Goal-Conditioned Imitation Learning using Score-based Diffusion Policies

Moritz Reuss, Maximilian Li, Xiaogang Jia and Rudolf Lioutikov
Intuitive Robots Lab, Karlsruhe Institute of Technology, Germany

***Abstract*—We propose a new policy representation based on score-based diffusion models (SDMs). We apply our new policy representation in the domain of Goal-Conditioned Imitation Learning (GCIL) to learn general-purpose goal-specified policies from large uncurated datasets without rewards. Our new goal-conditioned policy architecture "BEhavior generation with ScOre-based Diffusion Policies" (BESO) leverages a generative, score-based diffusion model as its policy. BESO decouples the learning of the score model from the inference sampling process, and, hence allows for fast sampling strategies to generate goal-specified behavior in just 3 denoising steps, compared to 30+ steps of other diffusion-based policies. Furthermore, BESO is highly expressive and can effectively capture multi-modality present in the solution space of the play data. Unlike previous methods such as Latent Plans or C-Bet, BESO does not rely on complex hierarchical policies or additional clustering for effective goal-conditioned behavior learning. Finally, we show how BESO can even be used to learn a goal-independent policy from play-data using classifier-free guidance. To the best of our knowledge this is the first work that a) represents a behavior policy based on such a decoupled SDM b) learns an SDM-based policy in the domain of GCIL and c) provides a way to simultaneously learn a goal-dependent and a goal-independent policy from play data. We evaluate BESO through detailed simulation and show that it consistently outperforms several state-of-the-art goal-conditioned imitation learning methods on challenging benchmarks. We additionally provide extensive ablation studies and experiments to demonstrate the effectiveness of our method for goal-conditioned behavior generation. Demonstrations and Code are available at https://intuitive-robots.github.io/beso-website.**

I. Introduction

Goal-conditioned Behavior Learning aims to train versatile embodied agents, that can handle a wide range of daily tasks. A common approach to tackle this challenge is Goal-conditioned Imitation Learning (GCIL). GCIL only requires an offline dataset without additional rewards or expensive environment interactions for training. However, GCIL typically requires a set of predefined tasks and a large number of labeled and segmented expert trajectories for each task, which can be costly and time-consuming. Additionally, it does not generalize well to new scenes and different tasks. Instead of teaching an agent a limited number of predefined goals, *Learning from Play* (LfP) [21] provides an effective way of collecting task-agnostic, teleoperated, uncurated, freeform datasets. Such datasets consist of rich, meaningful, multimodal interactions with the environment that cover different areas of the state space. Instead of manually labeling the trajectories, LfP pairs random sequences of each trajectory with one or more future states, i.e., the goal state, of the respective trajectory. Goal-conditioned policies distill useful, goal-oriented behavior from this collected play interaction data. However, learning from play data remains an open challenge, partially due to the multimodal nature of the demonstrations, e.g., the same task can be solved in very different ways and different tasks can be solved in very similar ways.

Effective behavior learning from these datasets demands policies that maintain such multimodal solutions and that are expressive enough to remain close to the seen state-action distribution of the offline data for executing long-term horizon skills. Most prior work tries to deal with this challenge, by combining generative models, such as Variational Autoencoders (VAEs) [12, 25, 32] and Generative Pretrained Transformer (GPTs) [6, 35], with additional models and networks to explicitly encode multimodality or hierarchy. However, these methods require supplementary networks or separation of skill execution and planning within their architecture, as the policy expression is not sufficient or cannot handle the multimodality of the observed behaviors. Additionally, multiple learning objectives are typically required, e.g. for low- and high-level policies, which provides additional tuning challenges.

We propose a novel approach, **BE**havior Generation using **S**c**O**re-based Diffusion models (BESO), which excels in learning goal-conditioned policies solely from reward-free, offline datasets. BESO uses Score-based Diffusion Models (SDMs) [37, 15, 41, 17], a new class of generative models, that progressively diffuse data to noise through a forward Stochastic Differential Equation (SDE). By training a neural network, known as the score or denoising model, to approximate the score function, one can reverse the SDE to generate new samples from noise in an iterative sampling process.

We demonstrate several benefits of modeling the goal-conditioned action distribution using a score-based diffusion model. First, we show, that the expressiveness of SDMs and their ability to capture multimodal distributions is key for effective conditioned behavior generation. On several challenging goal-conditioned benchmarks, including the conditioned Relay Kitchen and Block-Push environment [6], BESO consistently outperforms state-of-the-art methods such as C-BeT and Latent Motor Plans [6, 21]. Second, by leveraging Classifier-Free Guidance Training of SDMs, BESO effectively learns two policies simultaneously: a goal-dependent policy and a goal-independent policy, which both can be used together or independently at test time. Third, our model is easy and stable to train with a single training objective without additional rewards. This contrasts with other state-of-the-art generative models, such as Implicit Behavior Cloning (IBC) [10], or hierarchical policies [12]. Fourth, SDMs do not restrict the choice of the model architecture as in other generative models such as VAEs or energy-based models (EBMs) [10]. Thus, we apply a novel Transformer architecture augmented with preconditioning to synthesize step-based actions given a sequence of observations and desired goal states. Finally, BESO can diffuse new actions fast. While current diffusion-based policies [30] require 30+ denoising steps for a single action prediction to achieve good results, our proposed approach, BESO, performs exceptionally well on challenging GCIL benchmarks, outperforming state-of-the-art goal-conditioned policies, while using only 3 denoising steps. We achieve this, by using recent advances in Score-based Diffusion Models, which separate the training and inference process [17] and applying novel numerical solvers designed for fast diffusion inference [38, 19]. Therefore, we systematically evaluate the essential components of SDMs for fast and effective step-based action generation.

To summarize our contributions:

- BESO, a new policy representation based on score-based diffusion models for effective goal-conditioned behavior generation from uncurated play data
- Use of Classifier-Free Guidance based Diffusion Policy to simultaneously learn a goal-dependent and goal-independent policy from play
- Systematic evaluation of key components for fast and efficient action generation using Score-based Diffusion policies combined with extensive experiments and ablation studies

II. Related Work

Diffusion Generative Models. Score-based generative models (SGMs) [39, 40] and Denoising Diffusion Probabilistic Models (DDPMs) [37, 15] are two different variants of score-based diffusion models (SDMs). These models corrupt a data distribution with increasing Gaussian noise and use neural networks to learn to reverse this corruption to generate new data samples from noise. The two different models have been unified using the stochastic differential equation (SDE) framework [41]. SDEs describe the diffusion process as a time-continuous process instead of using discrete noise levels. BESO follows the SDE formulation proposed by Karras et al. [17]. To draw new samples from the diffusion models, they need to reverse the SDE discretized over T time steps. The SDE contains a *probability flow* ODE with the same marginal distributions, which allows for fast sampling [41]. ODE solvers do not add noise during the inference process, which can reduce the number of function evaluations and accelerate sampling [19]. Sampling can be further accelerated using specialized numerical ODE solvers designed for diffusion inference [15, 17, 20]. SDMs achieved state-of-the-art results in various tasks including image generation [17], text-based image synthesis [7, 33] and human motion generation [42].

Goal-Conditioned Imitation Learning (GCIL). It is a subdomain of Imitation Learning [29, 2], where each demonstration is augmented with one or more goal-states that are indicative of the task that the demonstration was provided for. The goal-state contains information that a learning method can leverage to disambiguate demonstrations. Consequently, a goal-conditioned policy, i.e., a policy that includes the goal-state in its condition set, can use a given goal-state to adapt its behavior accordingly. Similarly, goal-states have also extended the domain of reinforcement learning through Goal-Conditioned Reinforcement Learning (GCRL) [8, 9, 22, 32], where the agent is not provided expert demonstrations but reward signals instead. Typically these reward signals are difficult to define, especially for complex tasks and environments, providing demonstrations is often a more natural option in such situations. Additionally, the policy rollouts required by GCRL are often expensive in real-world settings. Recent work investigated Goal Conditioned Offline Reinforcement Learning [22, 34, 32, 46, 26], which does not require these expensive rollouts during training.

Learning from Play. The goal of *Learning from Play* (LfP) [21] is to learn goal-specified behavior from a diverse set of unlabeled state-action trajectories. Classical imitation learning datasets typically consist of uni-modal, segmented expert trajectories in a narrow state-space. Play data, on the other hand, is characterized by unsegmented, multimodal trajectories. This makes learning meaningful behaviors more challenging, as the policies need the ability to deal with multiple ways of solving a task, distinguish between similar ways to solve different tasks, as well as the ability of long-horizon planning to reach goals far into the future. Prior work aimed to extract representations from play data for effective downstream policy learning [47] or learned self-supervised representations of skills, referred to as latent plans, using Conditional Variational Autoencoders (CVAE) [12, 21, 25, 24]. Transformer-based architectures were also researched as a policy class for task-agnostic behavior learning [6, 35]. Another body of work tries to improve LfP, by focusing on the data aspect and learning from object-centric interactions, instead of randomly sampled sequences [3].

