boldsymbol{v}}_{1:J}$, where each token belongs to a vocabulary $\bar{\mathcal{V}}$ with size $|\bar{\mathcal{V}}|$, by defining a transformation $\texttt{tokenizer} : \boldsymbol{a}_{1:T} \rightarrow \bar{\boldsymbol{v}}_{1:J}$,

B-Splines (Basis Splines) [24] are widely used in the field of computer graphics and computer-aided design. A B-Spline curve y is formulated through a linear basis function representation

$$\textbf{1-DoF B-Spline:} \qquad y(u) = \sum_{n=0}^{N-1} \Phi_n^P(u)\; c_n = \boldsymbol{\Phi}^P(u)\; \boldsymbol{c},$$
$$0 \leq P < N, \quad u \in [k_0, k_M],$$

where $\boldsymbol{c}$ are N *control points* and u is a continuous parameter, often interpreted as *normalized time*. The *basis functions* $\boldsymbol{\Phi^P}(u) = [\Phi_0^P(u), .., \Phi_{N-1}^P(u)]$ are N polynomial basis functions of P-th degree. These basis functions are defined over M intervals determined by $M+1$ knots in a *vector* $[k_0, ..., k_M]$, and it satisfies $M = N + P$ [24]. Typically, the knot vector is normalized such that $k_0 = 0$ and $k_M = 1$. The basis functions $\Phi_n^P(u)$ are recursively computed using the *Cox–de Boor recursion* [3]. We denote all *recursive degrees*[1] as $q = 0 : P$. For $q = 0$, the basis functions are defined as piecewise constant and recursively using the $(q-1)$-th degree basis for $q > 0$ with

$$\Phi_n^0(u) = \begin{cases} 1 & \text{if } k_n \leq u < k_{n+1}, \\ 0 & \text{otherwise.} \end{cases} \text{ and} \tag{1}$$

$$\Phi_n^q(u) = k_n^{q-1}\Phi_n^{q-1}(u) + (1 - k_{n+1}^{q-1})\Phi_{n+1}^{q-1}(u), \tag{2}$$

where $k_n^{q-1} = (u - k_n)/(k_{n+q} - k_n)$.

Clamped B-Spline. In this work, we employ the *clamped uniform B-Spline*, where the first and last $P+1$ knots are repeated to ensure that the resulting curve starts at the first control point and ends at the last control point. In Figure 1, we demonstrate the resulting basis functions of degrees from $P = 0$ to $P = 4$, together with their generated trajectories, given the same five control points. Clamped uniform B-splines are particularly suited for trajectory generation due to their smoothness, compact representation, and local support, where each control point only affects the curve locally.

[1]The B-Spline degree P differs from the recursive degree q. Trajectories are represented by basis functions of degree P, while lower recursive degree q serve as intermediate representations in the recursive process.

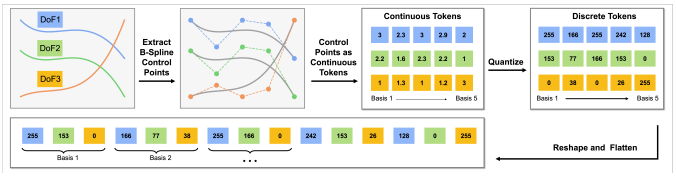

Fig. 2: **Overview of the BEAST Encoding Pipeline:** Given a normalized action sequence, the BEAST pipeline first uses linear regression to extract continuous-valued control points, forming control point matrices that serve as intermediate continuous representations. These matrices are then quantized uniformly into discrete values within the range $[0, 255]$ and subsequently flattened to produce discrete action tokens for auto-regressive next-token prediction or parallel prediction.

Parallel Decoding. Unlike autoregressive generation, which predicts tokens sequentially and thus requires K forward passes for a sequence of length K, *parallel decoding* [13] enables the prediction of the entire output sequence in a single forward pass. This is achieved by feeding the model with K empty token embeddings and replacing the causal attention mask with a bidirectional mask, allowing the decoder to infer the entire sequence simultaneously. OpenVLA-OFT [13] leverages this approach for action sequence generation. In this work, we adopt the parallel decoding strategy to predict all BEAST tokens in a single pass, improving the inference efficiency without sacrificing accuracy.

III. B-Spline Encoded Action Sequence Tokenizer

In this section, we first describe how BEAST utilizes B-Spline to construct an efficient action sequence tokenizer that converts action sequences into either continuous or discrete action tokens. We then explain how smooth transitions between consecutive action sequences are achieved by enforcing the initial conditions of clamped B-splines. Finally, we discuss strategies for efficient integrating BEAST with various model architectures that predict discrete or continuous tokens.

A. Action Sequence Tokenization with B-Spline Tokenizer

Following prior works in action tokenization [12, 23], we first normalize the input actions such that the 1st and 99th quantile value of each action dimension in the dataset maps to the range of $[-1, 1]$. Using quantiles makes the normalization robust against outlier data points.

Figure 2 presents an overview of the tokenization process. We begin by considering the tokenization of a 1-DoF trajectory. Given a normalized action sequence $a_{1:T} = [a_1, a_2, ..., a_T]$ of length T, our goal is to determine a set of N control points $\boldsymbol{c}$, with $N \leq T$, that approximate the given action sequence at spline evaluations $y(u)_{1:T}$. The linear transformation $u = {}^t/_T$ maps from action timestep to the parametric coordinate of the

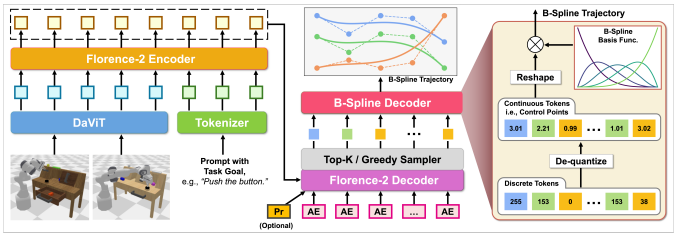

Fig. 3: **BEAST-F** is a new VLA model that combines BEAST encoding with Florence-2 [33], a lightweight VLM. BEAST produces uniform-length tokens, which allows BEAST-F to perform *parallel decoding* via learnable action embeddings (**AE**), instead of autoregressive next-token prediction. These discrete tokens are fed into the *B-Spline Decoder*, which first maps them to real-valued control points and then transforms those control points into continuous action sequences. The **Pr** token denotes an optional proprioceptive state.

B-Spline. The spline evaluations $y(u)_{1:T}$ are approximated by minimizing the least-squares error

$$\boldsymbol{c} = \arg\min_{\boldsymbol{c}} ||y_{1:T} - a_{1:T}||_2^2 = \arg\min_{\boldsymbol{c}} ||\boldsymbol{\Phi}^P(u)\boldsymbol{c} - a_{1:T}||_2^2, \tag{3}$$

where $\boldsymbol{\Phi}^P(u) = [\Phi_1^P(u), \Phi_2^P(u), ..., \Phi_N^P(u)]^\top$ represents pre-computed B-spline basis functions defined over interval $u \in [0, 1]$. Ridge regression estimates the control points in closed form, $\boldsymbol{c} = [c_0, c_1, ..., c_{N-1}] = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^\top a_{1:T}$, with λ acting as a regularization parameter. This efficient computation typically introduces only a small overhead, typically 3 to 5 milliseconds per batch. For a high-dimensional action sequence, i.e. $D > 1$, each DoF is encoded independently into $\boldsymbol{c}_d$, resulting in a matrix $\boldsymbol{C}$ of shape $D \times N$, that stacks each DoF's control points, $\boldsymbol{C} = [\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_D]^\top$.

To form the final token sequence, this matrix is flattened by interleaving different action dimensions corresponding to the same basis functions, as illustrated in Figure 2. This flattening strategy preserves the temporal order inherent in the trajectory segments associated with each basis function.

Remark 1: Connection to Action Chunking. Action chunking, defined as a discrete sequence of actions $a_0, a_1, ..., a_T$, is mathematically equivalent to a piecewise constant function generated by 0-th degree B-splines. As demonstrated in Figure 1 left most, each action step a_t can be identified as a control point c_n of 0-th degree B-Spline basis with $t = n, T = N$.

B. Enforcing Smooth Transition with Clamped B-Spline

Executing long-horizon tasks typically requires producing multiple small action sequences that connect seamlessly (replanning). While predicting action sequences effectively improves consistency within individual action chunks, a significant challenge lies in managing discontinuities at transitions between consecutive chunks, which often result in jerky motion during online execution. Common approaches to address this issue apply temporal ensembles of actions [34, 16], calculating moving averages over multiple predictions. However, these temporal ensembles require high-frequency replanning (typically every timestep) to generate sufficient chunks for effective ensemble averaging, which significantly constrains execution speed in online applications.

In contrast, BEAST employs clamped B-Spline to ensure smooth transitions between consecutive action chunks. As introduced in Section II, clamped B-Spline is a specialized variant of B-Spline that guarantees to start from the first control point and end at the last control point, which is utilized to generate seamlessly connected action sequences, as illustrated in Figure 1. To ensure smooth transitions, we directly set the first control point $\boldsymbol{c}_0$ to the last action of the previous sequence. We then compute the residual trajectory $\hat{\boldsymbol{a}}$ by subtracting the contribution of the first basis function: $\hat{\boldsymbol{a}} = \boldsymbol{a} - c_0\Phi_0^P$. The remaining control points $\hat{\boldsymbol{c}} = [c_1, c_2, ...c_{N-1}]$ are determined by solving the linear regression problem similar to equation 3: $\arg\min_{\boldsymbol{c}} ||\hat{\Phi}^P(u)\hat{\boldsymbol{c}} - \hat{\boldsymbol{a}}||_2$. Through this approach, BEAST consistently generates action sequences with mathematically guaranteed smooth transitions between chunks. This will be further discussed in our toy task experiment in Section IV-A.

C. Combining BEAST tokens with different architectures

Discrete Tokens. We first evaluate BEAST in a simplified setting with a decode-only transformer (see Figure 9) with CLIP [26] for language encoding and Film-conditioned ResNet-18 [9, 22] as image encoder. Film-ResNets are used in many prior works given their high efficiency and strong performance [1, 28, 4] The proprioceptive state of robot is projected to the embedding dimension with a two-layer MLP. We employ parallel decoding with bi-directional attention to accelerate the inference. To further demonstrate the scalability of BEAST with large-pretrained models, we combine BEAST with Florence-2, a small, pretrained VLM with Encoder-Decoder architecture (0.77B parameters). Following the previous works on autoregressive VLAs [12, 23], we overwrite the least used 256 tokens in the VLM vocabulary as our action tokens. We also employ a parallel decoding technique for the Florence variant, which significantly improves the throughput and reduces the latency for action generation. We provide an in-depth overview in Figure 3.

Continuous Tokens. We also explore the performance of combining BEAST with ACT [34]. ACT uses a conditional VAE (CVAE) with a Transformer Encoder-Decoder to predict a sequence of actions. We predict N BEAST continuous tokens, where each token has the dimension of D, instead of action sequences, this design choice keeps the temporal order inherent in the trajectory segments. Using the BEAST tokenization, we reduce the length of predicted token sequence by 6.67 times (from 100 to 15) without sacrificing the task performance. In addition, our method enables smooth trajectories without requiring temporal aggregation.