Generative Models in Policy Learning. Imitation Learning can be formulated as a state-occupancy matching problem, where the goal is to learn a policy that matches the state-occupancy distribution of expert demonstrations. The unknown expert demonstration can now be approximated through modern generative model architectures. One popular approach is the use of Generative Adversarial Networks (GANs) [13, 11]. These methods consist of a generator policy that learns to imitate the observed behavior of the expert and a discriminator, which distinguishes between real and fake trajectories. They require extensive rollouts during training, which is not feasible in our setting. Other approaches use CVAEs [23, 32, 12, 24, 34] to learn a latent embedding to represent the underlying skills. Recent work also applied Energy-based models as conditional policies for behavior cloning [10]. Normalizing flows have also been proposed as a policy representation [36].

Diffusion Generative Models in Robotics. Most approaches that apply diffusion models in robotics applications focus on the discrete DDPM variant [15]. The DDPM Diffusion model has been used in Offline-RL to generate state-

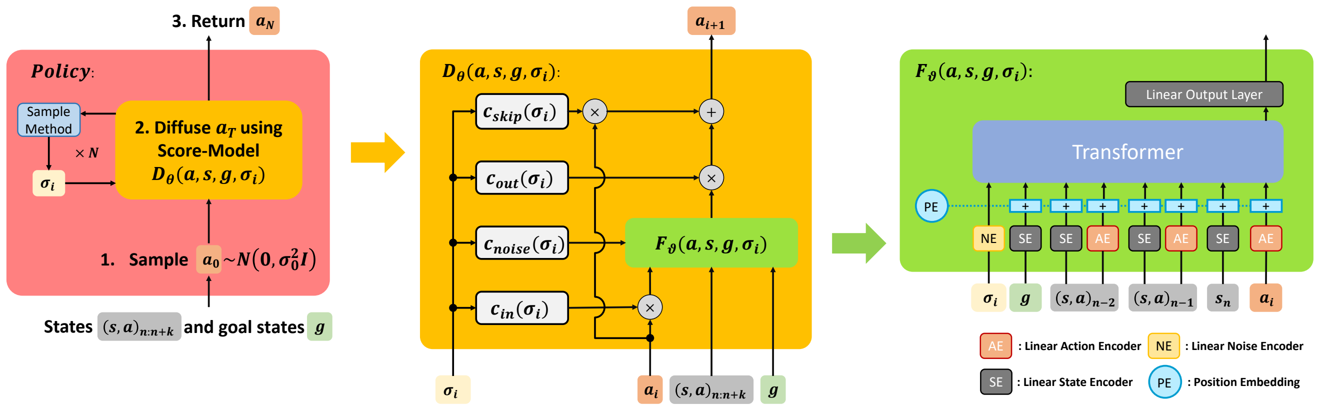

Fig. 1. Overview of the action generation process of BESO with the used model architecture. **Left:** General Action Generation Process using the Diffusion Process to denoise the next action given the current observation $\boldsymbol{s}$ and desired goal state $\boldsymbol{g}$ in N-steps. **Middle:** the high-level score-model with its pre-conditioning layers and skip connections. **Right:** the internal denoising score model, which uses a transformer architecture with causal masking to iteratively predict the denoised action given the sequence of prior observations, actions, and the goal sequence.

action or state-only trajectories using large U-net architectures [16, 1]. DDPM has also been applied as a policy regularization method in a step-based Offline-RL setting in combination with a learned Q-function [45]. Recently, score-based generative models have been leveraged to synthesize cost functions for grasp pose configurations [43]. In addition, Conditional score-based generative models have been proposed to learn the reward function for inverse reinforcement learning [18]. The closest related work to BESO is Diffusion Policy [5] and Diffusion-BC [30], which both propose the use of conditional, discrete DDPM as a new policy class for Behavior Cloning. Diffusion-BC synthesizes new actions in 50 stochastic sampling steps. To improve the performance, Diffusion-BC uses X-extra inference steps at the lowest noise level without additional noise. However, this method results in even slower action generation. BESO leverages the *probability flow* ODE combined with fast, deterministic samplers and optimized noise levels. Hence, BESO requires significantly fewer function evaluations in every action prediction.

III. Problem Formulation and Method

In this section, we describe our approach to goal-conditioned behavior generation using Score-based diffusion models.

A. Problem Formulation

The Goal of GCIL is to learn a general-purpose goal-conditioned policy from uncurated play data. Given a set of unstructured, task-agnostic trajectories, $\mathcal{T} = \left\{\boldsymbol{\tau}_k | \boldsymbol{\tau}_k = ((\boldsymbol{s}_n^k, \boldsymbol{a}_n^k))_{n=1}^{N_k}\right\}$, each trajectory can be split into a set of tuples containing sub-trajectory sequences and goal-states $\mathcal{D}_k = \left\{(\boldsymbol{o}, \boldsymbol{g}) | \boldsymbol{o} = (\boldsymbol{s}_n, \boldsymbol{a}_n)_{n=i}^{i_\dagger}, \boldsymbol{g} = (\boldsymbol{s}_n)_{n=j}^{j_\dagger}, (\boldsymbol{s}_n, \boldsymbol{a}_n) \in \boldsymbol{\tau}_k\right\}$, with $i \leq i_\dagger < j \leq j_\dagger$ denoting start and end steps of the sequence and goal-state respectively. As this formulation makes clear, the goal-state has to be one or more states of the same trajectory as the sequence and has to begin at some step after the respective sequence has ended. The set $\mathcal{D}_k$ can contain overlapping sequences and the final play dataset is given as $\mathcal{D} = \bigcup_{k=1}^{K} \mathcal{D}_k$. For simplicity, the indices of $\boldsymbol{o}_k$ and $\boldsymbol{g}_k$ simply indicate that the sequence and goal state belong together and the indices in $(\boldsymbol{s}_n, \boldsymbol{a}_n) \in \boldsymbol{o}$ refer to the relative time step in the sequence. The state-action pairs in the sequence $\boldsymbol{o}_k$ leading to the goal state $\boldsymbol{g}_k$ are now treated as the optimal behavior to reach $\boldsymbol{g}_k$[12, 25]. Goal-conditioned policies try to maximize the log-likelihood objective over the play dataset

$$\mathcal{L}_{\text{play}} = \mathbb{E}_{(\boldsymbol{o},\boldsymbol{g})\in\mathcal{D}}\left[\sum_{(\boldsymbol{s},\boldsymbol{a})\in\boldsymbol{o}} \log \pi_\theta\left(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g}\right)\right]. \tag{1}$$

Because of the multi-modal nature of the demonstrations, i.e. several trajectories leading to the same goal state, solving this objective successfully requires a policy that is capable of encoding such a multi-modal behavior.

B. Score-based Diffusion Policies

We now aim to learn the policy distribution $\pi_{\mathcal{D}}\left(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g}\right)$ underlying the play dataset $\mathcal{D}$ and, hence, the given demonstrations. We do so by defining a continuous diffusion process, which maps samples from our play dataset by gradually adding Gaussian noise to the intermediate distributions $p_t, t \in [0, T]$ with initial distribution $p_0 = \pi_{\mathcal{D}}$ and final distribution p_T.

The continuous diffusion process can be described using a stochastic-differential equation (SDE) [41]. In this work, we define the SDE analogously to a recently introduced formulation [17]:

$$d\mathbf{a} = \left(\beta_t \sigma_t - \dot{\sigma}_t\right)\sigma_t \nabla_a \log p_t(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g})dt + \sqrt{2\beta_t}\sigma_t d\omega_t, \tag{2}$$

where $\nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{g})$ refers to the score-function, ω_t is the Standard Wiener process, which can be understood as infinitesimal Gaussian noise. The noise scheduler is denoted by σ_t, and $\beta(t)$ describes the relative rate at which the current noise is replaced by new noise. In our approach, we adopt $\sigma_t(t) = t$, a method proven effective in image generation [17]. At every timestep t and related noise level there exists a corresponding marginal distribution $p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$, which is the result of injecting Gaussian noise to samples from p_{play}. This can be expressed as $p_t(\boldsymbol{a}_t|\boldsymbol{a}) = \mathcal{N}(\boldsymbol{a}, \sigma_t^2 \mathbf{I})$. The final action distribution of the diffusion process is a known tractable prior distribution $\boldsymbol{a}_T = p_T$. An unstructured Gaussian distribution $p_T = \mathcal{N}(\mathbf{0}, \sigma_T^2 \mathbf{I})$ is chosen without any information about the play data distribution.

In the case of BESO we are particularly interested in the *Probability Flow* Ordinary Differential Equation (ODE) within the SDE [4]. This ODE shares the same marginal distributions $p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ as the SDE at every timestep, but without the additional random noise injections. By setting $\beta(t) = 0$, we recover the *Probability Flow* ODE from Eq. (2):

$$d\boldsymbol{a} = -\dot{\sigma}_t \sigma_t \nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}) dt \tag{3}$$

The negative score-function $-\nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ specifies the vector field of the current marginal distribution $p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$. This vector field points towards regions of low data density and is scaled with the product of the current noise level $\dot{\sigma}_t$ and the change of it $\dot{\sigma}_t$.

Algorithm 1 BESO Training

1: **Require:** Play Dataset $\mathcal{L}_{\text{play}}$, Sequence Size c_o, Goal Sequence Size c_g
2: **Require:** Score Model $\mathbf{D}_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t)$
3: **Require:** Noise Distribution $\sigma \sim \text{LogLogistic}(\alpha, \beta)$
4: **for** $i \in \{0, ..., N_{\text{train steps}}\}$ **do**
5: Sample $(\boldsymbol{o}, \boldsymbol{g}) \sim \mathcal{L}_{\text{play}}$
6: Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\sigma_{\text{mean}}, \sigma_{\text{std}}^2 \mathbf{I})$
7: $\mathcal{L}_{D_\theta} \leftarrow \mathbb{E}_{\sigma, \boldsymbol{a}, \boldsymbol{\epsilon}} \left[\alpha(\sigma_t) \| D_\theta(\boldsymbol{a} + \boldsymbol{\epsilon}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t) - \boldsymbol{a} \|_2^2 \right]$
8: **end for**

C. Diffusion Training

In order to generate new samples by numerically approximating the reverse ODE, we require an accurate estimate of the score function $\nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ for all marginal distributions p_t in our diffusion process. To achieve this, we use a neural network $D_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t)$ that matches the score for all marginal distributions $p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$.

$$\nabla_{\boldsymbol{a}} \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}) = \left(D_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t) - \boldsymbol{a} \right) / \sigma_t^2. \tag{4}$$

The neural network is trained using the denoising score matching objective [44, 39], where we add Gaussian noise to the actions and minimize the difference between the network's output and the original actions:

$$\mathcal{L}_{D_\theta} = \mathbb{E}_{\boldsymbol{\sigma_t}, \boldsymbol{a}, \boldsymbol{\epsilon}} \left[\alpha(\sigma_t) \| D_\theta(\boldsymbol{a} + \boldsymbol{\epsilon}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t) - \boldsymbol{a} \|_2^2 \right], \tag{5}$$

where $\boldsymbol{a}$ is an action sample, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_t^2 \mathbf{I})$ represents the Gaussian noise. The losses at individual noise levels are weighted according to $\alpha(\sigma_t)$, and the current σ_t is sampled from the noise training distribution p_{train}. We use a truncated log-logistic distribution with location parameter α and scale parameter β: $p_{\text{train}} \sim \text{LogLogistic}(\alpha, \beta)$ in the range of $\{\sigma_{\text{min}}, \sigma_{\text{max}}\}$. The training process is summarized in Alg. 1. This allows us to effectively learn the noise-conditioned score function for our diffusion process and generate samples from the conditional density, $p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$, using the Probability Flow ODE.

D. Efficient Action Generation using Deterministic Samplers

New actions are generated by our policy by sampling from the prior distribution $\boldsymbol{a}_T \sim \mathcal{N}(\mathbf{0}, \sigma_T^2 \mathbf{I})$ and numerically simulating the reverse ODE or SDE by substituting the score-function with our learned model in Eq. (3). The process begins by selecting a random sample from our prior distribution, $\boldsymbol{a}_T \sim \mathcal{N}(\mathbf{0}, \sigma_T^2 \mathbf{I})$, and then iteratively denoise this sample. Utilizing a random sample as a starting point enables the creation of diverse and multimodal actions, even when the underlying ODE is deterministic. The ODE can be solved numerically, by discretizing the differential equation starting from T to 0. During the action prediction, we iteratively denoise the sample at N-discrete noise levels. BESO employs the DDIM solver, as described in detail in Alg. 2 [20, 38], for fast, deterministic sampling. The solver is a first-order deterministic sampler that is based on an exponential integrator method. A detailed comparison of state-of-the-art diffusion samplers is provided in Sec. B of the Appendix, which concludes, that DDIM has the best overall performance. An additional evaluation on the influence of noise concludes that ODE solvers are competitive with SDE variants for action prediction tasks. Our ablation studies in Sec. B suggest that only three denoising steps are necessary for BESO to generate actions with high accuracy. Increasing the number of inference steps further only marginally enhances the performance, while significantly slowing down the sampling process. Thus, we found that 3 steps strike the best balance between computational efficiency and performance. For inference, we can adapt the range of noise and the distribution of discrete timesteps. Based on empirical evaluations, we decide to use exponential time steps with a noise range of $\sigma \in \{0.005, 1\}$ for most applications.

IV. Goal-Guided Score-based Diffusion Policies

In this section, we introduce two variants of BESO optimized for synthesizing actions for goal-conditioned behavior.

Conditioned Policy (C-BESO). We define a goal-conditioned diffusion policy, $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$, by directly learning the goal-and-state-conditioned distribution with our score-based generative model. In contrast to standard goal-conditioned behavior cloning, our diffusion policy allows us to capture multiple solutions present in the play data while still being expressive enough to solve long-term goals.

Goal-Classifier-Free Guided Policy (CFG-BESO). We additionally combine BESO with a popular conditioning method for diffusion models, Classifier-Free Guidance (CFG) [14].

Algorithm 2 Action Generation Process using DDIM based Sampler (DPM-1) adapted for BESO [19, 38]

Require: Current state $\boldsymbol{s}$, goal $\boldsymbol{g}$
Require: Score-Denoising Model $D_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma)$
Require: Discrete time steps $t_{i \in \{0,..,N\}}$
Require: Noise scheduler $\sigma_i = t_i$
Require: $f_\beta(t) = -\log(t)$, $f_t(\beta) = \log(-\beta)$
Draw sample $\boldsymbol{a}_0 \sim \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I})$
for $i \in \{0, ..., N-1\}$ **do**
  $\mathbf{d}_i \leftarrow \big(\boldsymbol{a}_i - D_\theta(\boldsymbol{a}_i, \boldsymbol{s}, \boldsymbol{g}, \sigma_i)\big)/\sigma_i$
  $\beta_{t_i}, \beta_{t_{i+1}} \leftarrow f_\beta(t_i), f_\beta(t_{i+1})$
  $h_i \leftarrow \beta_{t_i} - \beta_{t_{i-1}}$
  $\boldsymbol{a}_{i+1} \leftarrow (\frac{t_{i+1}}{t_i})\boldsymbol{a}_i - \big(e^{(-h_i)} - 1\big)\mathbf{d}_i$
end for
return $\mathbf{a}_N$

We train a goal-conditioned diffusion policy $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$ by applying a dropout rate of 0.1 to the goal $\boldsymbol{g}$, which also trains an implicit goal-independent policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ within our goal-conditioned model. The generation process uses a combined gradient for the denoising process

$$\begin{aligned} \nabla_a \log p_{t,\lambda}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}) = \\ \lambda \nabla_a \log p_t(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}) + (1-\lambda)\nabla_a \log p_t(\boldsymbol{a}|\boldsymbol{s}), \end{aligned} \tag{6}$$

where the guidance factor λ balances the influence of the goal-conditioned and goal-independent gradient. In diffusion literature, λ commonly ranges from 2 to 7.5, to guide the diffusion model towards goal-conditional distribution $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$. CFG has demonstrated significant performance improvements compared to other conditioning methods [14, 19, 28]. Even though CFG has also been successfully applied for generating state-only trajectories in Offline-RL [1], recent work on behavioral cloning suggests that CFG performs significantly worse than simpler conditioning methods [30] for step-based action generation. We provide a detailed analysis of CFG for goal-guided action generation in our experiment section.