IV. Experiments

We conducted extensive evaluations in both simulated and real-world settings, targeting answering five key research questions (RQs): 1) What advantages does BEAST offer over commonly used binning-based tokenizers? 2) How does

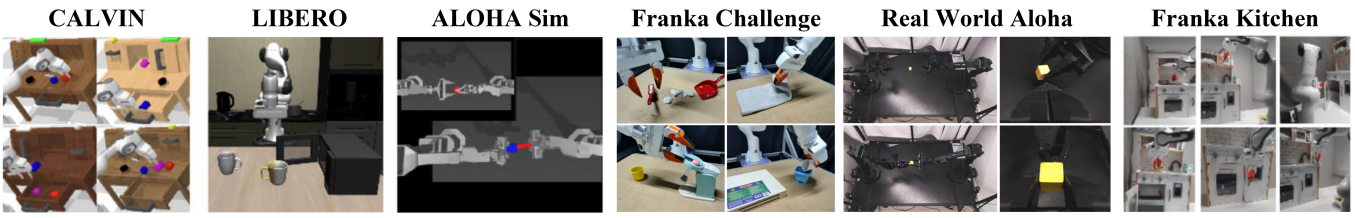

Fig. 4: Simulation [34, 19, 17] and real world (Franka Challenge, Aloha, Franka Kitchen) tasks.

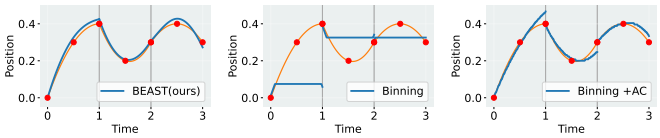

Fig. 5: Comparison among BEAST, single-step binning tokenization and binning tokenization with action chunking (AC). The comparison is conducted through the same auto-regressive model with different tokenizers to fit the **ground truth** cube splines given the same **context points**. BEAST is smooth within each sequence and continuous at the transitions.

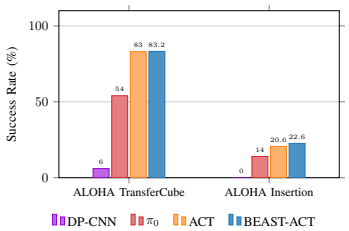

Fig. 6: **ALOHA Benchmark results.** The success rate is reported over 500 episodes of evaluation.

BEAST contribute to the performance on imitation learning benchmarks? 3) How does BEAST affect the training and inference efficiency? 4) Does BEAST generalize to real-world scenarios? 5) How do the design choices affect the performance of BEAST? BEAST is integrated into two different architectures: We combine BEAST and Florence-2 [33] and term this VLA variant as BEAST-F. In contrast to many baselines, we test BEAST-F **without second-stage pretraining on large-scale robot datasets**. We additionally apply BEAST to a small decoder-only transformer model (BEAST-D) and on top of a vanilla ACT [34] (BEAST-ACT).

A. Comparing Against Binning-Based Tokenization

To answer **RQ1**, we begin with a 1D toy task to investigate the advantages of BEAST over binning-based tokenization. We follow the autoregressive prediction pipeline used in previous works [12, 23]. Note that BEAST can be used for both autoregressive prediction and parallel decoding. A small decoder-only transformer is trained to predict cubic splines from 3 control points. We compare against: 1) Single-step binning (denoted as **Binning**) [12], which discretizes each action into one of 256 bins, and 2) Chunk-level binning (denoted as **Binning+AC**), which discretizes entire action sequences of fixed length. We generate 2000 trajectories, 1s each at 100Hz resolution. Each model is trained for 8k steps and evaluated on 200 test sequences. BEAST achieves the lowest MSE (0.0004 ± 0.0005), outperforming chunked binning (0.0009 ± 0.0013) and single-step binning (0.0215 ± 0.0216), with the latter performing two orders of magnitude worse. To simulate real-world action chunking [6], we repeat the rollout prediction three times. As visualized in Figure 5, single-step binning fails to capture temporal structure and produces erratic outputs. Chunked binning captures some temporal coherence but results in jerky transitions due to discretization and a lack of continuity across chunks. In contrast, BEAST generates smooth trajectories with minimal error and requires only 5 tokens per 100-step sequence, resulting in an approximately 20x reduction in inference steps.

B. Strong Performance on Established Simulation Benchmarks

To answer **RQ2**, we evaluate BEAST on established simulation benchmarks and compare with other SoTA imitation learning methods and VLAs.

Simulation Benchmarks. CALVIN [19] features 34 tabletop manipulation tasks with a Franka Panda robot using delta end-effector control across four scene configurations (splits A-D). The dataset contains $24,000$ language-annotated demonstrations. We evaluate two settings: CALVIN ABC (zero-shot generalization) and CALVIN ABCD (scaling with more data). Performance is measured by success rates on sequential tasks and mean sequence length completion. All evaluations require policies to follow free-form language instructions and complete 5 tasks in sequence across $1,000$ different instruction chains. **LIBERO** [17] tests a delta-EEF controlled Panda Robot across various scenes with **130** diverse tasks. We report results on four specialized benchmark settings with 10 tasks each (Long, Spatial, Object, and Goal). Success is measured as the percentage of successful task completions across 50 trials per task. **ALOHA** [34] tests an absolute joint position controlled ALOHA Robot in two challenging bi-manual manipulation tasks that require high precision.

Baselines. We compare our Vision-Language-Action Model (VLA) against SOTA VLA policies and specialized approaches, using results reported in prior publications for fair comparison. Our primary baselines are OpenVLA [12] (7.7B parameters), π_0 [2] (3.3B parameters), π_0-FAST[23] (3.3B parameters), and a standard Diffusion Policy using a CNN [6]. For the bi-manual manipulation tasks, we compare the BEAST-ACT variant with small action chunking models to a vanilla ACT [34], π_0, and a standard Diffusion Policy using a CNN.

Results. Table I summarizes the performance of all policies on the CALVIN benchmark, where BEAST-F outperforms a diverse set of baselines across two settings, establishing a

Train→Test	Method	PrT	Action Type	VLM	No. Instructions in a Row (1000 chains)					Avg. Len.
					1	2	3	4	5	
ABC→D	Diff-P-CNN [6]	×	Diffusion	×	63.5%	35.3%	19.4%	10.7%	6.4%	1.35
	MDT [27]	×	Diffusion	×	63.1%	42.9%	24.7%	15.1%	9.1%	1.55
	OpenVLA [12]	✓	Discrete	✓	91.3%	77.8%	62.0%	52.1%	43.5%	3.27
	3DDA [11]	×	Diffusion	×	93.8%	80.3%	66.2%	53.3%	41.2%	3.35
	MoDE [28]	✓	Diffusion	×	96.2%	88.9%	81.1%	71.8%	63.5%	4.01
	VPP [10]	✓	Diffusion	×	95.7%	91.2%	86.3%	81.0%	**75.0%**	4.29
	BEAST-F (ours)	×	Discrete	✓	**99.8%**	**96.5%**	**89.3%**	**82.7%**	74.4%	**4.42**
ABCD→D	Diff-P-CNN [6]	×	Diffusion	×	86.3%	72.7%	60.1%	51.2%	41.7%	3.16
	MoDE [28]	✓	Diffusion	×	97.1%	92.5%	87.9%	83.5%	77.9%	4.39
	MDT [27]	×	Diffusion	×	**98.6%**	95.8%	91.6%	86.2%	80.1%	4.52
	BEAST-F (ours)	×	Discrete	✓	98.1%	**96.2%**	**93.0%**	**89.3%**	**84.8%**	**4.61**

TABLE I: **CALVIN Benchmark results for ABC and ABCD.** The table reports average success rates for individual tasks within instruction chains and the average rollout length (Avg. Len.) to complete 5 consecutive instructions, based on 1000 chains. Zero standard deviation indicates methods without reported standard deviations. BEAST-F achieves SoTA performance in both tasks.

	Spatial		Object		Goal		Long		Average	
	SR (↑)	Rank (↓)	SR (↑)	Rank (↓)	SR (↑)	Rank (↓)	SR (↑)	Rank (↓)	SR (↑)	Rank (↓)
Diff-P-CNN	78.3 ± 1.1%	6	92.5 ± 0.7%	4	68.3 ± 1.2%	6	50.5 ± 1.3%	6	72.4 ± 0.7%	6
Octo	78.9 ± 1.0%	5	85.7 ± 0.9%	6	84.6 ± 0.9%	4	51.1 ± 1.3%	5	75.1 ± 0.6%	5
OpenVLA	84.7 ± 0.9%	4	88.4 ± 0.8%	5	79.2 ± 1.0%	5	53.7 ± 1.3%	4	76.5 ± 0.6%	4
π_0	**96.8%**	1	**98.8%**	1	**95.8%**	1	85.2%	2	94.2%	1
π_0-FAST	96.4%	2	96.8%	3	88.6%	3	60.2%	3	85.5%	3
BEAST-F	92.9 %	3	97.5 %	2	93.1 %	2	**86.4 %**	1	92.5%	2

TABLE II: **Experimental Results for the LIBERO Benchmarks.** SR: Success Rate. Best results in each column are shown in bold. BEAST-F achieves comparable performance state-of-the-art VLA, despite with a much smaller model and without robot data pretraining.

Method	Throughputs (Hz)↑	Latency (s)↓
DP (0.26B)	130.67	0.341
OpenVLA (7B)	6.09	0.164
π_0 (3.3B)	288.11	0.104
BEAST-F (0.77B)	**617.3**	**0.019**

TABLE III: **Mean inference efficiency** (1000 steps in Bf16). All policies except OpenVLA use chunking length 50.

new state of the art. Unlike the most competitive baselines, BEAST-F achieves these results without relying on additional pretraining. On the various LIBERO benchmarks, our tokenizer achieves strong performance, being surpassed only by π_0-VLA. However, π_0 relies on large-scale pretraining to reach its performance, whereas BEAST-F remains competitive without it. In the most challenging long-horizon task setting, LIBERO-LONG, BEAST-F outperforms all baselines. See Table II for detailed results. For the bi-manual tasks (Figure 6), BEAST-ACT and ACT demonstrate significantly better performance than π_0. BEAST-ACT achieved a higher success rate than vanilla ACT in both tasks.

C. Advantages in Training and Inference Speed

Next, we verify the inference and training efficiency of BEAST to answer **RQ3**. Specifically, we consider the VLA variant **BEAST-F** and compare it against several recent VLAs [12, 2, 23], as well as a standard CNN-based Diffusion Policy[6]. We measure the inference efficiency on an RTX 4090 GPU. As shown in Table III, BEAST-F demonstrates clear computational advantages. It achieves a throughput of 617.3 Hz (e.g., generates approximately 617 actions per second), which is $2.14\times$ faster than π_0, $4.72\times$ faster than Diffusion Policy, and $101.4\times$ faster than OpenVLA. In addition, BEAST-F achieves

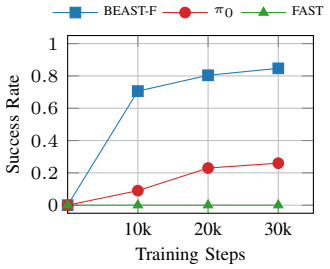

Fig. 7: LIBERO-LONG.

the lowest latency at just 19 milliseconds, where latency refers to the time taken to generate one action chunk. These gains are due to the parallel decoding, which enables generating the action sequence in a single forward pass.