A. Model Architecture

One of the main challenges of training the score-based diffusion model is the big range of noise levels $\sigma_t \in \{0.001, 40\}$ To address this challenge, we use an improved architecture [17] including additional skip-connections and two preconditioning layers, which are conditioned on the current noise level σ_t

$$\begin{aligned} D_\theta(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g}, \sigma_t) = \\ c_{\text{skip}}(\sigma_t)\boldsymbol{a} + c_{\text{out}}(\sigma_t)F_\theta(c_{\text{in}}(\sigma_t)\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, c_{\text{noise}}(\sigma_t)), \end{aligned} \tag{7}$$

The conditioning functions are described in detail in Section A of the Appendix and visualized in Figure 1.

These additional skip connections help the score model to scale the output to a wide range of noise levels σ_t, either by estimating the denoised sample $\boldsymbol{a}_{t-1}$, directly predicting the noise ϵ or something in between these two. Our proposed approach, BESO, integrates a Transformer-based architecture with causal masking as the inner model $F_\theta(\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{g}, \sigma_t)$. This enables our model to learn temporal relations between observations and actions, thereby improving its overall performance A detailed overview of our proposed architecture is shown in Figure 1. Three linear embedding layers encode the states $\boldsymbol{s}_n$, noise σ_t and the noisy actions $\boldsymbol{a}_n$ into a linear representation of the same dimension, $l_{\boldsymbol{s}}(\boldsymbol{s}), l_{\boldsymbol{a}}(\boldsymbol{a}), l_\sigma(\sigma)$. In addition, the position embedding information is added on the linear representations. The noise embedding is concatenated with the desired future states and all state-noise-action pairs in a large sequence for the model. During training, the denoised actions are inferred for all timesteps in the input series, yet only the last predicted action is utilized for inference. To take advantage of the causal masking in the transformer, we concatenate the goal-sequence before the current observation sequence [6], allowing for a sequence of goal-states.

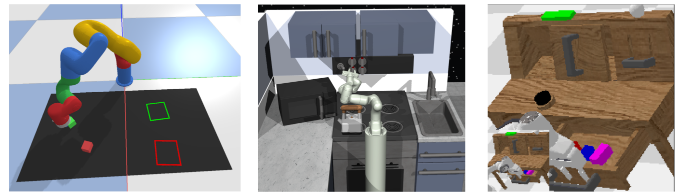

Fig. 2. Simulation environments for testing the performance of BESO: Multi-Modal Block-push (left); Relay Kitchen (middle); CALVIN (right)

V. Evaluation

The objective of our experiments was to answer the following key questions: **I**) Is BESO competitive on goal-conditioned environments against state-of-the-art baselines? **II**) What are the key components to enable fast sampling of Diffusion policies with good performance? **III**) Does Classifier-Free Guidance work for goal-conditional behavior synthesis? To answer these questions, we evaluated BESO on several challenging simulation benchmarks. First, we compared the performance of BESO against other state-of-the-art methods. Afterward, we examined BESO's components with respect to their contribution to the performance.

A. Baselines

We compare BESO against several state-of-the-art methods:

- **Goal-conditioned Behavior Cloning (GCBC)** learns a unimodal policy encoded as a simple multi-layer perceptron (MLP) with an trained with an MSE loss [21].
- **Relay Imitation Learning (RIL)** is a hierarchical policy, that learns a high-level sub-goal generator, which is used to condition a low-level MLP policy [12].
- **Latent Motor Plans (LMP)** is a hierarchical goal-conditioned policy, which consists of a seq2seq CVAE and an action decoder policy [21]. We use an adapted KL-weighting term and a transformer encoder, which has been shown to improve the performance of LMP [24].
- **Conditional Implicit Behavior Cloning (C-IBC)** uses an energy-based model as an implicit policy [10]. We

		GCBC	C-IBC	LMP	RIL	C-BeT	CX-Diff	C-BESO	CFG-BESO
Block-Push	Reward	0.13 (± 0.04)	0.46 (± 0.06)	0.04 (± 0.03)	0.06 (± 0.01)	0.91 (± 0.03)	0.93 (± 0.03)	*0.96* (± 0.02)	***0.97*** (± 0.02)
	Result	0.13 (± 0.04)	0.29 (± 0.10)	0.04 (± 0.03)	0.02 (± 0.01)	0.87 (± 0.07)	0.90 (± 0.04)	***0.93*** (± 0.02)	0.88 (± 0.04)
Relay-Kitchen	Reward	2.65 (± 0.25)	0.50 (± 0.09)	1.45 (± 0.22)	0.31 (± 0.15)	2.73 (± 0.28)	3.64 (± 0.14)	***3.98*** (± 0.07)	***3.98*** (± 0.07)
	Result	2.57 (± 0.26)	0.45 (± 0.08)	1.41 (± 0.22)	0.23 (± 0.11)	2.69 (± 0.28)	3.35 (± 0.15)	***3.75*** (± 0.08)	*3.47* (± 0.08)

TABLE I

MEAN AND STD ON THE CONDITIONED BLOCK-PUSH AND KITCHEN ENVIRONMENT, OVER 10 SEEDS WITH 100 RUNS EACH. C-BESO AND CFG-BESO CONSISTENTLY OUTPERFORMED ALL BASELINES, DESPITE ONLY USING 3 INFERENCE STEPS. CX-DIFF WITH 3 INFERENCE STEPS ACHIEVES A RESULT OF 2.74(±0.26) ON THE RELAY-KITCHEN. BOTH VARIANTS OF BESO SHOW A LOW DEVIATION ACROSS SEEDS, INDICATING THEIR ROBUSTNESS.

use a goal-conditioned extension of IBC to study the importance of the selected generative model architecture.

- **Conditional-Behavior Transformer (C-BeT)** is a GPT-like transformer-based policy, that predicts discrete action labels together with a continuous offset vector to learn multimodal behavior [35, 6]. The action labels are determined a priori via K-means clustering.
- **Diffusion-X (CX-Diff)** [30] is a DDPM [15] based policy with improved inference. It uses stochastic sampling and additional X-extra inference steps at the lowest noise level to synthesize actions in 50+X steps. While performing only slightly worse than the closely related KDE-Diff [30] it has a significantly lower computational cost.

To ensure a fair evaluation of all methods we kept the general hyperparameters, e.g., layer size and number, as consistent as possible while tuning the method-specific hyperparameters. A detailed summary of the baseline architectures and hyperparameters is provided in Sec. C of the Appendix. Additionally, we evaluated all models on the kitchen and block-push task with 10 seeds and 100 rollouts each. Given the high computational costs and time of training models for CALVIN, we restricted the tested methods to 3 seeds and limited the number of baselines.

B. Simulation Experiments

We evaluated BESO against the baselines on three simulation benchmarks, shown in Figure 2:

- **CALVIN Benchmark [25]:** We used the LfP benchmark, with a dataset consisting of 6 hours of unstructured play data. We restricted all methods to using a single static RBG image as observation input and predicting relative Cartesian actions as output [34]. We evaluated the methods on single tasks and 2 tasks in a row from a single goal image, both variants were conditioned on goal-images outside the training distribution, that did not contain the end-effector in the correct position.
- **Block-Push Environment [10]:** We used the adapted goal-conditioned variant [6]. The Block-Push Environment consists of an XARm robot that must push two blocks, a red and a green one, into a red and green squared target area. The dataset consists of 1000 demonstrations collected by a deterministic controller with 4 possible goal configurations. The methods got 0.5 credit for every block pushed into one of the targets with a maximum score of 1.0.
- **Relay Kitchen Environment [12]:** A multi-task kitchen environment with objects such as a kettle, door, and lights that the agent can interact with. The data consists of 566 human-collected trajectories with sequences of 4 executed skills. We used the same experiment settings as described in [6] to allow for fair comparisons. The models were evaluated using a pre-defined goal state, that consisted of 4 tasks for each rollout. Each correctly completed task gives 1 credit with a maximum of 4.

The methods were evaluated on two metrics: **result** evaluates how many of the desired goals of each rollout are achieved, while **reward** measures the overall performance by giving credit for reaching any goal defined in the environment.

C. Simulation Results

We compared BESO to the baselines on the Relay-Kitchen and Block-Push environments. The results are summarized in Table I. As shown in the table, BESO consistently outperformed the competitors on both tasks across 10 seeds. The low variance of BESO, additionally, indicates the robustness of our approach. Among the baselines, Diffusion-X and C-BeT perform well on the kitchen task and block-push environment, respectively. The diffusion policies excelled, outperforming all other baselines on the kitchen and the block-push task, whereas C-BeT demonstrated comparable performance on the block-push environment. Considering that BESO only used 3 denoising steps on both environments, compared to the $50(20) + 8$ steps of CX-Diff, makes BESO's performance even more impressive. By contrast, CX-Diff, when limited to 3 denoising steps, only managed an average result of $2.74(\pm 0.26)$ in the kitchen environment. This highlights the advantage of BESO's architecture combined with improved noise scheduling and sampler to achieve good results with only 3 denoising steps. On a modern desktop PC, BESO requires around 0.012 seconds to predict an action, while the CX-Diffusion model needs an average of 0.15 seconds. This makes BESO over 10 times faster.

In a more challenging simulation environment, the CALVIN environment, BESO demonstrated its ability to generalize to unseen goal states by achieving the best overall performance on 13 difficult single tasks. Each task was conditioned on a single goal image unseen during training, where the end-effector is not located near the corresponding task. This posed a significant challenge, as the models have to infer changes in the environmental state and perform the necessary

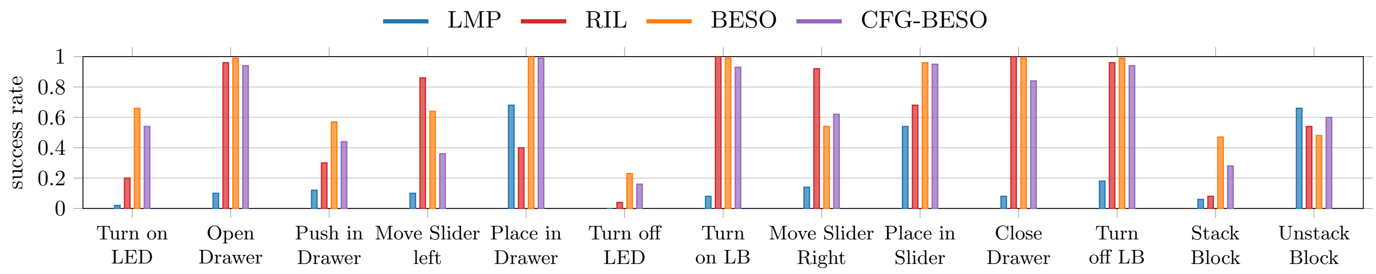

Fig. 3. The average Success rate of all tested models on executing single hard tasks in the CALVIN environment conditioned on a single goal image, that does not contain the end-effector of the robot near the required task.

tasks without relying on the position of the end-effector in the image for guidance. The results of this experiment are summarized in Figure 4 and the individual success rates of the tasks are summarized in Figure 3. As shown, BESO achieves the best overall performance on individual hard tasks, demonstrating its ability to also generalize to unseen goal-states. RIL is the second-best model and has a slightly better average performance on 2 tasks.

Additionally, the models were evaluated on solving two tasks with a single goal image. Similar to the first task, the end-effector was located at a different position away from both tasks. In this instance, BESO and its Classifier-Free Guidance (CFG) variant once again outperformed other models, though the CFG variant registered a slightly lower performance. The results illustrate that BESO can effectively learn meaningful behavior to solve downstream short-term and long-term goals by learning from random windows of play trajectories. This further supports the conclusion that BESO's ability to learn multimodal and expressive action distributions is key for effective learning from play. In addition, this experiment showcases BESO's proficiency in effectively from visual data. Overall, our results indicate that BESO is competitive against state-of-the-art baselines and capable of effectively learning from play data, making it a promising approach for goal-conditioned behavior learning. Hence, we can answer Question **I**) in the affirmative.

D. BESO design choices

We answer Question **II** by evaluating different components of BESO to study their contribution to the overall performance.

Conditioning Method. First, we evaluated different methods to condition the behavior generation on the desired goal state. We tested the FiLM-conditioning [31] and the sequential conditioning method used in C-BeT [6]. FiLM requires additional MLP models, which input the goal and scale the latent representations inside the transformer layers. The sequential conditioning method simply includes desired goal-states at the beginning of our sequence as depicted in the model overview of Figure 1. We tested both conditioning variants using the same transformer score model and evaluated it on the block-push and kitchen environment on 10 seeds. FiLM conditioning

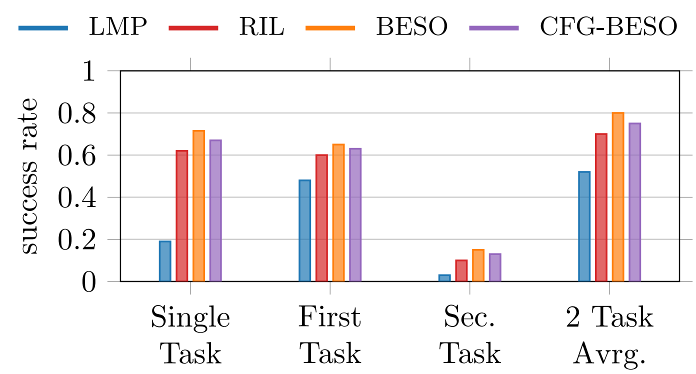

Fig. 4. The average performance of goal-conditioned policy on the CALVIN environment. The first column shows the average success rate of 13 individual tasks. The other three columns show the average success rate of all models conditioned on a single goal image with 2 tasks.

resulted in a performance drop compared to the sequential conditioning method from an average result of 0.93 to 0.91 and 3.76 to 3.4 on the block-push and kitchen environment respectively. Moreover, the FiLM method increase the overall model capacity. Hence, BESO uses the sequential conditioning method.

Sampling Algorithm. BESO generates actions by numerically approximating the reverse ODE with its learned score-model starting from a sample generated from our Gaussian prior distribution p_T. We investigated several numerical sampling algorithms used in diffusion research, such as DDIM [38], DPM [19], DPM++ [20], and Heun [17], to assess their contribution to BESO's performance. The samplers were evaluated on the block-push and kitchen environments with different number of denoising steps. The results show that the performance gap between the individual samplers is small, with DDIM achieving the best overall performance. Surprisingly, the second-order Heun solver has the worst average performance. Detailed results of this experiment are summarized in Table VII and Table VIII in the Appendix. Overall BESO is robust to the number of sampling steps and chosen sampler type, maintaining a similar performance from 3 to 50 inference steps.

		Deterministic	Stochastic
Block-Push	Reward	0.97 ($\pm$ 0.02)	0.97 ($\pm$ 0.02)
	Result	0.93 ($\pm$ 0.02)	0.92 ($\pm$ 0.03)
Relay-Kitchen	Reward	3.95 ($\pm$ 0.10)	4.03 ($\pm$ 0.07)
	Result	3.73 ($\pm$ 0.11)	3.80 ($\pm$ 0.08)
CALVIN	Hard Tasks	0.71 ($\pm$ 0.01)	0.68 ($\pm$ 0.03)
	2 Tasks	0.79 ($\pm$ 0.03)	0.79 ($\pm$ 0.02)

TABLE II
EVALUATION OF THE INFLUENCE OF NOISE INJECTION FOR GOAL-CONDITIONAL BEHAVIOR GENERATION AVERAGED OVER 2 SAMPLERS WITH AND WITHOUT RANDOM NOISE INJECTION.

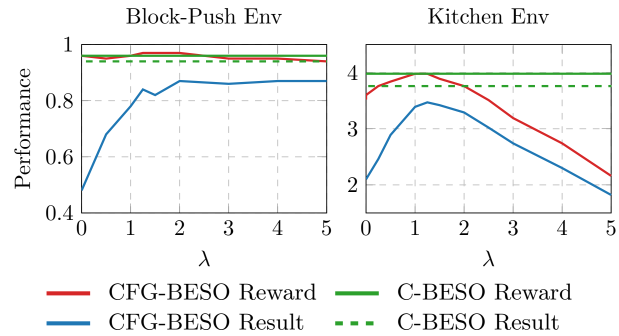

Fig. 5. Comparison of CFG Method for Goal-conditioned Behavior Learning from Play Data. For CFG-BESO we evaluate the 10 seeds on 100 rollouts each with different λ values. The CFG variant of BESO has a slightly worse average result in both environments with similar rewards. When using $\lambda = 0$, we can recover an unconditional policy, that performs random rollouts with high rewards and low results.

Stochastic vs. Deterministic Sampling. Current diffusion literature supports the assumption that stochastic samplers have a better overall performance compared to deterministic samplers [17, 41]. We tested this assumption with respect to step-based action generation. We evaluated the same models with 2 sampling algorithms DPM++(2S) and the Euler sampler [19, 17], each with and without noise injection. The noise scheduling was performed via the ancestral sampling strategy, as used in the DPPM variant [15, 41] and described in Alg. C. Experiments were again conducted in all environments. As shown in Table II, the results suggest that the addition of noise does not offer a significant benefit to the action generation of step-based diffusion policies. Stochastic samplers only increase the average performance in the kitchen environment. The discrepancy compared to common diffusion applications such as image synthesis [7] could be rooted in high-dimensional image spaces, making the generation process more difficult and requiring more steps for good results. In these high-dimensional spaces, errors are more likely to occur and accumulate over time. Adding noise during the inference process helps the model to correct errors of the gradient approximation, resulting in a better overall performance [17]. In contrast, step-based action-distributions are significantly lower dimensional than the high-dimensional latent spaces of image generation, hence, the addition of noise does not appear to benefit the average performance of step-based policies, as supported by our experimental results.