We further evaluate the training efficiency by comparing BEAST-F against π_0 and π_0-FAST. To exclude the bias introduced by the pretraining datasets, we trained all models **without robot dataset pretraining**. We report the success rate on LIBERO-LONG benchmark every 10k training steps in Figure 7. BEAST-F reaches a approximate 80% success rate at just 20k steps, whereas π_0 reaches only around 20% at the same point. Notably, π_0-FAST shows no success till 30k steps. π_0-FAST's poor performance indicates a heavy reliance on robot dataset pretraining, which further underscores the training efficiency of our method.

D. Real-World Evaluation with 3 Different Robot Setups

To answer **RQ4**, we assess the effectiveness of BEAST across diverse real-world scenarios with varying data collection frequencies. We evaluate BEAST on 8 challenging manipulation tasks across 3 different experimental setups: 1) **Franka**

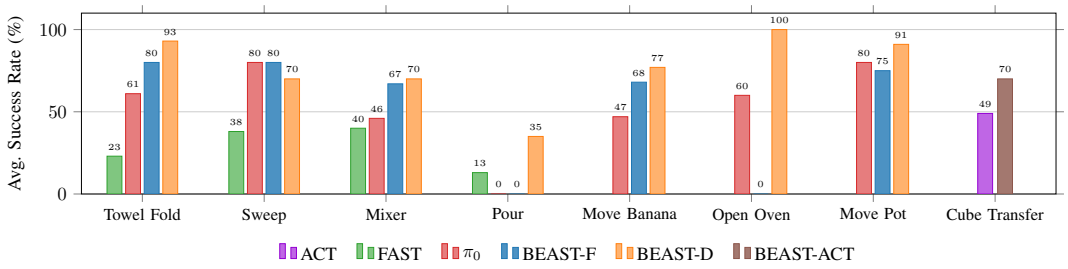

Fig. 8: **Experimental Results on Real-World Robot Tasks.** This figure shows the average task success rate across eight real-world tasks. Each task and method was evaluated over 10 runs (30 runs for Cube Transfer). Success rates are measured at the sub-task level.

Challenge: Four tabletop manipulation tasks (Towel Fold, Sweep, Mixer, Pour) using a joint position-controlled Franka robot with data collected at 20Hz, 2) **Real Kitchen**: Three manipulation tasks on a toy kitchen setup (Move Banana, Open Oven, Move Pot) with data collected at 35Hz, 3) **Bi-manual ALOHA**: A cube transfer task using a bi-manual ALOHA robot with data recorded at 60Hz. For each task in the **Franka Challenge** and **Franka Kitchen** setups, we conduct 10 evaluation runs per method, while for the **ALOHA cube transfer** task, we performed 30 runs. The average success rate for each task is reported in Figure 8. For tasks comprising multiple stages, we track intermediate milestones to better evaluate the completion of each sub-task. Appendix C provides a detailed description of all setups and tasks. We compare BEAST against π_0[2], π_0-FAST[23], and ACT [34]. We finetune π_0 and π_0-FAST from the official pretrained checkpoints for an additional 60k and 40k steps, respectively. For each method, we train one multitask model for all four tasks, Real Franka tasks, and another for the Real Kitchen tasks. The results demonstrate that BEAST-F achieves 52.86% success rate and BEAST-D achieves 76.57%. In contrast π_0 achieves 53.43% and FAST only 28.5%. Interestingly, the smaller model (BEAST-D) outperforms all the VLAs, including the Florence variant with BEAST. We attribute this effect to the relatively small real-world dataset of only 50 demonstrations for each task. For the Aloha Cube Transfer task, we compare BEAST-ACT against the base ACT that directly predicts action sequences in the joint space. BEAST-ACT achieves 70% success, which is 21% higher than the base ACT.

E. Ablation Studies

To answer **RQ5**, we conduct ablation studies to analyze the impact of various design choices of BEAST. All experiments in this section use the Florence variant of BEAST and are evaluated on the CALVIN ABC benchmark. All results are summarized in Table IV.

BEAST vs. Binning-based Tokenizer. We first compare BEAST against a commonly used binning-based tokenizer in VLAs[12], which discretizes single-step actions into one of 256 uniformly distributed bins. We implement this baseline using the same Florence-2 backbone and denote it as **Binning-F**. It is trained to perform autoregressive token prediction. As shown in Table IV, BEAST significantly outperforms the binning-based approach, improving the average sequence length from 1.41 to 4.43, underscoring the effectiveness of BEAST as an action tokenizer.

Variant	Avg. Len.
BEAST-F [10]	**4.43**
BEAST-F [5]	3.88
BEAST-F [15]	4.14
BEAST-F [20]	2.71
BEAST-SF	3.98
BEAST-CT	3.93
Binning-F	1.41

TABLE IV: Average Sequence Lengths for BEAST-F Ablations on CALVIN ABC.

Discrete Tokens vs. Continuous Tokens. Next, we study the choices between using discrete tokens or continuous tokens (denoted as **BEAST-CT**) as the action representation. In the continuous variant, the final hidden states of the Florence decoder are directly mapped to continuous BEAST tokens via a linear layer, and the learning objective is changed from cross-entropy to L1 regression loss. Results show that discrete tokens yield 12.7% better performance. We attribute this to the greater expressiveness of discrete representations, which are better suited to model multi-modal distributions.

Choice of Number of Basis Functions. Next, we evaluate how the number of basis functions affects the policy performance. We evaluate using $N = [5, 10, 15, 20]$ basis functions to model action chunks of 20 steps, denoted as **BEAST-F [N]** in Table IV. Fewer basis functions lead to fewer tokens for prediction, but it also reduces the expressiveness of the B-Spline representation. On the contrary, more basis functions increase representational power but reduce compression, which can also negatively influence the performance.

Scaling with Model Size. Finally, we assess the impact of model size on task performance. We compare BEAST-F, which uses Florence-2-large (0.77B parameters), with **BEAST-SF**, a smaller variant based on Florence-2-base (0.23B parameters). The larger model achieves an 11.3% improvement in average sequence length, demonstrating that BEAST benefits from increased model capacity. This result highlights its potential as a scalable building block for larger VLAs.

REFERENCES

[1] Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Roboagent:

Generalization and efficiency in robot manipulation via semantic augmentations and action chunking, 2023.

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[3] C de Boor. Subroutine package for calculating with b-splines. In *Los Alamos Sci. Lab.* Los Alamos, NM, USA, 1971.

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*, 2022.

[5] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[6] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[7] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. *Advances in neural information processing systems*, 32, 2019.

[8] William J. Gordon and Richard F. Riesenfeld. B-spline curves and surfaces. *Computer Aided Geometric Design*, pages 95–126, 1974. URL https://api.semanticscholar.org/CorpusID:118698454.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Yucheng Hu, Yanjiang Guo, Pengchao Wang, Xiaoyu Chen, Yen-Jen Wang, Jianke Zhang, Koushil Sreenath, Chaochao Lu, and Jianyu Chen. Video prediction policy: A generalist robot policy with predictive visual representations, 2024. URL https://arxiv.org/abs/2412.14803.

[11] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=gqCQxObVz2.

[12] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[13] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

[14] Lucy Lai, Ann Zixiang Huang, and Samuel J Gershman. Action chunking as policy compression. 2022.

[15] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. In *International Conference on Machine Learning*, pages 26991–27008. PMLR, 2024.

[16] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

[17] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems XVII*, 2020. URL https://api.semanticscholar.org/CorpusID:235657751.

[19] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

[20] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[21] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[22] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[23] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

[24] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*. Springer Science & Business Media, 2002.

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[26] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[27] Moritz Reuss, Ömer Erdinç Yağmurlu, Fabian Wenzel, and Rudolf Lioutikov. Multimodal diffusion transformer: Learning versatile behavior from multimodal goals. In *Robotics: Science and Systems*, 2024.

[28] Moritz Reuss, Jyothish Pari, Pulkit Agrawal, and Rudolf Lioutikov. Efficient diffusion transformer policies with mixture of expert denoisers for multitask learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=nDmwloEl3N.

[29] 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.

[30] Andrew Szot, Bogdan Mazoure, Harsh Agrawal, R Devon Hjelm, Zsolt Kira, and Alexander Toshev. Grounding multimodal large language models in actions. *Advances in Neural Information Processing Systems*, 37:20198–20224, 2024.

[31] Andrew Szot, Bogdan Mazoure, Omar Attia, Aleksei Timofeev, Harsh Agrawal, Devon Hjelm, Zhe Gan, Zsolt Kira, and Alexander Toshev. From multimodal llms to generalist embodied agents: Methods and lessons. *arXiv preprint arXiv:2412.08442*, 2024.

[32] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016.

[33] Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4818–4829, 2024.

[34] 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.

APPENDIX

A. Architectures

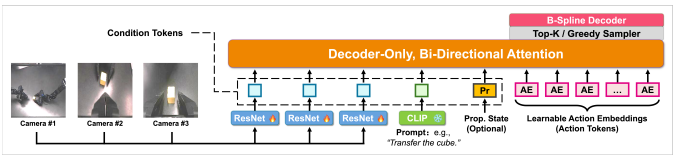

Fig. 9: **Overview of BEAST-D**: BEAST-D is a small transformer model that integrates BEAST. It replaces the causal attention in the decoder-only transformer with bidirectional attention to enable fast parallel decoding. BEAST-D integrates ResNet as image encoder and CLIP as language encoder.

B. Baselines Implementation

ACT: Action Trucking Transformer (ACT) adopts an encoder-decoder architecture similar to CAVE. It uses a transformer encoder to map the input action sequence into a latent vector. During inference, a latent variable is sampled to represent the mode variation. Combined with image conditions, this latent variable is fed into a transformer decoder to generate a chunk of actions. We use the implementation provided by Lerobot as our baseline.

π_0: π_0 is a generalist robot policy that combines a pre-trained VLM backbone with a lightweight *action expert* module trained from scratch to generate continuous actions using *flow matching*. A key innovation of π_0 is its cross-embodiment training strategy, which integrates over 900M timesteps of data from 7 distinct robot embodiments and 68 manipulation tasks, enabling generalization across heterogeneous hardware platforms. The model is trained using a two-phase pipeline: a large-scale pre-training stage leveraging Internet-scale semantic priors, followed by post-training on curated task-specific data to enhance performance on complex, dexterous tasks.

FAST: FAST introduces a novel compression-based tokenization method, named *Frequency-space Action Sequence Tokenization*, for training autoregressive VLA models on high-frequency, dexterous robot control tasks. Unlike prior VLAs that struggle with discretizing continuous actions at high frequencies, FAST leverages the *Discrete Cosine Transform (DCT)* and *Byte-Pair Encoding (BPE)* to produce compact, information-rich action tokens, marking a significant advance in training efficiency.