Classifier-Free-Guidance (CFG). Finally, we investigate Question **III** by evaluating the effect of Classifier Free Guidance (CFG) for step-based action generation with goal-conditioned policies. The results of this experiment, reported in Figure 5, indicate that CFG is an effective method for goal-conditioning in a step-based setting. The average result for the block-push and kitchen tasks is slightly worse than the standard goal-conditioned variant, while the average reward is equal. CFG-BESO is also able to learn effectively in the image-based CALVIN environment and achieves similar performance to the standard goal-conditioned variant. The performance of the CFG-model with $\lambda = 0$ demonstrates, that CFG-BESO is capable of learning a well-performing, unconditional policy $\pi(\boldsymbol{a}|\boldsymbol{s})$. The low average result in Figure 5 shows that the policy ignores the goal-state and aims to achieve a high reward solely based on the current state. This gives CFG-BESO a unique advantage over common play-based policies. However, CFG has a trade-off: it slightly lowers the average result for more diverse rollouts. Empirical evaluations suggest the best λ value is 1.25 for most tested environments. Experiments with higher values resulted in a lower average performance in environments with high-dimensional action spaces, indicating instability in the action generation. We hypothesize that the guidance provided by the goal-conditioning is only crucial in certain steps during the rollouts, specifically when the policy is deciding which task to solve.

VI. Conclusion

We introduced BESO, a new policy representation for goal-conditioned behavior generation that uses score-based diffusion models. We leveraged the expressiveness and multimodal properties of score-based diffusion models to learn task-agnostic behavior from offline, reward-free play datasets, without requiring hierarchical structures or additional clustering. In addition, we demonstrated the effectiveness of Classifier-Free Guidance for simultaneously learning a goal-dependent and goal-independent policy in a sequential setting. Experiments on several GCIL benchmarks showed that BESO significantly improves upon several state-of-the-art GCIL algorithms. Our ablation studies have demonstrated the key components of BESO that enable fast, deterministic behavior generation. It further outperformed standard DDPM policies with only 3 denoising steps, alleviating prior drawbacks of slow diffusion sampling.

While BESO demonstrates great performance as a standalone policy, it also offers the flexibility to be seamlessly integrated into other hierarchical frameworks as an action prediction policy. Serving as a practical alternative to traditional behavior cloning policies, BESO sets itself apart with distinct features that are inherent to diffusion models. In the future, we aim to extend BESO for language-guided behavior generation, offering more intuitive goal guidance for humans.

VII. Acknowledgments

The work presented here was funded by the German Research Foundation (DFG) – 448648559.

References

[1] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=sP1fo2K9DFG.

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

[3] Suneel Belkhale and Dorsa Sadigh. PLATO: Predicting latent affordances through object-centric play. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=UAA5bNospA0.

[4] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[5] 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.

[6] Zichen Jeff Cui, Yibin Wang, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. From play to policy: Conditional behavior generation from uncurated robot data. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=c7rM7F7jQjN.

[7] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

[8] Benjamin Eysenbach, Soumith Udatha, Ruslan Salakhutdinov, and Sergey Levine. Imitating past successes can be very suboptimal. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=iqCO3jbPjYF.

[9] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=vGQiU5sqUe3.

[10] 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.

[11] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkHywl-A-.

[12] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019.

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

[14] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[16] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

[17] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=k7FuTOWMOc7.

[18] Kuno Kim, Akshat Jindal, Yang Song, Jiaming Song, Yanan Sui, and Stefano Ermon. Imitation with neural density models. *Advances in Neural Information Processing Systems*, 34:5360–5372, 2021.

[19] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*, 2022.

[20] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=2uAaGwlP_V.

[21] 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.

[22] Yecheng Jason Ma, Jason Yan, Dinesh Jayaraman, and Osbert Bastani. Offline goal-conditioned reinforcement learning via f-advantage regression. In *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_h29VprPHD.

[23] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. GTI: Learning to Generalize across Long-Horizon Tasks from Human Demonstrations. In *Proceedings of Robotics: Science and Systems*, July 2020. doi: 10.15607/RSS.2020.XVI.061.

[24] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and*

Automation Letters (RA-L), 7(4):11205–11212, 2022.

[25] 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*, 2022.

[26] Lina Mezghani, Sainbayar Sukhbaatar, Piotr Bojanowski, Alessandro Lazaric, and Karteek Alahari. Learning goal-conditioned policies offline with self-supervised reward shaping. In *CoRL-Conference on Robot Learning*, 2022.

[27] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[28] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob Mcgrew, Ilya Sutskever, and Mark Chen. GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16784–16804. PMLR, 17–23 Jul 2022.

[29] 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.

[30] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, and Sam Devlin. Imitating human behaviour with diffusion models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Pv1GPQzRrC8.

[31] 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.

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

[33] 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.

[34] Erick Rosete-Beas, Oier Mees, Gabriel Kalweit, Joschka Boedecker, and Wolfram Burgard. Latent plans for task-agnostic offline reinforcement learning. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=ViYLaruFwN3.

[35] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. In *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=agTr-vRQsa.

[36] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations*, 2020.

[37] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[38] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.

[39] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

[40] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020.

[41] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020.

[42] Guy Tevet, Sigal Raab, Brian Gordon, Yoni Shafir, Amit Haim Bermano, and Daniel Cohen-or. Human motion diffusion model. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SJ1kSyO2jwu.

[43] Julen Urain, Niklas Funk, Jan Peters, and Georgia Chalvatzaki. Se(3)-diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion. *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[44] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011. doi: 10.1162/NECO_a_00142.

[45] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=AHvFDPi-FA.

[46] Rui Yang, Yiming Lu, Wenzhe Li, Hao Sun, Meng Fang, Yali Du, Xiu Li, Lei Han, and Chongjie Zhang. Rethinking goal-conditioned supervised learning and its connection to offline rl. In *International Conference on Learning Representations*, 2021.

[47] Sarah Young, Jyothish Pari, Pieter Abbeel, and Lerrel Pinto. Playful interactions for representation learning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 992–999, 2022. doi: 10.1109/IROS47612.2022.9981307.

Hyperparameter	Block-Push	Relay-Kitchen	CALVIN
Hidden Dimension	240	360	768
Hidden Layers	4	6	6
Window size	5	5	2
Goal window size	1	2	1
Number Heads	12	6	4
Attention Dropout	0.05	0.3	0.2
Residual Dropout	0.05	0	0.1
Learning rate	1e-4	1e-4	1e-4
Optimizer	Adam	Adam	AdamW
Denoising steps	3	3	5
σ_{max}	1	1	1
σ_{min}	0.05	0.005	0.005
σ_{data}	0.5	0.5	0.5
best λ for CFG	2	1.25	1.25
Noise distribution	Log-Logistic	Log-Logistic	Log-Logistic
EMA	True	True	True
Vision Encoder	None	None	ResNet-18
Sampler Type	DDIM	DDIM	DDIM
Noise scheduler	Exp	Exp	Exp
Batch Size	1024	1024	64
Train steps in thousand	60	40	120

TABLE III
OVERVIEW OF THE MOST IMPORTANT HYPERPARAMETERS FOR THE DIFFERENT MODEL ARCHITECTURES FOR ALL TESTED ENVIRONMENTS.

APPENDIX

A. BESO Hyperparameters

A summary of key hyperparameters of BESO is listed in Table III. We observe, that transformer specific-hyperparameters such as the dropout rates require tuning according to the task, while general diffusion hyperparameters remain consistent across different tasks.

Preconditioning. We utilize the preconditioning functions proposed in Karras et al. [17]:

- $c_{\text{skip}} = \sigma_{\text{data}}^2 / (\sigma_{\text{data}}^2 + \sigma_t^2)$
- $c_{\text{out}} = \sigma_t \sigma_{\text{data}} / \sqrt{\sigma_{\text{data}}^2 + \sigma_t^2}$
- $c_{\text{in}} = 1 / \sqrt{\sigma_{\text{data}}^2 + \sigma_t^2}$
- $c_{\text{noise}} = 0.25 \ln(\sigma_t)$

Normalization. BESO performs optimally when actions are diffused within a range of $[-1, 1]$ with a noise range of $0.005, 1$. We adopted this noise range for all three environments, scaling the action output accordingly. For action diffusion with a larger range of, such as $[-3, 3]$, it is advisable to expand the noise range to higher values: $\{0.4, 40\}$ for optimal performance. For the input, we recommend normalizing the data with a mean of 0 and a standard deviation of 1.