C. Robot System Details

Real Kitchen. This setup consists of a single Franka Emika robot operating within a simulated kitchen environment. It is equipped with two OAK-D Lite cameras providing top-down and side perspectives, each delivering visual input at a resolution of 250×250 pixels. The robot has an 8-dimensional configuration and action space, which includes seven joint and one gripper states.

Real Franka. This configuration features a single Franka Emika robot situated in a general-purpose tabletop environment

designed for more challenging manipulation tasks. Visual observations are obtained from two Orbbec Femto Bolt cameras, positioned to capture left and right perspectives. The input images are resized to a resolution of 180×320 pixels. The robot configuration and action space remain the same as the Franka Kitchen setup.

ALOHA. Based on the ALOHA setup [34], the system incorporates two 6-DoF Trossen ViperX robotic arms. The environment includes two wrist-mounted and an additional top-mounted Logitech C920 camera. The combined system operates in a 14-dimensional configuration and action space, accounting for both arms' joint and gripper states.

D. Tasks Description and Evaluation Metrics

In the Real Kitchen setup, the robot performs pick-and-place tasks, whereas in the Real Franka setup, the robot is required to execute more diverse manipulation behaviors, such as sweeping or pouring. For each task performed by the Franka Emika robot, a scoring rubric is defined to quantitatively evaluate task progression. The specific evaluation criteria for each task are detailed below.

- **Open the door:** The task begins under one of two initial conditions: with or without an object placed on the stove. The objective is for the robot to open the oven door. Task completion is evaluated as a success or a failure. Although the task involves three key motion phases, as shown in Figure 10 (Open the door), all the policies under evaluation are capable of completing the task in its entirety once the robot successfully grasps the handle. A trial is considered successful if the robot fully opens the oven door by first grasping the handle and then using its fingers to push the opposite side of the door, ensuring that it is completely open. We conducted four evaluation trials with no object on the stove and one with a randomly placed object.
- **Banana into the pot:** In this task, the robot aims to grasp a banana and place it on or into a pot. The initial conditions are categorized based on the relative positions of the pot and the banana, as well as the position of the banana relative to the corresponding stove. The pot is assumed to be correctly positioned on the stove. The banana, however, may be placed directly on the stove in one of three orientations or slightly offset to the left or right. A total of 10 trials are conducted across these different initial configurations. Task performance is scored on a scale of 0 to 3, with one point awarded for each of the following criteria: (1) successfully positioning the banana between the robot's two fingers, (2) lifting the banana off the surface, and (3) placing the banana onto or into the pot. If the robot attempts to grasp more than three times, exhibits jerky hand movements, or significantly displaces the pot from its original position, the subtask is awarded 0.5 points to reflect partial completion.
- **Pot into the sink:** This task is similar to the one described previously. The initial conditions in this task differ based on the relative position between the pot and the two stoves. Task performance is evaluated using three criteria: (1) successfully positioning the pot between the robot's two fingers - note that directly grasping the pot with its handle is considered an unstable grasp and is awarded 0.5 points, (2) lifting the pot off the surface, and (3) placing the pot in the sink. In this task, penalties for jerky hand movements are still applied.
- **Towel folding:** The objective of this task is to neatly fold a towel that is randomly oriented at the beginning of each trial. One point is awarded for each of the following: lifting a corner of the towel, completing the fold, and achieving accurate alignment of the folded towel.
- **Sweep:** In the Sweep task, the positions of the broom, dustpan, and trash vary across trials. Four pieces of trash are placed on the table for the robot to clean. A maximum of four points can be awarded, based on the following criteria: successfully grasping the broom, performing a single sweeping motion, demonstrating the ability to execute multiple sweeping motions, and sweeping all trash into the dustpan.
- **Mixer:** In this task, a cup and a mixer are placed on the table. The robot's objective is to sequentially (1) open the mixer, (2) grasp the cup, (3) place the cup on the mixer's platform, and (4) close the mixer. Task performance is evaluated based on the successful completion of these four subtasks, with one point awarded for each. Notably, unlike previous tasks, the language instructions provided to the robot consist of three separate sentences corresponding to the actions of opening/closing the mixer and placing the cup onto the platform.
- **Pour:** In the Pour task, the source cup is initially placed on a platform and contains plastic pellets that simulate liquid. The objective is to pour the pellets into a designated target cup. Task performance is evaluated out of a maximum of 4 points, awarded based on the following criteria: (1) successfully grasping the source cup, (2) pouring the pellets into the target cup, (3) ensuring that all pellets are poured into the target cup, and (4) placing the source cup back on the platform.
- **ALOHA cube transfer:** In the cube transfer task, the ALOHA robot is designed to pick up a randomly placed cube using its right arm and then transfer the cube to its left arm. The performance of the task is evaluated by assigning scores to three specific steps: (1) successfully picking up the cube, (2) successfully initiating the transfer with the left arm making contact with the cube, and (3) the left arm successfully taking possession of the cube while the right arm releases it.

E. Compute Resources

We train and evaluate all the models based on our private clusters. Each node contains 4 NVIDIA A100, for BEAST-F we use 4 GPUs for training. For BEAST-D and BEAST-ACT, we use one GPU for training. We report the average training cost in Table V.

	BEAST-F	BEAST-D	BEAST-ACT
vRAM	64GB	8GB	15GB
steps/hour	6000	10000	11000

TABLE V: Training time for each variant.

F. Hyperparameters

Hyperparameter	REAL KITCHEN	REAL FRANKA
Action Sequence Length	80	20
Number of Basis	15	5
Vocabulary Size	256	256
Optimizer	AdamW	AdamW
Betas	[0.9, 0.95]	[0.9, 0.95]
Learning Rate	2e-5	2e-5
Batch Size	96	96
Train Steps (k)	60	60

TABLE VI: BEAST-F hyperparameters for real robot experiments.

Hyperparameter	REAL KITCHEN	REAL FRANKA
Action Sequence Length	80	20
Number of Basis	10	5
Vocabulary Size	256	256
Transformer Layers	6	6
Attention Heads	8	8
Embedding Dim	256	256
Image Encoder	FiLM-ResNet18	FiLM-ResNet18
Goal Lang Encoder	CLIP ViT-B/32	CLIP ViT-B/32
Attn Dropout	0.1	0.1
Residual Dropout	0.1	0.1
MLP Dropout	0.1	0.1
Optimizer	AdamW	AdamW
Betas	[0.9, 0.999]	[0.9, 0.999]
Learning Rate	3e-4	3e-4
Weight Decay (Trans/Other)	0.05 / 0.05	0.05 / 0.05
Batch Size	384	256
Train Steps (k)	60	60
EMA	False	False

TABLE VII: BEAST-D hyperparameters for real robot experiments.

Hyperparameter	**LIBERO**				**CALVIN**	
	SPATIAL	OBJECT	GOAL	LONG	ABCD→D	ABC→D
Action Sequence Length	20	20	20	20	20	20
Number of Basis	10	10	10	10	10	10
Vocabulary Size	256	256	256	256	256	256
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Betas	[0.9, 0.95]	[0.9, 0.95]	[0.9, 0.95]	[0.9, 0.95]	[0.9, 0.95]	[0.9, 0.95]
Learning Rate	2e-5	2e-5	2e-5	2e-5	2e-5	2e-5
Batch Size	128	128	128	128	32	32
Train Steps (k)	35	35	50	70	30	30

TABLE VIII: Summary of BEAST-F hyperparameters for all simulation experiments.

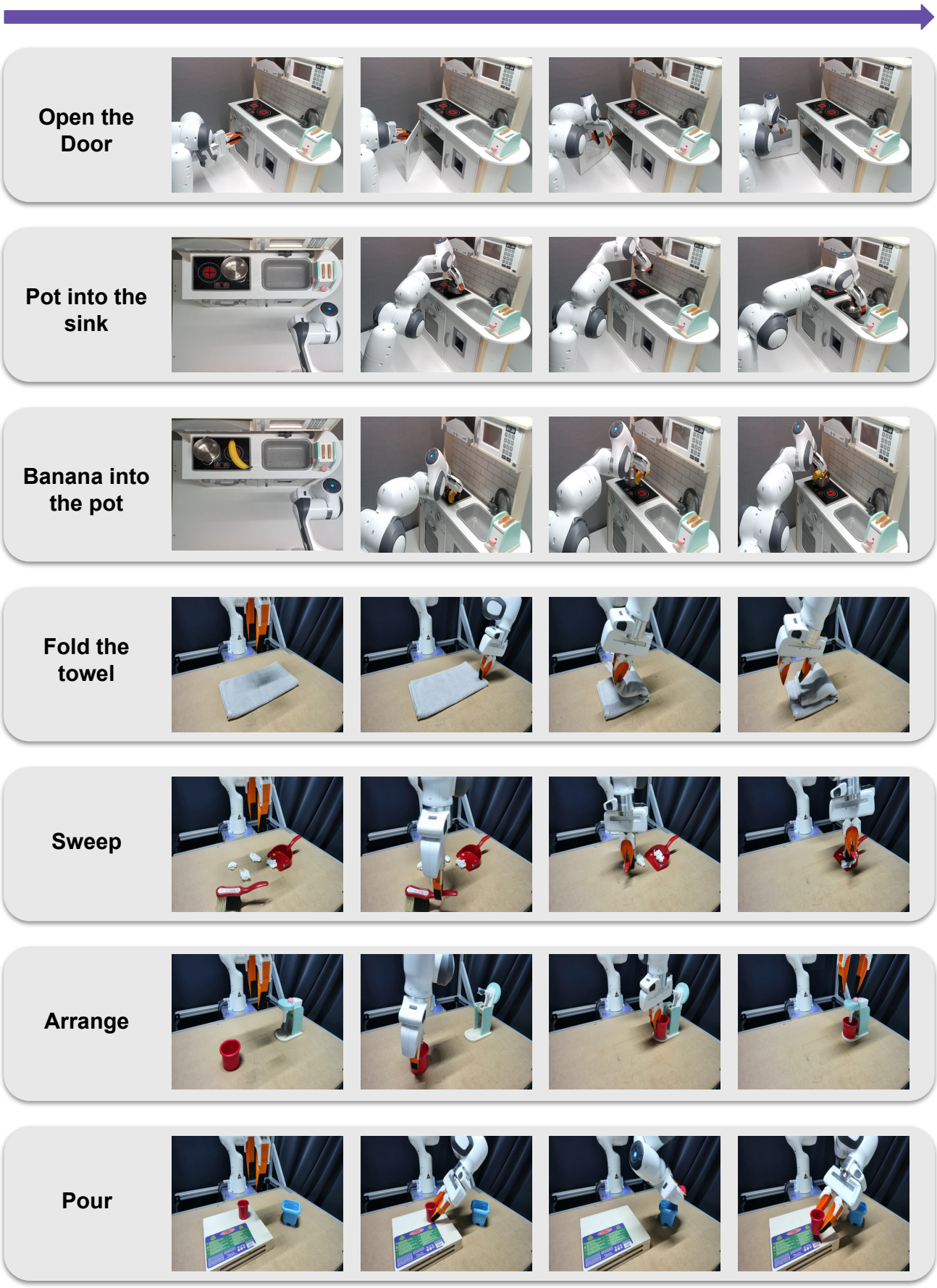

Fig. 10: **Key frames for real world different tasks**