Training Noise distribution. During training, noise values are sampled from a predefined noise distribution $P(\sigma)$. The standard distribution used in diffusion literature [17] is the log-normal, introducing two additional hyperparameters $\sigma_{\text{std}}, \sigma_{\text{max}}$, that require additional tuning. Our experiments revealed that the recommended values from prior work [17] are not optimal for action diffusion. Hence, we opted for the log-logistic distribution $\text{LogLogistic}(\alpha = 0.5, \beta = 0.5)$, which does not require additional parameters and works well in all our experiments.

	Block-Push		Relay Kitchen	
	Reward	Result	Reward	Result
Exponential	0.96 ($\pm$ 0.02)	0.93 ($\pm$ 0.02)	3.86 ($\pm$ 0.09)	3.74 ($\pm$ 0.09)
Linear	0.97 ($\pm$ 0.01)	0.94 ($\pm$ 0.01)	3.98 ($\pm$ 0.08)	3.76 ($\pm$ 0.10)
IDDPM [27]	0.95 ($\pm$ 0.01)	0.92 ($\pm$ 0.01)	3.88 ($\pm$ 0.08)	3.67 ($\pm$ 0.09)
Karras [17]	0.96 ($\pm$ 0.02)	0.93 ($\pm$ 0.03)	3.96 ($\pm$ 0.08)	3.75 ($\pm$ 0.07)
VE [41]	0.95 ($\pm$ 0.01)	0.93 ($\pm$ 0.01)	3.98 ($\pm$ 0.09)	3.75 ($\pm$ 0.10)
VP [41]	0.64 ($\pm$ 0.17)	0.61 ($\pm$ 0.16)	3.21 ($\pm$ 0.12)	2.95 ($\pm$ 0.14)

TABLE IV
EVALUATION OF THE INFLUENCE OF THE TIME STEPS FUNCTION FOR GOAL-CONDITIONAL BEHAVIOR GENERATION AVERAGED OVER 10 SEEDS AND 100 ROLLOUTS EACH. ALL MODELS USE THE DDIM SOLVER WITH 3 DENOSING STEPS FOR THE COMPARISON.

Optimization. For optimization, we employed the commonly used Adam or AdamW optimizer for our experiments with a standard learning rate of $1e{-}4$. Additionally, we use the Exponential Moving Average (EMA) to optimize our model's weights.

Time steps. One important choice is the function of time steps, which determines how noise levels are distributed over the discrete steps. Our empirical evaluation summarized in Table IV indicates that exponential time steps are the most effective for BESO on average. However, other discretization methods such as the linear scheduler [41] and Karras scheduler [17] also deliver comparable results and can increase the performance on individual tasks.

Recommendations. We recommend starting with the noise range of $\{0.005, 1\}$ for a new task together with exponential time steps and the DDIM solver. To get the best performance, it is worth trying out other samplers such as Euler Ancestral and the linear time steps.

B. Sampler Ablation

We evaluate various state-of-the-art ODE samplers and their SDE counterparts in different environments. To determine the best solver for conditional-behavior generation, we analyze the average performance of 10 different seeds with 100 rollouts each in different environments. In general, we differentiate first-order and second-order solvers: the first order solver is Euler [15] and the tested second order solver is Heun [17]. The tested samplers include:

- **Euler ODE (Euler):** A first-order ODE sampler from [17] without the additional addition and deleting of noise. The algorithm is summarized in C.
- **Euler-Ancestral (EA):** A continuous-time version of the standard DDPM sampler [15] introduced in [41].
- **2nd Order Heun Solver (Heun):** A second-order ODE solver using the Heun method [17].
- **DPM:** An exponential ODE integrator solver designed for synthesis in a few inference steps [20]. We use the second order method.
- **DDIM:** A first order variant of DPM, which has been introduced individually [38, 20] and has been designed for fast inference and CFG.
- **DPM-Ancestral:**A stochastic variant of DPM with ancestral noise injections.
- **DPM++(2S):** An improved version of the second order

DPM sampler for classifier-free guidance based conditional diffusion models with a single inference step [19]

- **DPM++(2M):** An improved version of the second-order DPM sampler for classifier-free guidance based conditional diffusion models [19], which is a second order method using two model predictions per step.

Several previous studies have compared the performance of ODE samplers in the context of image generation [17, 19]. However, these comparisons may not be entirely indicative as image generation tasks have unique challenges and requirements not relevant to action synthesis. To ensure a fair comparison, we evaluated all samplers on the same models across several simulation environments and report their average performance based on 100 runs for each environment. This allows us to accurately compare the effectiveness of each deterministic solver in the context of step-based action generation. The results for the kitchen environment are shown in Table VII and the performance for the block push is reported in Table VIII. As shown in both tables, the first-order exponential integrator solver DDIM achieves the best overall performance. Increasing the number of inference steps does not have a significant impact on the average performance, even reducing the average result of some samplers. Overall the performance differences of all evaluated samplers are small.

Hyperparameter	Block Push	Relay Kitchen
Hidden dimension	128	128
Hidden layers	6	6
Train steps	5000	1000
Noise Scale	0.3	0.3
Loss	InfoNCE	InfoNCE
Train samples	64	64
Noise shrink	0.5	0.5
Learning rate	0.001	0.001

TABLE V

OVERVIEW OF THE USED HYPERPARAMETERS OF GC-IBC FOR BOTH ENVIRONMENTS.

Hyperparameter	Block-Push	Relay-Kitchen	CALVIN
Decoder Hidden Dimension	{128, 256, 512, **1024**}	{128, 256, 512}	2048
n-Mixtures	10	10	10
n-Classes	{10, 32, 64, 128, **256**}	10	10
Policy-dropout	{0.1, **0.2**, 0.3}	{0.1, **0.2**, 0.3}	0.1
Plan Features	{16, **32**, 64, 128}	{16, 32, **64**, 128}	32
Plan Recognition Features	{64, 128, **256**, 512}	{64, 128, 256, **512**}	2048
Replan Freq	{5, **10**, 16, 32}	{5, 10, 16, **32** }	2
Planner Hidden Layers	2	2	2
Window size	{10, 16, 32, **48**}	{10, **16**, 32, 48, 64}	16
Goal window size	1	1	1
kl-beta	{0.001, 0.005, **0.01**}	{0.001, 0.005, **0.01**}	0.01
Learning rate	{0.001, **0.0005**, 0.0001}	{0.001, **0.0005**, 0.0001}	0.0001
Optimizer	Adam	Adam	Adam

TABLE VI

OVERVIEW OF THE HYPERPARAMETER-SWEEP FOR LATENT PLANS AND THE FINAL PARAMETERS USED FOR THE EVALUATION FOR EACH TESTED SIMULATION ENVIRONMENT

C. Baselines Implementation

The MLP-based models have 4 layers with 512 neurons and use the ReLU activation function. All diffusion models have the same transformer backbone, and C-BeT uses its recommended parameters. During training, the Adam optimizer was used with a learning rate of 0.001 for MLP models and $1e-4$ for transformer models. The batch size for MLP models was 512, while it was 1024 for transformer models, except for BeT, which used a batch size of 64 as recommended in [6].

GCBC For the GCBC model, the goal is concatenated with the state and fed into the 4-layer MLP architecture with a dropout rate of 0.1.

GC-IBC The GC-IBC model uses the same MLP architecture as GCBC and is optimized using the InfoNCE loss with additional energy-regularization and Wasserstein Gradient loss. During experiments, adding a penalty term with $\lambda = 0.005$ to restrict the average energy improved training stability [10]. Given the large number of tunable hyperparameters for IBC, we ran a hyperparameter search to determine the best ones. We want to note, that the model results of EBM were very sensitive to initial seeds and we had trouble getting consistent results for the models. Similar observations of IBC performance have been reported in related work [30, 5].

C-BeT For the performance of C-BeT, we use the recommended parameters from Cui et al. [6] for all tested environments. Our reported results are marginally worse, than the ones reported in the original work, since they do not average it over 10 seeds.

Latent Motor Plans The LMP model was evaluated on the Kitchen and Block Push environments with extensive hyperparameter sweeps to find the best-performing configuration. A detailed overview of the sweep parameters and the chosen ones is shown in Table VI. On the CALVIN environment, the proposed parameters from prior work were used [34]. We used the improved LMP variant, called HULC, from [24], which uses a different Kl-divergence weighting term and a transformer model the Seq2Seq CVAE.

RIL For the low-level policy of kitchen and block push we use 4 layers with 512 neurons each. For the CALVIN task, we use the baseline version from [34] and kept the hyperparameters the same for training.

Diffusion-X The baseline from [30] uses the same hyperparameters of our transformer model reported in III to guarantee a fair comparison. Diffusion-X uses 50 inference steps on the kitchen task combined with additional 10 fine-tuning steps at the lowest noise level, while we use 20 inference steps for the block-push environment and additional 8 fine-tuning steps. Diffusion-X uses a discrete variant of the Euler sampling method with an ancestral noise scheduler, which is reported in Alg. C [15, 41]. Further, it applies X-additional denoising steps at the lowest noise level.

Algorithm 3 Ancestral Noise Scheduler f_{ANC} [41, 15]

1: **Require:** t_{from}, t_{to}

2: $t_{\text{up}} \leftarrow \min(t_{\text{to}}, \sqrt{\frac{t_{\text{to}}^2(t_{\text{to}}^2 - t_{\text{from}}^2)}{t_{\text{from}}^2}})$

3: $t_{\text{to}} \leftarrow \sqrt{(t_{\text{to}}^2 - t_{\text{from}}^2)}$

4: **return** t_{down}, t_{up}

	Steps	Euler	Heun	DDIM	DPM	DPM++(2S)	DPM++(2M)
Reward	3	3.87 (± 0.09)	3.80 (± 0.03)	3.92 (± 0.07)	3.86 (± 0.08)	3.88 (± 0.08)	3.89 (± 0.08)
	5	3.87 (± 0.07)	3.84 (± 0.06)	3.88 (± 0.06)	3.90 (± 0.09)	3.87 (± 0.06)	3.87 (± 0.06)
	10	3.85 (± 0.08)	3.86 (± 0.09)	3.88 (± 0.06)	3.87 (± 0.10)	3.88 (± 0.07)	3.89 (± 0.05)
	20	3.86 (± 0.10)	3.91 (± 0.08)	3.87 (± 0.07)	3.88 (± 0.09)	3.89 (± 0.07)	3.88 (± 0.06)
	50	3.82 (± 0.08)	3.93 (± 0.04)	3.88 (± 0.06)	3.82 (± 0.10)	3.67 (± 0.04)	3.89 (± 0.06)
Result	3	3.66 (± 0.09)	3.62 (± 0.07)	3.69 (± 0.07)	3.67 (± 0.10)	3.67 (± 0.09)	3.67 (± 0.08)
	5	3.66 (± 0.07)	3.66 (± 0.06)	3.67 (± 0.08)	3.67 (± 0.08)	3.66 (± 0.07)	3.66 (± 0.08)
	10	3.65 (± 0.06)	3.63 (± 0.06)	3.67 (± 0.07)	3.66 (± 0.07)	3.66 (± 0.08)	3.67 (± 0.07)
	20	3.64 (± 0.07)	3.65 (± 0.09)	3.66 (± 0.09)	3.66 (± 0.07)	3.68 (± 0.09)	3.67 (± 0.08)
	50	3.62 (± 0.04)	3.67 (± 0.04)	3.67 (± 0.09)	3.62 (± 0.07)	3.67 (± 0.07)	3.67 (± 0.08)

TABLE VII

Comparison of the performance of deterministic samplers on the Kitchen Environment averaged over 10 seeds with 100 rollouts each. We tested BESO trained on log-normal noise distribution with $\sigma_{\text{MEAN}} = -2$, $\sigma_{\text{STD}} = -2$, $\sigma_{\text{MAX}} = 33$, $\sigma_{\text{MIN}} = 0.39$ and use the exponential time steps.

	Steps	Euler	Heun	DDIM	DPM	DPM++(2S)	DPM++(2M)
Reward	3	0.95 (± 0.02)	0.92 (± 0.02)	0.96 (± 0.02)	0.96 (± 0.02)	0.95 (± 0.03)	0.97 (± 0.02)
	5	0.94 (± 0.04)	0.95 (± 0.02)	0.96 (± 0.02)	0.97 (± 0.01)	0.94 (± 0.02)	0.93 (± 0.02)
	10	0.97 (± 0.03)	0.93 (± 0.02)	0.96 (± 0.01)	0.95 (± 0.03)	0.96 (± 0.02)	0.96 (± 0.03)
	20	0.98 (± 0.02)	0.96 (± 0.03)	0.98 (± 0.02)	0.96 (± 0.03)	0.96 (± 0.02)	0.97 (± 0.03)
	50	0.98 (± 0.01)	0.96 (± 0.01)	0.97 (± 0.02)	0.97 (± 0.05)	0.97 (± 0.01)	0.94 (± 0.05)
Result	3	0.94 (± 0.02)	0.90 (± 0.05)	0.94 (± 0.04)	0.94 (± 0.01)	0.92 (± 0.03)	0.94 (± 0.03)
	5	0.91 (± 0.06)	0.93 (± 0.03)	0.95 (± 0.02)	0.95 (± 0.02)	0.91 (± 0.03)	0.93 (± 0.03)
	10	0.94 (± 0.02)	0.91 (± 0.04)	0.95 (± 0.02)	0.91 (± 0.04)	0.94 (± 0.02)	0.96 (± 0.02)
	20	0.96 (± 0.02)	0.94 (± 0.03)	0.95 (± 0.04)	0.95 (± 0.04)	0.93 (± 0.03)	0.96 (± 0.03)
	50	0.98 (± 0.01)	0.95 (± 0.02)	0.95 (± 0.01)	0.93 (± 0.03)	0.94 (± 0.03)	0.92 (± 0.06)

TABLE VIII

Comparison of the performance of deterministic samplers on the Block Push environment averaged over 10 seeds with 100 rollouts each. We tested BESO trained on log-normal noise distribution with $\sigma_{\text{MEAN}} = -0.17$, $\sigma_{\text{STD}} = -2$, $\sigma_{\text{MAX}} = 40.5$, $\sigma_{\text{MIN}} = 0.39$ and use the exponential time steps.

Algorithm 4 Deterministic 1st Order Euler Sampler [17]

Require: Current state $\boldsymbol{s}$, goal $\mathbf{g}$
Require: Score-Denoising Model $D_\theta(\mathbf{a}, \boldsymbol{s}, \mathbf{g}, \sigma)$
Require: Noise scheduler $\sigma_t = \sigma(t_i)$
Require: Discrete time steps $t_{i \in \{0,..,N\}}$
Draw sample $\mathbf{a}_0 \sim \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I})$
for $i \in \{0, ..., N-1\}$ **do**
  $\mathbf{d}_i \leftarrow \big(\mathbf{a}_i - D_\theta(\mathbf{a}_i, \boldsymbol{s}, \mathbf{g}, \sigma_i)\big)/\sigma_i$
  $\mathbf{a}_{i+1} \leftarrow \mathbf{a}_i + (t_{i+1} - t_i)\mathbf{d}_i$
end for
return $\mathbf{a}_N$

Algorithm 5 Stochastic 1st Order Euler sampler [17]

Require: Current state $\boldsymbol{s}$, goal $\mathbf{g}$
Require: Score-Denoising Model $D_\theta(\mathbf{a}, \boldsymbol{s}, \mathbf{g}, \sigma)$
Require: Noise scheduler $\sigma_i = t_i$, f_{ANC} from Alg. C
Require: Discrete time steps $t_{i \in \{0,..,N\}}$
Draw sample $\mathbf{a}_0 \sim \mathcal{N}(\mathbf{0}, \sigma_0^2 \mathbf{I})$
for $i \in \{0, ..., N-1\}$ **do**
  $\mathbf{d}_i \leftarrow \big(\mathbf{a}_i - D_\theta(\mathbf{a}_i, \boldsymbol{s}, \mathbf{g}, \sigma_i)\big)/\sigma_i$
  $t_{\text{down}}, t_{\text{up}} \leftarrow f_{\text{ANC}}(t_i, t_{i+1})$
  $\mathbf{a}_{i+1} \leftarrow \mathbf{a}_i + (t_{\text{down}} - t_i)\mathbf{d}_i$
  $\epsilon_{\text{up}} \sim \mathcal{N}(\mathbf{0}, \sigma_{t_{\text{up}}}^2 \mathbf{I})$
  $\mathbf{a}_{i+1} \leftarrow \mathbf{a}_{i+1} + \epsilon_{\text{up}}$
end for
return $\mathbf{a}_N$